cna: An R Package for Configurational Causal Inference and Modeling

Michael Baumgartner
University of Bergen, Norway

Mathias Ambühl
Consult AG, Switzerland

Abstract

The R package **cna** provides comprehensive functionalities for causal inference and modeling with *Coincidence Analysis* (CNA), which is a configurational comparative method of causal data analysis. In this vignette, we first review the theoretical and methodological background of CNA. Second, we introduce the data types processable by CNA, the package's core analytical functions with their arguments, and some auxiliary functions for data simulations. Third, CNA's output along with relevant fit parameters and output attributes are discussed. Fourth, we provide guidance on how to interpret that output and, in particular, on how to proceed in case of model ambiguities. Finally, some considerations are offered on benchmarking the reliability of CNA.

Keywords: configurational comparative methods, set-theoretic methods, Coincidence Analysis, Qualitative Comparative Analysis, INUS causation, Boolean causation.

1. Introduction

Coincidence Analysis (CNA) is a configurational comparative method of causal data analysis that was introduced for crisp-set (i.e. binary) data in (Baumgartner 2009a; 2009b; 2015) and substantively extended, reworked, and generalized for multi-value and fuzzy-set data in (Baumgartner and Ambühl 2020). In recent years, CNA has been applied in numerous studies across the social, political, and behavioral sciences, with a particularly rapid uptick in usage in public health, covering a wide range of topics such as colorectal cancer screening, patient safety in nursing homes, implementation of Hepatitis C virus treatments, drug withdrawal, COVID-19 vaccination rates, or the connection between firearm laws and homicide rates.[1] In contrast to more standard methods of data analysis, which primarily quantify effect sizes, CNA belongs to a family of methods designed to group causal influence factors conjunctively (i.e. in complex bundles) and disjunctively (i.e. on alternative pathways). It is firmly rooted in a so-called regularity theory of causation and it is the only method of its kind that can recover causal structures with multiple outcomes (effects), for example, causal chains.

Many disciplines investigate causal structures with one or both of the following features: (i) causes are arranged in complex bundles that only become operative when all of their components are properly co-instantiated, each of which in isolation is ineffective or leads to different outcomes, and (ii) outcomes can be brought about along alternative causal routes

[1] The Zotero CNA library provides detailed references to more than 70 applications of CNA. Among other impacts, CNA has been showcased in the flagship journal of implementation science (Whitaker *et al.* 2020).

#	S	C	F
c_1	1	1	1
c_2	0	0	1
c_3	1	0	0
c_4	0	1	0

(a)

	S	C	F
S	1.00	0.00	0.00
C	0.00	1.00	0.00
F	0.00	0.00	1.00

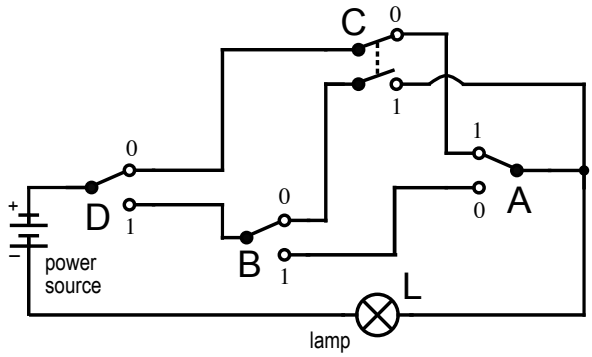
(b)

Table 1: Table (a) contains ideal configurational data, where each row depicts a different configuration of the factors S, C and F. Configuration c_1, for example, represents cases (units of observation) in which all factors take the value 1, whereas in c_2, S and C are 0 and F is 1, etc. Table (b) is the corresponding correlation matrix.

such that, when one route is suppressed, the outcome may still be produced via another one. For example, from a given set of implementation strategies available to medical facilities, some strategies yield a desired outcome (e.g. high uptake of treatment innovation) in combination with certain other strategies, whereas in different combinations the same strategies may have opposite or no effects (e.g. Yakovchenko *et al.* 2020). Or, a variation in a phenotype only occurs if many single-nucleotide polymorphisms interact, and various such interactions can independently induce the same phenotype (e.g. Culverhouse *et al.* 2002). Different labels are used for features (i) and (ii) in different disciplines: "interactions", "component causation", "conjunctural causation", "alternative causation", "equifinality", etc. For uniformity's sake, we will subsequently refer to (i) as *conjunctivity* and to (ii) as *disjunctivity* of causation, reflecting the fact that causes form conjunctions and disjunctions, that is, Boolean AND- and OR-connections.

Causal structures featuring conjunctivity and disjunctivity pose severe challenges for methods of causal data analysis. As many theories of causation entail that it is necessary (though not sufficient) for X to be a cause of Y that there be some kind of dependence (e.g. probabilistic or counterfactual) between X and Y, standard methods—for instance, regression and Bayesian network methods—infer that X is *not* a cause of Y if X and Y are *not* pairwise dependent (i.e. correlated). However, structures displaying conjunctivity and disjunctivity often do not exhibit pairwise dependencies. As a very simple illustration, consider the interplay between a person's skills to perform an activity, the challenges posed by that activity, and the actor's autotelic experience of complete involvement with the activity called *flow* (Csikszentmihalyi 1975). A binary model of this interplay involves the factors S, with values 0/1 representing low/high skills, C, with 0/1 standing for low/high challenges, and F, with 0/1 representing the absence/presence of flow. Csikszentmihalyi's (1975, ch. 4) flow theory entails that flow is triggered if, and only if, skills and challenges are either both high or both low, meaning that F=1 has the two alternative causes S=1 & C=1 and S=0 & C=0. If the flow theory is true, ideal (i.e. unbiased, unconfounded, noise-free) data on this structure feature the four configurations c_1 to c_4 in Table 1a, and no others. As can easily be seen from the corresponding correlation matrix in Table 1b, there are no pairwise dependencies. In consequence, standard methods will struggle to find the flow model, even when processing ideal data on it.

Although standard methods provide various protocols for tracing interaction effects involving two or three exogenous factors, these interaction calculations face tight computational complexity restrictions when more exogenous factors are involved and quickly run into multicollinearity issues (Brambor *et al.* 2006). Yet, structures with conjunctivity and disjunctivity



(a)

	A	*B*	*C*	*D*	*L*
A	1.00	0.00	0.00	0.00	0.00
B	0.00	1.00	0.00	0.00	0.00
C	0.00	0.00	1.00	0.00	0.00
D	0.00	0.00	0.00	1.00	0.26
L	0.00	0.00	0.00	0.26	1.00

(b)

	A	*B*	*D*	*C*	*L*
c_1	0	1	1	1	1
c_2	1	0	1	1	1
c_3	0	0	1	1	1
c_4	0	1	1	0	1
c_5	1	1	0	0	1
c_6	1	0	0	0	1
c_7	1	1	1	1	0
c_8	1	1	0	1	0
c_9	0	1	0	1	0
c_{10}	1	0	0	1	0
c_{11}	0	0	0	1	0
c_{12}	1	1	1	0	0
c_{13}	1	0	1	0	0
c_{14}	0	0	1	0	0
c_{15}	0	1	0	0	0
c_{16}	0	0	0	0	0

(c)

Figure/Table 2: Diagram (a) depicts a simple electrical circuit with three single-pole switches D, B, A, one double-pole switch C, and one lamp L. Table (c) comprises ideal data on that circuit and Table (b) the correlation matrix corresponding to that data.

may be much more complex than the flow model. Consider the electrical circuit in Figure 2a. It comprises a lamp L that can be on or off and four switches A to D, each of which can either be in position 1 or position 0. There are three alternative conjunctions of switch positions that close the circuit and cause the lamp to be on: A=0 & B=1 & D=1 OR A=1 & C=0 & D=0 OR B=0 & C=1 & D=1. As the switches are mutually independent, there are $2^4 = 16$ logically possible configurations of switch positions. For each of these configurations c_1 to c_{16}, Table 2c lists whether the lamp is on (L=1) or off (L=0). That table thus contains all and only the empirically possible configurations of the five binary factors representing the switches and the lamp. These are ideal data for the circuit in Figure 2a. Yet, even though all of the switch positions are causes of the lamp being on in some combination or other, factors A, B, and C are pairwise *independent* of L; only D is weakly correlated with L, as can be seen from the correlation matrix in Table 2b (which results from Table 2c). Standard methods of causal data analysis cannot infer the causal structure behind that circuit from Table 2c. They are not designed to group causes conjunctively and disjunctively.

A switch position as A=0 can only be identified as cause of L=1 by finding the whole conjunction of switch positions in which A=0 is indispensable for closing the circuit. More generally, discovering causal structures exhibiting conjunctivity and disjunctivity calls for a method that tracks causation as defined by a theory not treating a dependence between individual causes and effects as necessary for causation and that embeds values of exogenous factors in complex Boolean AND- and OR-functions over many other causes, fitting those functions as a whole to the data. But the space of Boolean functions over even a handful of factors is vast. For n binary factors there exist 2^{2^n} Boolean functions. For the switch positions in our circuit there exist 65536 Boolean functions; if we add only one additional binary switch that number jumps to 4.3 billion and if we also consider factors with more than two values that number explodes beyond controllability. That means a method capable of correctly discovering causal

structures with conjunctivity and disjunctivity must find ways to efficiently navigate in that vast space of possibilities. This is the purpose of CNA.

CNA takes data on binary, multi-value or fuzzy-set factors as input and infers causal structures as defined by the so-called *INUS theory* from it. The INUS theory was first developed by Mackie (1974) and later refined to the *MINUS theory* by Graßhoff and May (2001) (see also Baumgartner and Falk 2023b; Beirlaen *et al.* 2018). It defines causation in terms of redundancy-free Boolean dependency structures and, importantly, does not require causes and their outcomes to be pairwise dependent. As such, it is custom-built to account for structures featuring conjunctivity and disjunctivity.

CNA is not the only method for the discovery of (M)INUS structures. Other methods that can be used for that purpose are Logic Regression (Ruczinski *et al.* 2003; Kooperberg and Ruczinski 2005), which is implemented in the R package **LogicReg** (Kooperberg and Ruczinski 2023),[2] and Qualitative Comparative Analysis (QCA; Ragin 2008; Rihoux and Ragin 2009; Cronqvist and Berg-Schlosser 2009; Thiem 2018), implemented in the R packages **QCApro** (Thiem 2018) and **QCA** (Dusa 2024).[3] But CNA is the only method of its kind that can build models with more than one outcome and, hence, can analyze common-cause and causal chain structures as well as causal cycles and feedbacks. Moreover, unlike the models produced by Logic Regression or Qualitative Comparative Analysis, CNA's models are guaranteed to be redundancy-free, which makes them directly causally interpretable in terms of the (M)INUS theory; and CNA is more successful than any other method at exhaustively uncovering *all* (M)INUS models that fit the data equally well. For detailed comparisons of CNA with Qualitative Comparative Analysis and Logic Regression see (Baumgartner and Ambühl 2020; Swiatczak 2021) and (Baumgartner and Falk 2023a), respectively.

The **cna** package reflects and implements CNA's latest stage of development. This vignette provides a detailed introduction to **cna**. We first exhibit **cna**'s theoretical and methodological background. Second, we discuss the main inputs of the package's core function `cna()` along with numerous auxiliary functions for data review and simulation. Third, the working of the algorithm implemented in `cna()` is presented. Fourth, we explain `cna()`'s output along with relevant fit parameters and output attributes. Fifth, we provide some guidance on how to interpret that output and, in particular, on how to proceed in case of model ambiguities. Finally, some considerations are offered on benchmarking the reliability of `cna()`.

2. Background

The (M)INUS theory of causation belongs to the family of so-called *regularity theories*, which have roots as far back as Hume (1999 (1748)). It is a type-level theory of causation (cf. Baumgartner 2020) that analyzes the dependence relation of causal relevance between factors/variables taking on specific values, as in "X=χ is causally relevant to Y=γ". It assumes that causation is ultimately a deterministic form of dependence, such that whenever the same complete cause occurs the same effect follows. This entails that indeterministic behavior patterns in data result from insufficient control over background influences generating noise and not from the indeterministic nature of the underlying causal processes. For X=χ to be

[2] Another package implementing a variation of Logic Regression is **logicFS** (Schwender and Tietz 2024).

[3] Other useful QCA software include **QCAfalsePositive** (Braumoeller 2015) and **SetMethods** (Oana *et al.* 2023).

a (M)INUS cause of Y=γ, X=χ must be a difference-maker of Y=γ, meaning—roughly—that there exists a context in which other causes take constant values and a change from X≠χ to X=χ is associated with a change from Y≠γ to Y=γ.

To further clarify that theory as well as the characteristics and requirements of inferring (M)INUS structures from empirical data a number of preliminaries are needed.

2.1. Factors and their values

Factors are the basic modeling devices of CNA. They are analogous to (random) variables in statistics, that is, they are functions from (measured) properties into a range of values. They can be used to represent categorical properties that partition sets of units of observation (cases) either into two sets, in case of binary properties, or into more than two (but finitely many) sets, in case of multi-value properties, such that the resulting sets are exhaustive and pairwise disjoint. Factors representing binary properties can be *crisp-set* (*cs*) or *fuzzy-set* (*fs*); the former can take on 0 and 1 as possible values, whereas the latter can take on any (continuous) values from the unit interval $[0, 1]$. Factors representing multi-value properties are called *multi-value (mv) factors*; they can take on any of an open (but finite) number of non-negative integers.

Values of a *cs* or *fs* factor X can be interpreted as membership scores in the set of cases exhibiting the property represented by X. A case of type X=1 is a full member of that set, a case of type X=0 is a (full) non-member, and a case of type X=χ_i, $0 < \chi_i < 1$, is a member to degree χ_i. A case is considered a member of X if its membership score χ_i reaches the 0.5-anchor, that is, $\chi_i \geq 0.5$, and it is a non-member of X if $\chi_i < 0.5$. An alternative interpretation, which lends itself particularly well for causal modeling, is that "X=1" stands for the full presence of the property represented by X, "X=0" for its full absence, and "X=χ_i" for its partial presence (to degree χ_i). By contrast, the values of an *mv* factor X designate the particular way in which the property represented by X is exemplified. For instance, if X represents the education of subjects, X=2 may stand for "high school", with X=1 ("no completed primary schooling") and X=3 ("university") designating other possible property exemplifications. *Mv* factors taking on one of their possible values also define sets, but the values themselves must not be interpreted as membership scores; rather they denote the relevant property exemplification.

As the explicit "Factor=value" notation yields convoluted syntactic expressions with increasing model complexity, the **cna** package uses the following shorthand notation, which is standard in Boolean algebra (Bowran 1965): membership in a set is expressed by italicized upper case and non-membership by italicized lower case letters. "X" signifies membership in the set of cases exhibiting the property represented by X and "x" signifies non-membership in that set. Italicization thus carries meaning: "X" designates the factor and "X" membership in the set of cases with values of X above 0.5. In case of *mv* factors, value assignments to factors are not abbreviated but always written out, using the "Factor=value" notation.

2.2. Boolean operations

The (M)INUS theory defines causation using the Boolean operations of negation ($\neg X$, or x), conjunction ($X{*}Y$), disjunction ($X + Y$), implication ($X \rightarrow Y$), and equivalence ($X \leftrightarrow Y$).[4]

[4]Note that "$*$" and "$+$" are used as in Boolean algebra here, which means, in particular, that they do not

Inputs		Outputs				
X	Y	$\neg X$	$X{*}Y$	$X+Y$	$X \rightarrow Y$	$X \leftrightarrow Y$
1	1	0	1	1	1	1
1	0	0	0	1	0	0
0	1	1	0	1	1	0
0	0	1	0	0	1	1

Table 2: Classical Boolean operations applied to *cs* factors.

Negation is a unary truth function, the other operations are binary truth functions. That is, they take one resp. two truth values as inputs and output a truth value. When applied to *cs* factors, both their input and output set is $\{0, 1\}$. Negation is typically translated by "not", conjunction by "and", disjunction by "or", implication by "if ... then", and equivalence by "if and only if (iff)". Their classical definitions are given in Table 2.

These operations can be straightforwardly applied to *mv* factors as well, in which case they amount to functions from the *mv* factors' domain of values into the set $\{0, 1\}$. To illustrate, let both X and Y be ternary factors with values from the domain $\{0, 1, 2\}$. The negation of X=2, *viz.* $\neg$(X=2), then returns 1 iff X is not 2, meaning iff X is 0 or 1. X=2*Y=0 yields 1 iff X is 2 and Y is 0. X=2 + Y=0 returns 1 iff X is 2 or Y is 0. X=2 $\rightarrow$ Y=0 yields 1 iff X is not 2 or Y is 0. X=2 $\leftrightarrow$ Y=0 issues 1 iff either X is 2 and Y is 0 or X is not 2 and Y is not 0.

For *fs* factors, the classical Boolean operations must be translated into fuzzy logic. There exist numerous systems of fuzzy logic (for an overview cf. Hájek 1998), many of which come with their own renderings of Boolean operations. In the context of CNA (and QCA), the following fuzzy-logic rendering is standard: negation $\neg X$ is translated in terms of $1 - X$, conjunction $X{*}Y$ in terms of the minimum membership score in X and Y, i.e., $\min(X, Y)$, disjunction $X + Y$ in terms of the maximum membership score in X and Y, i.e., $\max(X, Y)$, an implication $X \rightarrow Y$ is taken to express that the membership score in X is smaller or equal to Y ($X \leq Y$), and an equivalence $X \leftrightarrow Y$ that the membership scores in X and Y are equal ($X = Y$).

Based on the implication operator, the notions of *sufficiency* and *necessity* are defined, which are the two Boolean dependencies exploited by the (M)INUS theory:

Sufficiency X is sufficient for Y iff $X \rightarrow Y$ (or equivalently: $x + Y$; and colloquially: "if X is present, then Y is present");

Necessity X is necessary for Y iff $Y \rightarrow X$ (or equivalently: $x \rightarrow y$ or $y + X$; and colloquially: "if Y is present, then X is present").

Analogously for more complex expressions:

- X=3*Z=2 is sufficient for Y=4 iff X=3*Z=2 $\rightarrow$ Y=4;
- X=3 + Z=2 is necessary for Y=4 iff Y=4 $\rightarrow$ X=3 + Z=2;
- X=3 + Z=2 is sufficient and necessary for Y=4 iff X=3 + Z=2 $\leftrightarrow$ Y=4.

represent the linear algebraic (arithmetic) operations of multiplication and addition (notational variants of Boolean "*" and "+" are "$\wedge$" and "$\vee$"). For a standard introduction to Boolean algebra see (Bowran 1965).

2.3. (M)INUS causation

Boolean dependencies of sufficiency and necessity amount to mere patterns of co-occurrence of factor values; as such, they carry no causal connotations whatsoever. In fact, most Boolean dependencies do not reflect causal dependencies. To mention just two well-rehearsed examples: the sinking of a properly functioning barometer in combination with high temperatures and blue skies is sufficient for weather changes, but it does not cause the weather; or whenever it rains, the street gets wet, hence, wetness of the street is necessary for rainfall but certainly not causally relevant for it. At the same time, some dependencies of sufficiency and necessity are in fact due to underlying causal dependencies: rainfall is sufficient for wet streets and also a cause thereof, or the presence of oxygen is necessary for fires and also a cause thereof.

That means the crucial problem to be solved by the (M)INUS theory is to filter out those Boolean dependencies that are due to underlying causal dependencies and are, hence, amenable to a causal interpretation. The main reason why most sufficiency and necessity relations do not reflect causation is that they either contain redundancies or are themselves redundant to account for the behavior of the outcome, whereas causal conditions do not feature redundant elements and are themselves indispensable to account for the outcome in at least one context. Accordingly, to filter out the causally interpretable Boolean dependencies, they need to be freed of redundancies. In Mackie's (1974, 62) words, causes are *I*nsufficient but *N*on-redundant parts of *U*nnecessary but *S*ufficient conditions (thus the acronym INUS).

While Mackie's INUS theory only requires that sufficient conditions be freed of redundancies, he himself formulates a problem for that theory, *viz.* the *Manchester Factory Hooters* problem (Mackie 1974, 81-87), which Graßhoff and May (2001) solve by eliminating redundancies also from necessary conditions. Accordingly, modern versions of the INUS theory stipulate that whatever can be removed from sufficient or necessary conditions without affecting their sufficiency and necessity is not a difference-maker and, hence, not a cause. The causally interesting sufficient and necessary conditions are *minimal* in the following sense:

Minimal sufficiency A conjunction Φ of coincidently instantiated factor values (e.g., $X_1 * \ldots * X_n$) is a minimally sufficient condition of Y iff $\Phi \rightarrow Y$ and there does not exist a proper part Φ' of Φ such that $\Phi' \rightarrow Y$, where a proper part Φ' of Φ is the result of eliminating one or more conjuncts from Φ.

Minimal necessity A disjunction Ψ of minimally sufficient conditions (e.g., $\Phi_1 + \ldots + \Phi_n$) is a minimally necessary condition of Y iff $Y \rightarrow \Psi$ and there does not exist a proper part Ψ' of Ψ such that $Y \rightarrow \Psi'$, where a proper part Ψ' of Ψ is the result of eliminating one or more disjuncts from Ψ.

Minimally sufficient and minimally necessary conditions can be combined to so-called *atomic MINUS-formulas* (Beirlaen *et al.* 2018; or, equivalently, *minimal theories*, Graßhoff and May 2001), which, in turn, can be combined to *complex MINUS-formulas*:[5]

Atomic MINUS-formula An atomic MINUS-formula of an outcome Y is an expression $\Psi \leftrightarrow Y$, where Ψ is a minimally necessary disjunction of minimally sufficient conditions of Y, in disjunctive normal form.[6]

[5] We provide suitably simplified definitions that suffice for our purposes here. For complete definitions see (Baumgartner and Falk 2023b).

[6] An expression is in disjunctive normal form iff it is a disjunction of one or more conjunctions of one or more literals (i.e. factor values; Lemmon 1965, 190).

Complex MINUS-formula A complex MINUS-formula of outcomes $Y_1, \ldots, Y_n$ is a conjunction $(\Psi_1 \leftrightarrow Y_1)^* \ldots {}^*(\Psi_n \leftrightarrow Y_n)$ of atomic MINUS-formulas that is itself redundancy-free, meaning it does not logically entail a proper part of itself.[7]

MINUS-formulas connect Boolean dependencies to causal dependencies: only those Boolean dependencies are causally interpretable that appear in MINUS-formulas. To make this concrete, consider the following atomic MINUS-formula:

$$A{*}e \; + \; C{*}d \; \leftrightarrow \; B \tag{1}$$

(1) being a MINUS-formula of B entails that $A{*}e$ and $C{*}d$, but neither A, e, C, nor d alone, are sufficient for B and that $A{*}e \; + \; C{*}d$, but neither $A{*}e$ nor $C{*}d$ alone, are necessary for B. If this holds, it follows that for each factor value in (1) there exists a *difference-making pair*, meaning a pair of configurations such that a change in that factor value alone accounts for a change in the outcome (Baumgartner and Falk 2023b, 9). For example, A being part of the MINUS-formula (1) entails that there are two configurations σ_i and σ_j such that e is given and $C{*}d$ is not given in both σ_i and σ_j, while σ_i features A and B and σ_j features neither A nor B. Only if such a difference-making pair $\langle \sigma_i, \sigma_j \rangle$ exists is A indispensable to account for B. Analogously, (1) being a MINUS-formula entails that there exist difference-making pairs for all other factor values in (1).

To define causation in terms of Boolean difference-making, an additional constraint is needed because not all MINUS-formulas faithfully represent causation. Complete redundancy elimination is relative to the set of analyzed factors $\mathbf{F}$, meaning that factor values contained in MINUS-formulas relative to some $\mathbf{F}$ may fail to be part of a MINUS-formulas relative to supersets of $\mathbf{F}$ (Baumgartner 2013). In other words, by adding further factors to the analysis, factor values that originally appeared to be non-redundant to account for an outcome can turn out to be redundant after all. Hence, a *permanence* constraint needs to be imposed: only factor values that are permanently non-redundant, meaning that cannot be rendered redundant by expanding factor sets, are causally relevant.

These considerations yield the following definition of causation:

Causal Relevance (MINUS) X is causally relevant to Y if, and only if, (I) X is part of a MINUS-formula of Y relative to some factor set $\mathbf{F}$ and (II) X remains part of a MINUS-formula of Y across all expansions of $\mathbf{F}$.

Two features of the (MINUS) definition make it particularly well suited for the analysis of structures affected by conjunctivity and disjunctivity. First, (MINUS) does not require that causes and effects are pairwise dependent. The following is a well-formed MINUS-formula expressing the flow model from the introduction: $S{*}C \; + \; s{*}c \; \leftrightarrow \; F$. As shown in Table 1, ideal data generated from that model feature no pairwise dependencies. Nonetheless, if, say, high skills are permanently non-redundant to account for flow in combination with high challenges, they are causally relevant for flow subject to (MINUS), despite being uncorrelated with flow. Second, MINUS-formulas whose elements satisfy the permanence constraint not only identify causally relevant factor values but also place a Boolean ordering over these

[7] The purpose of the redundancy-freeness constraint imposed on complex MINUS-formulas is to avoid structural and partial structural redundancies; see section 5.5 below.

causes, such that conjunctivity and disjunctivity can be directly read off their syntax. Take the following example:

$$(A{*}b \ + \ a{*}B \ \leftrightarrow \ C) \ * \ (C{*}f \ + \ D \ \leftrightarrow \ E) \tag{2}$$

If (2) complies with (MINUS.I) and (MINUS.II), it entails these causal ascriptions:

1. the factor values listed on the left-hand sides of "$\leftrightarrow$" are causally relevant for the factor values on the right-hand sides;
2. A and b are jointly relevant to C and located on a causal path that differs from the path on which the jointly relevant a and B are located; C and f are jointly relevant to E and located on a path that differs from D's path;
3. there is a causal chain from $A{*}b$ and $a{*}B$ via C to E.

2.4. Inferring MINUS causation from data

Inferring MINUS causation from data faces various challenges. First, as anticipated in section 1, causal structures for which conjunctivity and disjunctivity hold cannot be uncovered by scanning data for dependencies between pairs of factor values and suitably combining dependent pairs. Instead, discovering MINUS causation requires searching for dependencies between complex Boolean functions of exogenous factors and outcomes. But the space of Boolean functions over more than five factors is so vast that it cannot be exhaustively scanned. Hence, algorithmic strategies are needed to purposefully narrow down the search.

Second, condition (MINUS.II) is not comprehensively testable. Once a MINUS-formula of an outcome Y comprising a factor value X has been inferred from data δ, the question arises whether the non-redundancy of X in accounting for Y is an artefact of δ, due, for example, to the uncontrolled variation of confounders, or whether it is genuine and persists when further factors are taken into consideration. But in practice, expanding the set of factors is only feasible within narrow confines. To make up for the impossibility to test (MINUS.II), the analyzed data δ should be collected in such a way that Boolean dependencies in δ are not induced by an uncontrolled variation of latent causes but by the measured factors themselves. If the dependencies in δ are not artefacts of latent causes, they cannot be neutralized by factor set expansions, meaning they are permanent and, hence, causal. It follows that in order for it to be guaranteed that causal inferences drawn from δ are error-free, δ must meet very high quality standards. In particular, the uncontrolled causal background of δ must be *homogeneous* (Baumgartner and Thiem 2020, 286):

Homogeneity The unmeasured causal background of data δ is homogeneous if, and only if, latent causes not connected to the outcome(s) in δ on causal paths via the measured exogenous factors (so-called *off-path* causes) take constant values (i.e. do not vary) in the cases recorded in δ.

However, third, real-life data often do not meet very high quality standards. Rather, they tend to be *fragmented* to the effect that not all empirically possible configurations of analyzed factors are actually observed. Moreover, real-life data typically feature *noise*, that is, configurations incompatible with data-generating causal structures. Noise is induced, for instance,

by measurement error or limited control over latent causes, i.e. confounding. In the presence of noise there may be no strict Boolean sufficiency or necessity relations in the data, meaning that methods of MINUS discovery can only approximate strict MINUS structures by fitting their models more or less closely to the data using suitable parameters and thresholds of model fit (see De Souter 2024). Moreover, noise stemming from the uncontrolled variation of latent causes gives rise to homogeneity violations, which yield that inferences to MINUS causation are not guaranteed to be error-free. In order to nonetheless distill causal information from noisy data, strategies for avoiding over- and underfitting and estimating the error risk are needed (see Parkkinen and Baumgartner 2023).

Fourth, according to the MINUS theory, the inference to *causal irrelevance* is much more demanding than the inference to causal relevance. Establishing that X is a MINUS cause of Y requires demonstrating the existence of at least one context with a constant background in which a difference in X is associated with a difference in Y, whereas establishing that X is *not* a MINUS cause of Y requires demonstrating the *non-existence* of such a context, which is impossible on the basis of the non-exhaustive data samples that are typically analyzed in real-life studies. Correspondingly, the fact that, say, G does not appear in (2) does not imply that G is causally irrelevant to C or E. The non-inclusion of G simply means that the data from which (2) has been derived do not contain evidence for the causal relevance of G. However, future research having access to additional data might reveal the existence of a difference-making context for G and, hence, entail the causal relevance of G to C or E after all.

Finally, on a related note, as a result of the common fragmentation of real-life data δ MINUS-formulas inferred from δ cannot be expected to completely reflect the causal structure generating δ. That is, MINUS-formulas inferred from δ are inevitably going to be *incomplete*. They only detail those causally relevant factor values along with those conjunctive, disjunctive, and sequential groupings for which δ contain difference-making evidence. What difference-making evidence is contained in δ not only depends on the cases recorded in δ but, when δ is noisy, also on the tuning thresholds imposed to approximate strict Boolean dependency structures; relative to some such tuning settings an association between X and Y may pass as a sufficiency or necessity relation whereas relative to another setting it will not. Hence, the inference to MINUS causation is sensitive to the chosen tuning settings, to the effect that choosing different settings is often going to be associated with changes in inferred MINUS-formulas.

A lot of variance (though not all) in inferred MINUS-formulas is unproblematic. Two different MINUS-formulas $\mathbf{m}_i$ and $\mathbf{m}_j$ derived from δ using different tuning settings are in no disagreement if $\mathbf{m}_i$ and $\mathbf{m}_j$ are related in terms of the submodel relation:

Submodel relation A MINUS-formula $\mathbf{m}_i$ is a *submodel* of another MINUS-formula $\mathbf{m}_j$ if, and only if, the causal ascriptions entailed by $\mathbf{m}_i$ are a subset of the causal ascriptions entailed by $\mathbf{m}_j$.

If $\mathbf{m}_i$ is a submodel of $\mathbf{m}_j$, $\mathbf{m}_j$ is a *supermodel* of $\mathbf{m}_i$. All of $\mathbf{m}_i$'s causal ascriptions are contained in its supermodels' ascriptions, and $\mathbf{m}_i$ contains the causal ascriptions of its own submodels. The submodel relation is reflexive: every model is a submodel (and supermodel) of itself; or differently, if $\mathbf{m}_i$ and $\mathbf{m}_j$ are submodels of one another, then $\mathbf{m}_i$ and $\mathbf{m}_j$ are identical. Most importantly, if two MINUS-formulas related by the submodel relation are not identical, they can be interpreted as describing the same causal structure at different levels of detail.

Before we turn to the **cna** package, a terminological note is required. In the literature on configurational comparative methods it has become customary to refer to the models produced by the methods as *solution formulas*. To mirror that convention, the **cna** package refers to atomic MINUS-formulas inferred from data by CNA as *atomic solution formulas*, ***asf***, for short, and to complex MINUS-formulas inferred from data as *complex solution formulas*, ***csf***. For brevity, we will henceforth mainly use the shorthands *asf* and *csf*.

3. The input of CNA

The goal of CNA is to output all *asf* and *csf* within provided bounds of model complexity that fit an input data set relative to provided tuning settings, in particular, fit thresholds. The algorithm performing this task in the **cna** package is implemented in the function `cna()`. Its most important arguments are:

```
cna(x, type, ordering = NULL, strict = FALSE, outcome = TRUE,
    exclude = character(0), con = 1, cov = 1, con.msc = con, notcols = NULL,
    maxstep = c(3, 4, 10), inus.only = TRUE, suff.only = FALSE,
    what = if (suff.only) "m" else "ac", details = FALSE, acyclic.only = FALSE)
```

This section explains most of these inputs and introduces some auxiliary functions. The arguments `inus.only`, `what`, `details`, and `acyclic.only` will be discussed in section 5.

3.1. Data

Data δ processed by CNA have the form of $m \times k$ matrices, where m is the number of units of observation (cases) and k is the number of measured factors. δ can either be of type "crisp-set" (cs), "multi-value" (mv) or "fuzzy-set" (fs). Data that feature cs factors only are cs. If the data contain at least one mv factor, they count as mv. Data featuring at least one fs factor are treated as fs.[8] Examples of each data type are given in Table 3. Raw data collected in a study typically need to be suitably calibrated before they can be fed to `cna()`. We do not address the calibration problem here because it is the same for CNA as for QCA, in which context it has been extensively discussed, for example, by Thiem and Duşa (2013) or Schneider and Wagemann (2012). The R packages **QCApro**, **QCA**, and **SetMethods** provide all tools necessary for data calibration.

Data are given to the `cna()` function via the argument `x`, which must be a data frame or an object of class "configTable" as output by the `configTable()` function (see section 3.1.1 below). The **cna** package contains a number of exemplary data sets from published studies, `d.autonomy`, `d.educate`, `d.irrigate`, `d.jobsecurity`, `d.minaret`, `d.pacts`, `d.pban`, `d.performance`, `d.volatile`, `d.women`, and one simulated data set, `d.highdim`. For details on their contents and sources, see the **cna** reference manual. After having loaded the **cna** package, all of them are directly (i.e. without separate loading) available for processing:

```
R> library(cna)
R> cna(d.educate)
R> cna(d.women)
```

[8] Note, first, that factors calibrated at crisp-set thresholds may appear with unsuitably extreme values if the data as a whole are treated as fs due to some fs factor, and second, that mixing mv and fs factors in one analysis is (currently) not supported.

	A	B	C	D
c_1	0	0	0	0
c_2	0	1	0	0
c_3	1	1	0	0
c_4	0	0	1	0
c_5	1	0	0	1
c_6	1	0	1	1
c_7	0	1	1	1
c_8	1	1	1	1

(a) *cs* data

	A	B	C	D
c_1	1	3	3	1
c_2	2	2	1	2
c_3	2	1	2	2
c_4	2	2	2	2
c_5	3	3	3	2
c_6	2	4	3	2
c_7	1	3	3	3
c_8	1	4	3	3

(b) *mv* data

	A	B	C	D	E
c_1	0.37	0.30	0.16	0.06	0.25
c_2	0.89	0.39	0.64	0.09	0.03
c_3	0.06	0.61	0.92	0.37	0.15
c_4	0.65	0.93	0.92	0.18	0.93
c_5	0.08	0.08	0.12	0.86	0.91
c_6	0.70	0.02	0.85	0.91	0.97
c_7	0.04	0.72	0.76	0.90	0.68
c_8	0.81	0.96	0.89	0.72	0.82

(c) *fs* data

Table 3: Data types processable by CNA.

Prior to version 3.2 of the **cna** package, `cna()` needed be told explicitly what type of data `x` contains using the `type` argument. Now, `type` has the default value `"auto"` inducing automatic detection of the data type. The `type` argument remains in the package for backwards compatibility and in order to allow the user to specify the data type manually: `type = "cs"` stands for *cs* data, `type = "mv"` for *mv* data, and `type = "fs"` for *fs* data.[9]

Configuration tables

To facilitate the reviewing of data, the `configTable()` function assembles cases with identical configurations in a so-called *configuration table*. In previous versions of the **cna** package, these tables were called "truth tables", which however led to confusion with the QCA terminology, where a very different type of object is also referred to by that label. While a QCA truth table indicates for every configuration of all exogenous factors (i.e. for every minterm) whether it is sufficient for the outcome, a CNA configuration table does not express relations of sufficiency but simply amounts to a compact representation of the data that lists all configurations exactly once and adds a column indicating how many instances (cases) of each configuration are contained in the data.

```
configTable(x, type, case.cutoff = 0)
```

The first input `x` is a data frame or matrix. The function then merges multiple rows of `x` featuring the same configuration into one row, such that each row of the resulting table corresponds to one determinate configuration of the factors in `x`. The number of occurrences of a configuration and an enumeration of the cases instantiating it are saved as attributes "n" and "cases", respectively. The argument `type` is the same as in the `cna()` function; it specifies the data type and takes the default value `"auto"` inducing automatic data type detection.

```
R> configTable(d.women)

configTable of type "cs"
            ES QU WS WM LP WNP | n.obs
SE           1  1  1  0  0   1 |     1
FI           1  0  1  0  0   1 |     1
```

[9] The corresponding shortcut functions `cscna()`, `mvcna()`, and `fscna()` also remain available; see `?shortcuts`.

```
IS,NO        1  1  1  1  1   1 |     2
DK           1  0  1  1  1   1 |     1
BE,NL        1  1  0  1  1   1 |     2
ES           1  1  0  1  0   1 |     1
AT           1  1  0  0  1   1 |     1
NZ           0  0  0  1  1   1 |     1
DE           0  1  0  1  1   1 |     1
CH,GR,PT     1  1  0  0  0   0 |     3
AU,FR,GB,IE  0  1  0  1  0   0 |     4
LU           1  0  0  0  1   0 |     1
CA,US        0  0  0  1  0   0 |     2
IT           0  1  0  0  0   0 |     1
Total no.of.cases: 22
```

`configTable()` provides a numeric argument called `case.cutoff`, which allows for setting a minimum frequency cutoff determining that configurations with less instances in the data are not included in the configuration table and the ensuing analysis. For instance, `configTable(x, case.cutoff = 3)` entails that configurations that are instantiated in less than 3 cases are excluded.

Configuration tables produced by `configTable()` can be directly passed to `cna()`. Moreover, as configuration tables generated by `configTable()` are objects that are very particular to the **cna** package, the function `ct2df()` is available to transform configuration tables back into ordinary R data frames.

```
R> pact.ct <- configTable(d.pacts, case.cutoff = 2)
R> ct2df(pact.ct)
```

Data simulations

The **cna** package provides extensive functionalities for data simulations—which, in turn, are essential for inverse search trials that benchmark CNA's output (see section 7). In a nutshell, the functions `allCombs()` and `full.ct()` generate the space of all logically possible configurations over a given set of factors, `selectCases()` selects, from this space, the configurations that are compatible with a data-generating causal structure, which, in turn, can be randomly drawn by `randomAsf()` and `randomCsf()`, `makeFuzzy()` fuzzifies that data, and `some()` randomly selects cases, for instance, to produce data fragmentation.

More specifically, `allCombs(x)` takes an integer vector `x` as input and generates a data frame of all possible value configurations of `length(x)` factors, the first factor having `x[1]` values, the second `x[2]` values etc. The factors are labeled using capital letters in alphabetical order. Analogously, but more flexibly, `full.ct(x)` generates a configuration table with all logically possible value configurations of the factors defined in the input `x`, which can be a configuration table, a data frame, an integer, a list specifying the factors' value ranges, or a character vector featuring all admissible factor values.

```
R> allCombs(c(2, 2, 2)) - 1
R> allCombs(c(3, 4, 5))
```

```
R> full.ct("A + B*c")
R> full.ct(6)
R> full.ct(list(A = 1:2, B = 0:1, C = 1:4))
```

The input of `selectCases(cond, x)` is a character string `cond` specifying a Boolean function, which typically (but not necessarily) expresses a data-generating MINUS structure, as well as, optionally, a data frame or configuration table `x`. If `x` is specified, the function selects the cases that are compatible with `cond` from `x`; if `x` is not specified, it selects from `full.ct(cond)`. It is possible to randomly draw `cond` using `randomAsf(x)` or `randomCsf(x)`, which generate random atomic and complex solution (i.e. MINUS-)formulas, respectively, from a data frame or configuration table `x`.

```
R> dat1 <- allCombs(c(2, 2, 2)) - 1
R> selectCases("A + B <-> C", dat1)
R> selectCases("(h*F + B*C*k + T*r <-> G)*(A*b + H*I*K <-> E)")
R> target <- randomCsf(full.ct(6))
R> selectCases(target)
```

The closely related function `selectCases1(cond, x, con = 1, cov = 1)` additionally allows for providing consistency (`con`) and coverage (`cov`) thresholds (see section 3.2), such that some cases that are incompatible with `cond` are also selected, as long as `cond` still meets `con` and `cov` in the resulting data. Thereby, measurement error or noise can be simulated in a manner that allows for controlling the degree of case incompatibilities.

```
R> dat2 <- full.ct(list(EN = 0:2, TE = 0:4, RU = 1:4))
R> selectCases1("EN=1*TE=3 + EN=2*TE=0 <-> RU=2", dat2, con = .75, cov = .75)
```

`makeFuzzy(x, fuzzvalues = c(0, 0.05, 0.1))` simulates fuzzy-set data by transforming a data frame or configuration table `x` consisting of *cs* factors into an *fs* configuration table. To this end, the function adds values selected at random from the argument `fuzzvalues` to the 0's and subtracts them from the 1's in `x`. `fuzzvalues` is a numeric vector of values from the interval [0,1].

```
R> makeFuzzy(selectCases("Hunger + Heat <-> Run"),
+            fuzzvalues = seq(0, 0.4, 0.05))
```

Finally, `some(x, n = 10, replace = TRUE)` randomly selects `n` cases from a data frame or configuration table `x`, with or without replacement. If `x` features all configurations that are compatible with a data-generating structure and `n < nrow(x)`, the data frame or configuration table issued by `some()` is *fragmented*, meaning it does not contain all empirically possible configurations. If `n > nrow(x)`, data of large sample sizes can be generated featuring multiple instances of the empirically possible configurations.

```
R> dat3 <- allCombs(c(3, 4, 5))
R> dat4 <- selectCases("A=1*B=3 + A=3 <-> C=2", mvct(dat3))
R> some(dat4, n = 10, replace = FALSE)
R> some(dat4, n = 1000, replace = TRUE)
```

3.2. Consistency and coverage

As real-life data tend to feature measurement error or noise induced by variations in latent causes, strictly sufficient or necessary conditions for an outcome often do not exist. In order to nonetheless distill some causal information from such data, methods for MINUS discovery have to suitably fit their models to the data. To this end, Ragin (2006) imported the so-called *consistency* and *coverage* measures into the QCA protocol, both of which are also serviceable for the purposes of CNA. Consistency is formally equivalent to what is known as *precision* or *positive predictive value* in various fields of machine learning. It measures the proportion of cases in the data instantiating a condition (whether sufficient, necessary or a whole model) in combination with the outcome. Coverage, which is equivalent to *recall* and *sensitivity* in machine learning, measures the proportion of cases in the data that instantiate an outcome in combination with the condition (whether sufficient, necessary, or a whole model).[10] As the implication operator underlying the notions of sufficiency and necessity is defined differently in classical and in fuzzy logic, the two measures have different formal definitions for, on one hand, crisp-set and multi-value data (which both have a classical footing), and, on the other, for fuzzy-set data. *Cs-consistency* (con^{cs}) and *cs-coverage* (cov^{cs}) are defined as follows:

$$con^{cs}(X \to Y) = \frac{|X{*}Y|_\delta}{|X|_\delta} \qquad cov^{cs}(X \to Y) = \frac{|X{*}Y|_\delta}{|Y|_\delta}$$

where X and Y are individual factor values or Boolean functions thereof in the analyzed data δ, and $|\ldots|_\delta$ stands for the cardinality of the set of cases in δ instantiating the enclosed expression. *Fs-consistency* (con^{fs}) and *fs-coverage* (cov^{fs}) of $X \to Y$ are defined as follows, where n is the number of cases in the data:

$$con^{fs}(X \to Y) = \frac{\sum_{i=1}^{n} \min(X_i, Y_i)}{\sum_{i=1}^{n} X_i} \qquad cov^{fs}(X \to Y) = \frac{\sum_{i=1}^{n} \min(X_i, Y_i)}{\sum_{i=1}^{n} Y_i}$$

Whenever the values of X and Y are restricted to 1 and 0 in the crisp-set measures, con^{cs} and cov^{cs} are equivalent to con^{fs} and cov^{fs}, but for binary factors with values other than 1 and 0 and for multi-value factors that equivalence does not hold. Nonetheless, we will in the following not explicitly distinguish between the cs and fs measures because our discussion will make it sufficiently clear which of them is at issue.

Consistency and coverage thresholds can be given to the `cna()` function using the arguments `con.msc`, `con`, and `cov` that take values from the interval $[0, 1]$. `con.msc` sets the consistency threshold for minimally sufficient conditions (*msc*), `con` does the same for *asf* and *csf*, while `cov` sets the coverage threshold for *asf* and *csf* (no coverage threshold is imposed on *msc*). As illustrated on pp. 16-17 of the **cna** reference manual, setting different consistency thresholds for *msc* and *asf*/*csf* can enhance the informativeness of `cna()`'s output in certain cases but is non-standard. The standard setting is `con = con.msc`.

The default numeric value for all thresholds is 1, i.e. perfect consistency and coverage. Contrary to QCA, which often returns solutions that do not comply with the chosen consistency threshold and which does not impose a coverage threshold at all, CNA uses consistency and coverage as authoritative model building criteria such that, if they are not met, CNA abstains from issuing solutions. That means, if the default thresholds are used, `cna()` will only output perfectly consistent *msc*, *asf*, and *csf* and only perfectly covering *asf* and *csf*.

[10] De Souter (2024) has recently shown that CNA can also benefit from other evaluation measures from machine learning, such as *specificity* and *negative predictive value*.

If the data are noisy, the default thresholds will typically not yield any solution formulas. In such cases, `con` and `cov` may be suitably lowered. By lowering `con` to, say, 0.75 in a *cs* analysis, `cna()` is given permission to treat X as sufficient for Y, even though Y is absent from 25% of the cases with X. Or by lowering `cov` to 0.75 in an *fs* analysis, `cna()` is allowed to treat X as necessary for Y, even though the sum of the membership scores in Y over all cases in the data exceeds the sum of the membership scores in $\min(X, Y)$ by 25%.

Determining the optimal values to which `con` and `cov` should be lowered in a specific discovery context is a delicate task. On the one hand, CNA faces a severe overfitting risk when the data contain configurations incompatible with the data-generating structure, meaning that `con` and `cov` must not be set too high (i.e. too close to 1). On the other hand, the lower `con` and `cov` are set, the less complex and informative CNA's output will be, that is, the more CNA's purpose of uncovering causal complexity will be undermined. To find a suitable balance between over- and underfitting, Parkkinen and Baumgartner (2023) systematically re-analyze the data at all `con` and `cov` settings in the interval $[0.7, 1]$, collect all solutions resulting from such a re-analysis series in a set $\mathbf{M}$, and select the solution formula(s) with the most sub- and supermodels in $\mathbf{M}$. These are the solutions with the highest *overlap in causal ascriptions* with the other solutions in $\mathbf{M}$. They are the most *robust* solutions inferable from the data. This approach to robustness scoring is implemented in the function `frscored_cna(x, fit.range, granularity)` of the R package **frscore** (Parkkinen and Baumgartner 2024).[11] The function accepts all arguments of `cna()`, except for `con`, `con.msc`, and `cov`. It analyzes the data `x` at all consistency and coverage threshold combinations in the interval `fit.range` with increments specified by `granularity`, and it scores the resulting models based on their robustness.

```
R> library(frscore)
R> frscored_cna(d.autonomy, ordering = "AU", fit.range = c(1, 0.8),
+     granularity = 0.1)
```

If the analyst does not want to conduct a whole robustness analysis, reasonable non-perfect consistency and coverage settings are `con` = `cov` = 0.8 or 0.75. To illustrate, `cna()` does not build solutions for the `d.jobsecurity` data at the following `con` and `cov` thresholds (the argument `outcome` is explained in section 3.3 below):

```
R> cna(d.jobsecurity, outcome = "JSR", con = 1, cov = 1)
R> cna(d.jobsecurity, outcome = "JSR", con = .9, cov = .9)
```

But if `con` and `cov` are set to 0.75, 20 solutions are returned:

```
R> cna(d.jobsecurity, outcome = "JSR", con = .75, cov = .75)
```

In the presence of noise, it is generally advisable to vary the `con` and `cov` settings to some degree in order to get a sense for how sensitive the model space reacts to changes in tuning settings and for the overlap in causal ascriptions between different solutions. Less complex solutions with more overlap with the rest of the returned solutions are generally preferable over more complex solutions with less overlap. If the consistency and coverage scores of

[11] In addition, the R package **cnaOpt** (Ambühl and Baumgartner 2022) provides functions for finding the maximal consistency and coverage scores obtainable from a given data set and for identifying models reaching those scores. For a discussion of possible applications of maximal scores see (Baumgartner and Ambühl 2021).

resulting solutions can be increased by raising the `con` and `cov` settings without, at the same time, disproportionately increasing the solutions' complexity, solutions with higher fit are preferable over solutions with lower fit. But if an increase in fit comes with a substantive increase in model complexity, less complex models with lower fit are to be preferred (to avoid overfitting).

3.3. Outcome, ordering, and exclude

In principle, the `cna()` function does not need to be told which factors in the data `x` have values that are endogenous (i.e. outcomes) and which ones have exogenous values (i.e. causes). It attempts to infer that from `x`. However, in ordinary research contexts, analysts do not start from scratch. They often possess some prior theoretical or causal knowledge about an investigated causal structure that allows them to, for example, identify potential outcomes or to exclude certain causal relationships. The `cna()` function provides three arguments—`outcome`, `ordering`, and `exclude`—through which prior causal information can be supplied to efficiently constrain the search space. This section introduces these arguments and their interplay.

Prior knowledge about which factors have values that can figure as outcomes can be given to `cna()` via the argument `outcome`, which takes as input a character vector specifying one or several factor values that are to be considered as potential outcome(s). For *cs* and *fs* data, factor values are expressed by upper and lower cases (e.g. `outcome = c("A", "b")`), for *mv* data, they are expressed by the "Factor=value" notation (e.g. `outcome = c("A=1","B=3")`). The default is `outcome = TRUE`, which means that all factor values in `x` are potential outcomes. For example, the following function call determines that of all 9 factors in `d.volatile`, only *VO*2, i.e. VO2 taking the value 1, is a potential outcome, meaning that `cna()` does not attempt to model values of any other factors as outcomes:

```
R> cna(d.volatile, outcome = "VO2")
```

When the data `x` contain multiple potential outcomes, it may moreover be known that these outcomes are causally ordered in a certain way, to the effect that some of them are causally upstream of the others. Such information can be given to CNA via a *causal ordering*, which is a relation A $\prec$ C (defined on the factors in `x`) entailing that values of C cannot cause values of A (e.g. because instances of A occur temporally before instances of C). That is, an ordering excludes certain causal dependencies but does not stipulate any. The corresponding argument, `ordering`, takes as value a character string. For example, `ordering = "A, B < C"` determines that factor C is causally located after A and B, meaning that values of C are not potential causes of values of A and B. The latter are located on the same level of the ordering, for A and B are unrelated by $\prec$, whereas C is located on a level that is downstream of the A, B-level. If an ordering is provided, `cna()` only searches for MINUS-formulas in accordance with the ordering. An ordering does not need to explicitly mention all factors in `x`. If only a subset of the factors are assigned to `ordering`, the non-included factors are entailed to be upstream of the included ones. Hence, `ordering = "C"` means that C is located downstream of all other factors in `x`.

To further specify the causal ordering, the logical argument `strict` is available. It determines whether the elements of one level in an ordering can be causally related or not. For example, if `ordering = "A, B < C"` and `strict = TRUE`, then values of A and B are excluded to be

causally related and `cna()` skips corresponding tests. By contrast, if `ordering = "A, B < C"` and `strict = FALSE`, then `cna()` also searches for dependencies among values of A and B.

Let us illustrate this with the data set `d.autonomy`. Relative to the following function call, which stipulates that values of AU cannot cause values of EM, SP, and CO and that the latter factors are not mutually causally related, `cna()` infers that SP is causally relevant to AU:[12]

```
R> dat.aut.1 <- d.autonomy[15:30, c("AU","EM","SP","CO")]
R> ana.aut.1 <- cna(dat.aut.1, ordering = "EM, SP, CO < AU", strict = TRUE,
+     con = .9, cov = .9)
R> printCols <- c("condition", "consistency", "coverage")
R> csf(ana.aut.1)[printCols]

  condition consistency coverage
1 SP <-> AU       0.935    0.915
```

If we set `strict` to `FALSE` and, thereby, allow for causal dependencies among values of EM, SP, and CO, it turns out that SP not only causes AU, but, on another causal path, also makes a difference to EM:

```
R> ana.aut.2 <- cna(dat.aut.1, ordering = "EM, SP, CO < AU", strict = FALSE,
+    con = .9, cov = .9)
R> csf(ana.aut.2)[printCols]

  condition                     consistency coverage
1 (SP <-> AU)*(SP + CO <-> EM)       0.912    0.915
```

The arguments `ordering` and `outcome` interact closely. It is often not necessary to specify both of them. For example, `ordering = "C", strict = TRUE` is equivalent to `outcome = "C"`. Still, it is important to note that the characters assigned to `ordering` are interpreted as *factors*, whereas the characters assigned to `outcome` are interpreted as *factor values*. This difference may require the specification of both `ordering` and `outcome`, in particular, when only specific values of the factors in the ordering are potential outcomes. To illustrate, compare the following two function calls:

```
R> cna(d.pban, ordering = "T, PB", con = .75, cov = .75)
R> cna(d.pban, outcome = c("T=2", "PB=1"), ordering = "T, PB",
+    con = .75, cov = .75)
```

The first call entails that any values of the factors T and PB, in that order, are located at the downstream end of the causal structure generating the data `d.pban`. It returns various solutions for PB=1 as well as for both T=0 and T=2. The second call, by contrast, narrows the search down to T=2 as only potential outcome value of factor T, such that no solutions for T=0 are produced.

A causal ordering excludes *all* values of a factor as potential causes of an outcome. However, a user might only wish to exclude *some* values as potential causes. This selective exclusion can

[12] The function `csf()` used in the following code builds the *csf* from a `cna()` solution object; see section 5.1.

be specified in the `exclude` argument, available as of version 3.6.1 of **cna**. It is assigned a vector of character strings, where factor values to be excluded are listed to the left of the `"->"` sign, and the corresponding outcomes are listed to the right. For example, `exclude = "A=1,C=3 -> B=1"` excludes that the value 1 of factor A and the value 3 of factor C are considered as causes of the value 1 of factor B. It is also possible to exclude factor values as potential causes of multiple outcomes; for instance, `exclude = c("A,c -> B", "b,H -> D")`. In the context of *cs* and *fs* data, upper case letters are interpreted as 1, while lower case letters are interpreted as 0. If factor names have multiple letters, any upper case letter is interpreted as 1, and the absence of upper case letters as 0. For *mv* data, the "Factor=value" notation is required. The `exclude` argument can be used either independently or in conjunction with `outcome` and `ordering`. But if the assignments to `outcome` and `ordering` contradict those to `exclude`, the assignments to `exclude` will be disregarded.

For example, the following call narrows down the search beyond what is specified in the `outcome` and `ordering` arguments by excluding the causal relevance of C=2 for T=2 and of T=1 and V=0 for PB=1:

```
R> cna(d.pban, outcome = c("T=2", "PB=1"), ordering = "T, PB",
+    con = .75, cov = .75, exclude = c("C=2 -> T=2", "T=1,V=0 -> PB=1"))
```

Selective exclusion of certain causal relationship might also be the only type of prior knowledge available to an analyst. In that case, `exclude` can be used as sole search space constraint.

```
R> cna(d.jobsecurity, con = .85, cov = .85, exclude = c("s,c -> JSR",
+    "jsr, L -> R"))
```

In general, `cna()` should be given all the causal information about the interplay of the factors in the data that is available prior to the analysis. There often exist many MINUS-formulas that fit the data equally well. The more prior information `cna()` has at its disposal, the more specific the output will be, on average.

3.4. Maxstep

As will be exhibited in more detail in section 4, `cna()` builds atomic solution formulas (*asf*), *viz.* minimally necessary disjunctions of minimally sufficient conditions (*msc*), from the bottom up by gradually permuting and testing conjunctions and disjunctions of increasing complexity for sufficiency and necessity. The combinatorial search space that this algorithm has to scan depends on a variety of different aspects, for instance, on the number of factors in `x`, on the number of values these factors can take, on the number and length of the recovered *msc*, etc. As the search space may be too large to be exhaustively scanned in reasonable time, the argument `maxstep` allows for setting an upper bound for the complexity of the generated *asf*. `maxstep` takes a vector of three integers $c(i, j, k)$ as input, entailing that the generated *asf* have maximally j disjuncts with maximally i conjuncts each and a total of maximally k factor values. The default is `maxstep = c(3,4,10)`. The user can set it to any complexity level if computational time and resources are not an issue.

The `maxstep` argument is particularly relevant for the analysis of high dimensional data and data featuring severe model ambiguities. As an example of the first kind, consider the data `d.highdim` comprising 50 crisp-set factors, V1 to V50, and 1191 cases, which were simulated

from a presupposed data-generating structure with the outcomes $V13$ and $V11$ (see the **cna** reference manual for details). These data feature 20% noise and massive fragmentation. At the default `maxstep`, the following analysis, which finds the complete data-generating structure, takes between 15 and 20 seconds to complete; lowering `maxstep` to `c(2,3,10)` reduces that time to less than one second, at the expense of only finding half of the data-generating structure:

```
R> cna(d.highdim,  outcome = c("V13", "V11"), con = .8, cov = .8)
R> cna(d.highdim,  outcome = c("V13", "V11"), con = .8, cov = .8,
+    maxstep = c(2,3,10))
```

A telling example of extensive model ambiguities is the data set `d.volatile`. When only constrained by an ordering, `cna()` quickly recovers 416 complex solution formulas (*csf*) at the default `maxstep`. But those are by far not all *csf* that fit `d.volatile` equally well. When `maxstep` is increased only slightly to `c(4,4,10)`, the number of *csf* jumps to 2860:[13]

```
R> cna(d.volatile, ordering = "VO2", maxstep = c(3,4,10))
R> vol1 <- cna(d.volatile, ordering = "VO2", maxstep = c(4,4,10))
R> csf(vol1, n.init = 3000)
```

If `maxstep` is further increased, the number of solutions explodes and the analysis soon fails to terminate in reasonable time. When a complete analysis cannot be completed, `cna()` can be told to only search for *msc* by setting the argument `suff.only` to its non-default value `TRUE`. As the search for *msc* is the part of a CNA analysis that is least computationally demanding, it will typically terminate quickly and, thus, shed some light on the dependencies among the factors in `x` even when the construction of all models is infeasible.

```
R> cna(d.volatile, ordering = "VO2", maxstep = c(8,10,40), suff.only = TRUE)
```

If `suff.only` is set to `TRUE`, CNA can process data of higher dimensionality than at the argument's default value. Yakovchenko *et al.* (2020), for example, run `cna()` on data comprising 73 exogenous factors with `suff.only = TRUE`. Based on the resulting *msc*, they then select a proper subset of those factors for further processing.

While the `maxstep` argument is valuable for controlling the search space in case of high-dimensional and ambiguous data, it also comes with a pitfall: it may happen that `cna()` fails to find a model because of a `maxstep` that is too low. An example is `d.jobsecurity`. At the default `maxstep`, `cna()` does not build a solution, but if `maxstep` is increased, two solutions are found.

```
R> ana.jsc.1 <- cna(d.jobsecurity, ordering = "JSR", con = .9, cov = .85)
R> csf(ana.jsc.1)[printCols]

[1] condition   consistency coverage
<0 rows> (or 0-length row.names)
```

[13] In the standard print method of `cna()`, the `n.init` parameter in `csf()` is set to 1000; to get all *csf*, this parameter needs to be increased. See section 5.1 for details.

```
R> ana.jsc.2 <- cna(d.jobsecurity, ordering = "JSR", con = .9, cov = .85,
+    maxstep = c(3,5,12))
R> csf(ana.jsc.2)[printCols]
```

```
  condition                                  consistency coverage
1 S*V + C*l + L*P + R*V + S*c*R <-> JSR          0.906    0.859
2 C*l + R*V + P*v + S*c*R + S*C*P <-> JSR        0.912    0.853
```

In sum, there are two possible reasons for why `cna()` fails to build a solution: (i) the chosen `maxstep` is too low; (ii) the chosen `con` and/or `cov` values are too high, meaning the processed data `x` are too noisy. Accordingly, in case of a null result, two paths should be explored (in that order): (i) gradually increase `maxstep`; (ii) lower `con` and `cov`, as described in section 3.2 above.

3.5. Negated outcomes

In classical logic, the law of Contraposition ensures that an expression of type $\Psi \leftrightarrow Y$ is equivalent to the expression that results from negating both sides of the double arrow: $\neg\Psi \leftrightarrow \neg Y$. Applied to the context of configurational causal modeling that entails that an *asf* for Y can be transformed into an *asf* for the negation of Y, *viz.* y, based on logical principles alone, i.e. without a separate data analysis. However, that transformability only holds for *asf* with perfect consistency and coverage (`con = cov = 1`) that are inferred from exhaustive (non-fragmented) data (see section 5.3 for details on exhaustiveness). If an *asf* of an outcome Y does not reach perfect consistency or coverage or is inferred from fragmented data, identifying the causes of y requires a separate application of `cna()`.

There are two ways to search for the causes of negated outcomes. The first is by simply specifying the factor values of interest in the `outcome` argument. While `outcome = c("A", "B")` yields MINUS-formulas for the positive outcomes A and B, `outcome = c("a", "b")` induces `cna()` to search for models of the corresponding negated outcomes. Alternatively, the argument `notcols` allows for negating the values of factors in *cs* and *fs* data (in case of *mv* data, `cna()` automatically searches for models of all possible values of endogenous factors, thereby rendering `notcols` redundant). If `notcols = "all"`, all factors are negated, i.e. their values i are replaced by $1 - i$. If `notcols` is given a character vector of factors in the data, only the values of the factors in that vector are negated. For example, `notcols = c("A", "B")` determines that only A and B are negated.

When processing *cs* or *fs* data, CNA should first be used to model the positive outcomes. If resulting *asf* and *csf* do not reach perfect consistency, coverage, and exhaustiveness scores (and the causes of the negated outcomes are of interest), a second CNA should be run negating the values of all factors that have been modeled as outcomes in the first CNA. To illustrate, we revisit our analysis of `d.autonomy` from section 3.3, which identified AU and EM as outcomes. The following two calls conduct analyses of the corresponding negated outcomes that produce the same solutions.

```
R> ana.aut.3 <- cna(dat.aut.1, outcome = c("au", "em"), con = .88, cov = .82)
R> csf(ana.aut.3)[printCols]
```

```
  condition                  consistency coverage
1 (sp <-> au)*(sp*co <-> em)       0.882    0.821
```

```
R> ana.aut.4 <- cna(dat.aut.1, ordering = "AU", con = .88, cov = .82,
+     notcols = c("AU", "EM"))
R> csf(ana.aut.4)[printCols]

  condition                  consistency coverage
1 (sp <-> au)*(sp*co <-> em)       0.882    0.821
```

4. The CNA algorithm

This section explains the working of the algorithm implemented in the `cna()` function. We first provide an informal summary and then a detailed outline in four stages. The aim of `cna()` is to find all *msc*, *asf*, and *csf* that meet `con.msc`, `con` and `cov` in the input data `x` in accordance with `outcome`, `ordering`, `exclude`, and `maxstep`. The algorithm starts with single factor values and tests whether they meet `con.msc`; if that is not the case, it proceeds to test conjunctions of two factor values, then to conjunctions of three, and so on. Whenever a conjunction meets `con.msc` (and no proper part of it has previously been identified to meet `con.msc`), it is automatically a minimally sufficient condition *msc*, and supersets of it do not need to be tested any more. Then, it tests whether single *msc* meet `con` and `cov`; if not, it proceeds to disjunctions of two, then to disjunctions of three, and so on. Whenever a disjunction meets `con` and `cov` (and no proper part of it has previously been identified to meet `con` and `cov`), it is automatically a minimally necessary disjunction of *msc*, and supersets of it do not need to be tested any more. All and only those disjunctions of *msc* that meet both `con` and `cov` are then issued as *asf*. Finally, recovered *asf* are conjunctively concatenated to *csf* while eliminating structural redundancies and deleting tautologous and contradictory solutions as well as solutions with partial structural redundancies and constant factors.

The `cna()` algorithm can be more specifically broken down into four stages.

Stage 1 On the basis of `outcome`, `ordering`, and `exclude`, `cna()` first builds a set of potential outcomes $\mathbf{O} = \{\mathrm{O}_h{=}\omega_f, \ldots, \mathrm{O}_m{=}\omega_g\}$ from the set of factors $\mathbf{F} = \{\mathrm{O}_1, \ldots, \mathrm{O}_n\}$ in `x`,[14] where $1 \leq h \leq m \leq n$, and second assigns a set of potential cause factors $\mathbf{C}_{\mathrm{O}_i}$ from $\mathbf{F} \setminus \{\mathrm{O}_i\}$ to every element $\mathrm{O}_i{=}\omega_k$ of $\mathbf{O}$. If no `outcome` and `ordering` are provided, all value assignments to all elements of $\mathbf{F}$ are treated as possible outcomes in case of *mv* data, whereas in case of *cs* and *fs* data $\mathbf{O}$ is set to $\{\mathrm{O}_1{=}1, \ldots, \mathrm{O}_n{=}1\}$. If both `ordering` and `exclude` are empty, all values of all factors in $\mathbf{F} \setminus \{\mathrm{O}_i\}$ are treated as possible causes of $\mathrm{O}_i{=}\omega_k$, for every $\mathrm{O}_i{=}\omega_k \in \mathbf{O}$.

Stage 2 `cna()` attempts to build a set $\mathbf{msc}_{\mathbf{O}_i=\omega_k}$ of minimally sufficient conditions that meet `con.msc` for each $\mathrm{O}_i{=}\omega_k \in \mathbf{O}$. To this end, it first checks for each value assignment $\mathrm{X}_h{=}\chi_j$ of each element of $\mathbf{C}_{\mathrm{O}_i}$, such that $\mathrm{X}_h{=}\chi_j$ has a membership score above 0.5 in at least one case in `x`, whether the consistency of $\mathrm{X}_h{=}\chi_j \rightarrow \mathrm{O}_i{=}\omega_k$ meets `con.msc`, i.e. whether $con(\mathrm{X}_h{=}\chi_j \rightarrow \mathrm{O}_i{=}\omega_k) \geq$ `con.msc`. If, and only if, that is the case, $\mathrm{X}_h{=}\chi_j$ is put into the set $\mathbf{msc}_{\mathbf{O}_i=\omega_k}$. Next, `cna()` checks for each conjunction of two factor values $\mathrm{X}_m{=}\chi_j * \mathrm{X}_n{=}\chi_l$ from $\mathbf{C}_{\mathrm{O}_i}$, such that $\mathrm{X}_m{=}\chi_j * \mathrm{X}_n{=}\chi_l$ has a membership score above 0.5

[14] Note that if `x` is a data frame, `cna()` first transforms `x` into a configuration table by means of `configTable(x)`, thereby passing the argument `type` (and the two additional arguments `rm.dup.factors` and `rm.const.factors`) to the `configTable()` function.

in at least one case in `x` and no part of $X_m{=}\chi_j * X_n{=}\chi_l$ is already contained in $\mathbf{msc}_{\mathbf{O}_i=\omega_k}$, whether $con(X_m{=}\chi_j * X_n{=}\chi_l \rightarrow O_i{=}\omega_k) \geq$ `con.msc`. If, and only if, that is the case, $X_m{=}\chi_j * X_n{=}\chi_l$ is put into the set $\mathbf{msc}_{\mathbf{O}_i=\omega_k}$. Next, conjunctions of three factor values with no parts already contained in $\mathbf{msc}_{\mathbf{O}_i=\omega_k}$ are tested, then conjunctions of four factor values, etc., until either all logically possible conjunctions of the elements of $\mathbf{C}_{O_i}$ have been tested or `maxstep` is reached. Every non-empty $\mathbf{msc}_{\mathbf{O}_i=\omega_k}$ is passed on to the third stage.

Stage 3 `cna()` attempts to build a set $\mathbf{asf}_{\mathbf{O}_i=\omega_k}$ of atomic solution formulas for every $O_i{=}\omega_k \in \mathbf{O}$, which has a non-empty $\mathbf{msc}_{\mathbf{O}_i=\omega_k}$, by disjunctively concatenating the elements of $\mathbf{msc}_{\mathbf{O}_i=\omega_k}$ to minimally necessary conditions of $O_i{=}\omega_k$ that meet `con` and `cov`. To this end, it first checks for each single condition $\Phi_h \in \mathbf{msc}_{O_i=\omega_k}$ whether $con(\Phi_h \rightarrow O_i{=}\omega_k) \geq$ `con` and $cov(\Phi_h \rightarrow O_i{=}\omega_k) \geq$ `cov`. If, and only if, that is the case, Φ_h is put into the set $\mathbf{asf}_{\mathbf{O}_i=\omega_k}$. Next, `cna()` checks for each disjunction of two conditions $\Phi_m + \Phi_n$ from $\mathbf{msc}_{O_i=\omega_k}$, such that no part of $\Phi_m + \Phi_n$ is already contained in $\mathbf{asf}_{\mathbf{O}_i=\omega_k}$, whether $con(\Phi_m + \Phi_n \rightarrow O_i{=}\omega_k) \geq$ `con` and $cov(\Phi_m + \Phi_n \rightarrow O_i{=}\omega_k) \geq$ `cov`. If, and only if, that is the case, $\Phi_m + \Phi_n$ is put into the set $\mathbf{asf}_{\mathbf{O}_i=\omega_k}$. Next, disjunctions of three conditions from $\mathbf{msc}_{O_i=\omega_k}$ with no parts already contained in $\mathbf{asf}_{\mathbf{O}_i=\omega_k}$ are tested, then disjunctions of four conditions, etc., until either all logically possible disjunctions of the elements of $\mathbf{msc}_{O_i=\omega_k}$ have been tested or `maxstep` is reached. Every non-empty $\mathbf{asf}_{\mathbf{O}_i=\omega_k}$ is passed on to the fourth stage.

Stage 4 `cna()` calls the function `csf()`, which builds a set $\mathbf{csf}_{\mathbf{O}}$ of complex solution formulas. This is done in a stepwise manner as follows. First, all logically possible conjunctions of exactly one element from every non-empty $\mathbf{asf}_{O_i=\omega_k}$ are constructed. Second, if `inus.only = TRUE` (see section 5.2 below), the conjunctions resulting from step 1 are freed of structural redundancies (cf. Baumgartner and Falk 2023b), and tautologous and contradictory solutions as well as solutions with partial structural redundancies and constant factors are eliminated. Third, if `acyclic.only = TRUE`, solutions with cyclic substructures are eliminated. Fourth, for those solutions that were modified in the previous steps, consistency and coverage are re-calculated and solutions that no longer reach `con` or `cov` are eliminated. The remaining solutions are returned as $\mathbf{csf}_{\mathbf{O}}$. If there is only one non-empty set $\mathbf{asf}_{O_i=\omega_k}$, $\mathbf{csf}_{\mathbf{O}}$ is identical to $\mathbf{asf}_{O_i=\omega_k}$.

To illustrate, the following code chunk, first, simulates the data in Table 3c, p. 12, and second, runs `cna()` (and `csf()`) on that data at `con = .8` and `cov = .8`, with default `maxstep`, and without specification of `outcome`, `ordering`, and `exclude`.

```
R> dat5 <- allCombs(c(2, 2, 2, 2, 2)) -1
R> dat6 <- selectCases("(A + B <-> C)*(A*B + D <-> E)", dat5)
R> set.seed(3)
R> tab3c <- makeFuzzy(dat6, fuzzvalues = seq(0, 0.4, 0.01))
R> cna(tab3c, con = .8, cov = .8, what = "mac")
```

Table 3c contains data of type *fs*, meaning that the values in the data matrix are interpreted as membership scores in fuzzy sets. As is customary for this data type, we use uppercase italics for membership in a set and lowercase italics for non-membership. In the absence of any prior causal knowledge, the set of potential outcomes is determined to be

$\mathbf{O} = \{A, B, C, D, E\}$ in stage 1, that is, the presence of each factor in Table 3c is treated as a potential outcome. Moreover, all other factors are potential cause factors of every element of $\mathbf{O}$, hence, $\mathbf{C}_A = \{\mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}\}$, $\mathbf{C}_B = \{\mathrm{A}, \mathrm{C}, \mathrm{D}, \mathrm{E}\}$, $\mathbf{C}_C = \{\mathrm{A}, \mathrm{B}, \mathrm{D}, \mathrm{E}\}$, $\mathbf{C}_D = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{E}\}$, and $\mathbf{C}_E = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$.

In stage 2, `cna()` succeeds in building non-empty sets of minimally sufficient conditions in compliance with `con.msc` and `con` for all elements of $\mathbf{O}$: $\mathbf{msc}_A = \{B{*}d{*}E\}$, $\mathbf{msc}_B = \{C{*}d,\ d{*}E,\ a{*}C{*}D,\ a{*}C{*}E,\ a{*}C{*}e\}$, $\mathbf{msc}_C = \{A,\ B,\ d{*}E\}$, $\mathbf{msc}_D = \{b{*}E,\ a{*}E,\ c{*}E\}$, $\mathbf{msc}_E = \{D,\ A{*}B,\ A{*}C\}$. But only the elements of $\mathbf{msc}_C$ and $\mathbf{msc}_E$ can be disjunctively combined to atomic solution formulas that meet `cov` in stage 3: $\mathbf{asf}_C = \{A + B \leftrightarrow C\}$ and $\mathbf{asf}_E = \{D + A{*}B \leftrightarrow E,\ D + A{*}C \leftrightarrow E\}$. For the other three factors in $\mathbf{O}$, the coverage threshold of 0.8 cannot be satisfied. `cna()` therefore abstains from issuing *asf* for A, B and D.

Finally, in stage 4 one redundancy-free *csf* is built from the inventory of *asf* in $\mathbf{asf}_C$ and $\mathbf{asf}_E$, which constitutes `cna()`'s final output for Table 3c:

$$(A\ +\ B \leftrightarrow C) * (D\ +\ A{*}B \leftrightarrow E) \qquad con = 0.836;\ cov = 0.897 \tag{3}$$

5. The output of CNA

5.1. Customizing the output

The default output of `cna()` first lists the provided ordering, second, the pre-identified outcomes, third, the *asf* that were recovered in accordance with the ordering, and fourth, the *csf*. *Asf* and *csf* are ordered by complexity and the product of consistency and coverage. For *asf* and *csf*, four attributes are standardly computed: `consistency`, `coverage`, `complexity`, and `inus`. `consistency` and `coverage` correspond to a solution's consistency and coverage scores, which measures have been explained in section 3.2 above. The `complexity` score amounts to the number of factor values on the left-hand sides of "→" or "↔" in *asf* and *csf*; and the `inus` attribute indicates whether a solution has the form of a well-formed MINUS structure (for more on the `inus` attribute cf. section 5.2 below).

`cna()` can compute additional solution attributes, all of which will be explained below: `exhaustiveness`, and `faithfulness` for both *asf* and *csf*, as well as `coherence`, `redundant`, and `cyclic` for *csf*. These attributes are accessible via the `details` argument, which can be given the values `TRUE`/`FALSE`, for computing all/none of the additional attributes, or a character vector specifying the attributes to be computed: for example, `details = c("faithfulness", "exhaustiveness")`—the strings can also be abbreviated, e.g. `"f"` for `"faithfulness"`, `"e"` for `"exhaustiveness"`, etc.

```
R> cna(d.educate, details = TRUE)
R> cna(d.educate, details = c("co", "cy"))
```

The output of `cna()` can be further customized through the argument `what` that controls which solution items to print. It can be given a character string specifying the requested solution items: `"t"` stands for the configuration table, `"m"` for minimally sufficient conditions (*msc*), `"a"` for *asf*, `"c"` for *csf*, and `"all"` for all solution items.

```
R> cna(d.educate, what = "tm")
R> cna(d.educate, what = "mac")
R> cna(d.educate, what = "all")
```

As shown in section 3.4, it can happen that many *asf* and *csf* fit the data equally well. The standard output of `cna()` only features 5 solution items of each type. To recover all *msc* and *asf* the functions `msc(x)` and `asf(x)` are available, where `x` is a solution object of `cna()`.

```
R> vol2 <- cna(d.volatile, ordering = "VO2", con = .9, cov = .9)
R> msc(vol2)
R> asf(vol2)
R> print(asf(vol2), Inf)
```

While `msc()` and `asf()` simply access the complete sets of *msc* and *asf* stored in `x`, the *csf* are not stored in `x`. The construction of *csf* in the fourth stage of the CNA algorithm is not conducted by the `cna()` function itself, rather, it is outsourced to the function `csf()` with these main arguments:

```
csf(x, n.init = 1000, inus.only = x$inus.only,
   minimalizeCsf = inus.only, cyclic.only = x$acyclic.only,
   cycle.type = x$cycle.type, verbose = FALSE)
```

The arguments `inus.only`, `minimalizeCsf`, `cyclic.only`, and `cycle.type` will be further discussed in the following sections, `n.init` and `verbose` are explained in the remainder of this one. It can happen that the set $\mathbf{asf}_{\mathbf{O}_i=\omega_k}$ contains too many *asf* to construct all *csf* in reasonable time. The argument `n.init` therefore allows for controlling how many conjunctions of *asf* are initially built in the first step of *csf* construction (see stage 4 of the CNA algorithm); it defaults to 1000. Increasing or lowering that default results in more or less *csf* being built and in longer or shorter computing times, respectively.

```
R> csf(vol2, n.init = 2000)
R> csf(vol2, n.init = 100)
```

Setting the argument `verbose` to its non-default value `TRUE` prints some information about the *csf* construction process to the console, e.g. how many structural redundancies or cyclic substructures have been eliminated along the way.

```
R> csf(vol2, verbose = TRUE)
```

5.2. INUS vs. non-INUS solutions

The (M)INUS-theory of causation (cf. section 2.3) has been developed for strictly Boolean discovery contexts, meaning for deterministic (i.e. noise-free) data that feature perfectly sufficient and necessary conditions. In such contexts, some Boolean expressions can be identified as non-minimal (i.e. as featuring redundant elements) on mere *logical grounds*, that is, independently of data. For instance, in an expression as

$$A + a*B \leftrightarrow C \tag{4}$$

a in the second disjunct is redundant, for (4) is logically equivalent to $A + B \leftrightarrow C$. These two formulas state exactly the same. Under no conceivable circumstances could a as contained in (4) ever make a difference to C. To see this, note that a necessary condition for $a{*}B$ to be a complex cause of C is that there exists a context $\mathcal{F}$ such that C is only instantiated when *both* a and B are given. That means that, in $\mathcal{F}$, C is not instantiated if B is given but a is not, which, in turn, means that C is not instantiated if B is given and A (*viz.* not-a) is given. But such an $\mathcal{F}$ cannot possibly exist, for A itself is sufficient for C according to (4). It follows that in every context where B is instantiated, a change from A to a is not associated with a change in C (which takes the value 1 throughout the change in the factor A), meaning that a cannot possibly make a difference to C and, hence, cannot be a cause of C. That is, (4) can be identified as non-minimal independently of all data. (4) is not a well-formed causal model. It is not a MINUS-formula, or not an *INUS solution*.

Correspondingly, a solution as (4) or any other non-INUS expression will never be inferred from strictly deterministic data. By contrast, it is possible that a necessary disjunction of sufficient conditions that does *not* have INUS form is redundancy-free (or minimal) relative to indeterministic data. Hence, the solution attribute `inus` makes explicit whether an *asf* or *csf* is an INUS solution. Moreover, the `cna()` and the `csf()` functions both have an argument `inus.only` controlling whether only INUS solutions (MINUS-formulas) shall be built or whether non-INUS solutions shall be issued as well (if they are inferable from data). As `inus.only` defaults to `TRUE`, standard calls of `cna()` and `csf()` will never yield non-INUS solutions, even if such solutions might be inferable from data. But if the analyst is interested in non-(M)INUS structures as well, she can request corresponding solutions by `inus.only = FALSE`.

To illustrate this, we first show that CNA never infers non-(M)INUS solutions from deterministic data—even if `inus.only = FALSE`. For this purpose, we simulate ideal data from the non-INUS solution in (4); `cna(..., inus.only = FALSE)` will always, i.e. upon an open number of re-runs of the following code chunk, return $A + B \leftrightarrow C$, regardless of the fact that we select cases based on (4) with `selectCases1("A + a*B <-> C", ...)`.

```
R> dat.inu.1 <- allCombs(c(2, 2, 2)) -1
R> dat.inu.2 <- some(dat.inu.1, 40, replace = TRUE)
R> dat.inu.3 <- selectCases1("A + a*B <-> C",  con = 1, cov = 1, dat.inu.2)
R> asf(cna(dat.inu.3, con = 1, cov = 1, inus.only = FALSE))

  outcome condition   consistency coverage complexity inus
1 C       A + B <-> C           1        1          2 TRUE
```

But in real-life discovery contexts, especially in observational studies, deterministic dependencies are the exception rather than the norm. Ordinary (observational) data are indeterministic, meaning that causes tend to be combined both with the presence and the absence of outcomes. In such discovery contexts, which can be simulated by lowering `con` and `cov` in `selectCases1()`, non-INUS expressions may count as minimally necessary disjunctions of minimally sufficient conditions. Correspondingly, if `inus.only` is set to `FALSE`, `cna()` may infer non-INUS solutions:

```
R> set.seed(26)
R> dat.inu.4 <- some(dat.inu.1, 40, replace = TRUE)
```

```
R> dat.inu.5 <- selectCases1("A + a*B <-> C", con = .8, cov = .8, dat.inu.4)
R> asf(cna(dat.inu.5, con = .8, cov = .8, inus.only = FALSE))

  outcome condition     consistency coverage complexity  inus
1 C       A + a*B <-> C       0.826    0.826          3 FALSE
```

In indeterministic data, it can happen that a is needed to lift the consistency of B above the chosen `con` threshold. In such a case, a can be argued to make a difference to C: only in conjunction with a does B reach `con`; and this holds notwithstanding the fact that A itself also meets `con`. Or put differently, when the consistency of $A \rightarrow C$ is below 1, there exist cases where A is instantiated and C is not, which, in turn, renders it possible for a change from A to a, while B is constantly instantiated, to be associated with a change in C, meaning that a can be a difference-maker for C.

If non-INUS expressions can be inferred from indeterministic data, the crucial follow-up question is whether the indeterminism in the data is due to insufficient control of background influences (i.e. to noise, measurement error, etc.) or to the inherent indeterministic nature of the physical processes themselves (as can e.g. be found in the domain of quantum mechanics, cf. Albert 1992). If the latter is the case, the difference-making relations stipulated by non-INUS solutions should be taken seriously. By contrast, if the former is the case (i.e. the indeterminism is due to noise), the difference-making relations entailed by non-INUS solutions are mere artifacts of the noise in the data, meaning that they would disappear if the corresponding causal structure were investigated under more ideal discovery circumstances. In that case, which obtains in the macro domains to which CNA is typically applied, the argument `inus.only` should be left at its default value `TRUE` both in `cna()` and `csf()`. If `inus.only` is switched to `TRUE` in the `cna()`-call from the previous code chunk, no solution is returned any more, that is, there does not exist a MINUS-formula satisfying `con` and `cov` for `dat.inu.5`:

```
R> asf(cna(dat.inu.5, con = .8, cov = .8, inus.only = TRUE))

[1] outcome     condition   consistency coverage    complexity
[6] inus
<0 rows> (or 0-length row.names)
```

The function behind the solution attribute `inus` and the argument `inus.only` is also available as stand-alone function `is.inus()`. Logical redundancies as contained in non-INUS solutions can be eliminated by means of the function `minimalize()` (see the **cna** reference manual for details).

5.3. Exhaustiveness and faithfulness

Exhaustiveness and faithfulness are two measures of model fit that quantify the degree of correspondence between the configurations that are, in principle, compatible with a solution $\mathbf{m}$ and the configurations actually contained in the data from which $\mathbf{m}$ is derived. To demonstrate those measures, let $\mathbf{F_m}$ symbolize the set of factors with values contained in $\mathbf{m}$. Exhaustiveness is high when *all* or most configurations of the factors in $\mathbf{F_m}$ that are *compatible* with $\mathbf{m}$ are actually contained in the data. More specifically, it amounts to the

ratio of the number of configurations over $\mathbf{F_m}$ in the data that are compatible with $\mathbf{m}$ to the total number of configurations over $\mathbf{F_m}$ that are compatible with $\mathbf{m}$. To illustrate, consider `d.educate`, which contains all configurations that are compatible with the two *csf* issued by `cna()` and `csf()`:

```
R> printCols <- c("condition", "consistency", "coverage", "exhaustiveness")
R> csf(cna(d.educate, details = "exhaust"))[printCols]

  condition                        consistency coverage exhaustiveness
1 (L + G <-> E)*(U + D <-> L)                1        1              1
2 (U + D + G <-> E)*(U + D <-> L)            1        1              1
```

If, say, the first configuration in `d.educate` (*viz.* $U{*}D{*}L{*}G{*}E$) is not observed or removed—as in `d.educate[-1,]`—, `cna()` still builds the same solutions (with perfect consistency and coverage). In that case, however, the resulting *csf* are not exhaustively represented in the data, for one configuration that is compatible with both *csf* is not contained therein.

```
R> csf(cna(d.educate[-1,], details = "exhaust"))[printCols]

  condition                        consistency coverage exhaustiveness
1 (L + G <-> E)*(U + D <-> L)                1        1          0.875
2 (U + D + G <-> E)*(U + D <-> L)            1        1          0.875
```

In a sense, faithfulness is the complement of exhaustiveness. It is high when *no* or only few configurations of the factors in $\mathbf{F_m}$ that are *incompatible* with $\mathbf{m}$ are in the data. More specifically, faithfulness amounts to the ratio of the number of configurations over $\mathbf{F_m}$ in the data that are compatible with $\mathbf{m}$ to the total number of configurations over $\mathbf{F_m}$ in the data. The two *csf* resulting from `d.educate` also reach perfect faithfulness:

```
R> printCols <- c("condition", "consistency", "coverage", "faithfulness")
R> csf(cna(d.educate, details = "faithful"))[printCols]

  condition                        consistency coverage faithfulness
1 (L + G <-> E)*(U + D <-> L)                1        1            1
2 (U + D + G <-> E)*(U + D <-> L)            1        1            1
```

If we add a configuration that is not compatible with these *csf*, say, $U{*}D{*}l{*}G{*}e$ and lower the consistency threshold, the same solutions along with one other result—this time, however, with non-perfect faithfulness scores.

```
R> csf(cna(rbind(d.educate,c(1,1,0,1,0)), con = .8, details = "f"))[printCols]

  condition                        consistency coverage faithfulness
1 (L + G <-> E)*(U + D <-> L)            0.857        1        0.889
2 (U + D + G <-> E)*(E <-> L)            0.857        1        0.778
3 (U + D + G <-> E)*(U + D <-> L)        0.857        1        0.889
```

If both exhaustiveness and faithfulness are high, the configurations over $\mathbf{F_m}$ in the data are all and only the configurations of the factors in $\mathbf{F_m}$ that are compatible with $\mathbf{m}$. Low exhaustiveness and/or faithfulness, by contrast, means that the data do not contain many configurations of the factors in $\mathbf{F_m}$ compatible with $\mathbf{m}$ and/or the data contain many configurations not compatible with $\mathbf{m}$. In general, solutions with higher exhaustiveness and faithfulness scores are preferable over solutions with lower scores.

5.4. Coherence

Coherence is a measure for model fit that is custom-built for *csf*. It measures the degree to which the *asf* combined in a *csf* cohere, that is, are instantiated together in the data rather than independently of one another. Coherence is intended to capture the following intuition. Suppose a *csf* entails that A is a sufficient cause of B, which, in turn, is entailed to be a sufficient cause of C. Corresponding data δ should be such that the $A-B$ link of that causal chain and the $B-C$ link are either both instantiated or both not instantiated in the cases recorded in δ. By contrast, a case in δ such that, say, only the $A-B$ link is instantiated but the $B-C$ link is not, pulls down the coherence of that *csf*. The more such non-cohering cases are contained in δ, the lower the overall coherence score of the *csf*.

Coherence is more specifically defined as the ratio of the number of cases satisfying all *asf* contained in a *csf* to the number of cases satisfying at least one *asf* in the *csf*. More formally, let a *csf* contain $asf_1, asf_2, \ldots, asf_n$, coherence then amounts to (where $|\ldots|_\delta$ represents the cardinality of the set of cases in δ satisfying the corresponding expression):

$$\frac{|\, asf_1 * asf_2 * \ldots * asf_n \,|_\delta}{|\, asf_1 + asf_2 + \ldots + asf_n \,|_\delta}$$

To illustrate, we add a case of type $U*d*L*g*e$ to `d.educate`. When applied to the resulting data (`d.edu.exp1`), `cna()` and `csf()` issue two *csf*.

```
R> d.edu.exp1 <- rbind(d.educate, c(1,0,1,0,0))
R> printCols <- c("condition", "consistency", "coverage", "coherence")
R> csf(cna(d.edu.exp1, con = .8, details = "cohere"))[printCols]

  condition                          consistency coverage coherence
1 (L + G <-> E)*(U + D <-> L)              0.875        1     0.889
2 (U + D + G <-> E)*(U + D <-> L)          0.875        1     0.889
```

In the added case, none of these two *csf* cohere, as only one of their component *asf* is satisfied.

Coherence is an additional parameter of model fit that allows for selecting among multiple solutions: the higher the coherence score of a *csf*, the better the overall model fit.

5.5. Structural redundancies and partial structural redundancies

It is not only possible that Boolean expressions describing the behavior of single outcomes contain redundant proper parts, but such expressions can themselves—as a whole—be redundant in superordinate structures. For instance, when three *asf* are conjunctively concatenated to $asf_1 * asf_2 * asf_3$ in the first step of stage 4 of the CNA algorithm (see section 4), it can

happen that $asf_1 * asf_2 * asf_3$ is logically equivalent to $asf_1 * asf_2$, meaning that asf_3 makes no difference to accounting for the behavior of the outcomes in that structure and is, thus, redundant. This is called a *structural redundancy* (for a detailed discussion see Baumgartner and Falk 2023b).

This type of redundancy is best introduced with a concrete example. Consider the following complex MINUS-formula:

$$(A{*}B + C \leftrightarrow D) * (a + c \leftrightarrow E) \tag{5}$$

(5) represents a causal structure such that $A{*}B$ and C are two alternative causes of D and a and c are two alternative causes of E. That is, the presence of A and C is relevant to D and their absence is relevant to E. A possible interpretation of these factors might be the following. Suppose a city has two power stations: a wind farm and a nuclear plant. Let A express that the wind farm is operational and C that the nuclear plant is operational and let operationality be sufficient for a nuclear plant to produce electricity, while a wind farm produces electricity provided it is operational and there is wind (B). Hence, the wind farm being operational while it is windy or the nuclear plant being operational ($A{*}B + C$) are two alternative causes of the city being power supplied (D). Whereas the wind farm or the nuclear plant not being operational ($a + c$) are two alternative causes of an alarm being triggered (E).

The following data (`dat.redun`) comprise all and only the configurations that are compatible with (5):

```
R> (dat.redun <- ct2df(selectCases("(A*B + C <-> D)*(a + c <-> E)")))
```

```
   A B C D E
2  0 1 1 1 1
4  0 0 1 1 1
5  1 1 0 1 1
14 0 1 0 0 1
15 1 0 0 0 1
16 0 0 0 0 1
17 1 1 1 1 0
19 1 0 1 1 0
```

The problem now is that `dat.redun` does not only entail the two *asf* contained in (5), *viz.* (6) and (7), but also a third one, *viz.* (8):

$$A{*}B + C \leftrightarrow D \tag{6}$$

$$a + c \leftrightarrow E \tag{7}$$

$$a{*}D + e \leftrightarrow C \tag{8}$$

That means the behavior of C, which is exogenous in the data-generating structure (5), can be expressed as a redundancy-free Boolean function of its two effects D and E. (8), hence, amounts to an upstream (or backtracking) *asf*, which, obviously, must not be causally interpreted. Indeed, when (8) is embedded in the superordinate dependency structure (9) that results from a conjunctive concatenation of all *asf* that follow from `dat.redun`, it turns out

that (8) is redundant. The reason is that (9) has a proper part which is logically equivalent to (9), namely (5).

$$(A{*}B + C \leftrightarrow D) * (a + c \leftrightarrow E) * (a{*}D + e \leftrightarrow C) \tag{9}$$

(9) and (5) state exactly the same about the behavior of the factors in `dat.redun`, meaning that (8) makes no difference to that behavior over and above (6) and (7). By contrast, neither (6) nor (7) can be eliminated from (9) such that the remaining expression is logically equivalent to (9). Both of these downstream *asf* make their own distinctive difference to the behavior of the factors in `dat.redun`. The upstream *asf* (8), however, is a *structural redundancy* in (9). (9) must not be causally interpreted because it is not a complex MINUS-formula (see p. 8 above).

Accordingly, in its default setting, the `csf()` function, which performs stage 4 of the CNA algorithm, removes all structurally redundant *asf* from conjunctions of *asf*; that is, when applied to `dat.redun` it returns (5), not (9):

```
R> printCols <- c("condition", "consistency", "coverage", "inus", "redundant")
R> csf(cna(dat.redun, details = "r"))[printCols]
```

```
  condition                      consistency coverage inus redundant
1 (C + A*B <-> D)*(a + c <-> E)           1        1 TRUE     FALSE
```

In previous versions (< 3.0) of the **cna** package, structurally redundant *asf* were not automatically removed but only marked by means of the solution attribute `redundant`. The solutions with `redundant = TRUE` then had to be further processed by the function `minimalizeCsf()`. To reproduce that old behavior, `csf()` now has an additional argument `minimalizeCsf`, which defaults to `TRUE`. If set to `FALSE`, structural redundancies are not automatically eliminated. Accordingly, the following call returns (9) and marks it as containing a structural redundancy—and thus as not having (M)INUS form:

```
R> csf(cna(dat.redun, details = "r"), inus.only = FALSE,
+      minimalizeCsf = FALSE)[printCols]
```

```
  condition                                          consistency coverage
1 (e + a*D <-> C)*(C + A*B <-> D)*(a + c <-> E)           1        1
   inus redundant
1 FALSE      TRUE
```

As *csf* with `redundant = TRUE` must never be causally interpreted, the setting `minimalizeCsf = FALSE` is deprecated. It is mainly kept in the package for backwards compatibility and developing purposes. Correspondingly, the solution attribute `redundant` is no longer relevant, as `cna()` and `csf()` no longer output *csf* with structural redundancies in the first place.

While structural redundancies of whole *asf* can occur in both deterministic and indeterministic data, the latter type of data may induce yet another, but related, type of redundancy. When data do not feature strict Boolean dependencies, building *csf* from the inventory of *asf* recovered in stage 3 of the CNA algorithm may lead to the redundancy of proper parts

of *asf*—parts which are not redundant when those *asf* are considered in isolation. That is, a complex structure can entail that one of its *asf* has a redundant proper part, which redundancy, however, is not visible in the data. We call this a *partial structural redundancy*.

Again, a concrete example helps to clarify the problem. Consider the solutions resulting from the following analysis of data `d.autonomy`:

```
R> printCols <- c("condition", "consistency", "coverage", "inus")
R> csf(cna(d.autonomy, ordering = "AU", con = .9, cov = .94,
+     maxstep = c(2, 2, 8), inus.only = FALSE))[printCols]
```

```
  condition                                    consistency coverage  inus
1 (SP*RE + CO*cn <-> EM)*(ci + EM*co <-> SP)       0.938    0.947  TRUE
2 (SP*co + CO*cn <-> EM)*(ci + EM*co <-> SP)       0.938    0.947  TRUE
3 (SP*RE + ci*cn <-> EM)*(ci + EM*co <-> SP)       0.938    0.947 FALSE
4 (SP*RE + CO*cn <-> EM)*(ci + EM*RE <-> SP)       0.928    0.947  TRUE
5 (SP*co + CO*cn <-> EM)*(ci + EM*RE <-> SP)       0.928    0.947  TRUE
6 (SP*RE + ci*cn <-> EM)*(ci + EM*RE <-> SP)       0.928    0.947 FALSE
```

Solution #3 in that list logically entails (10):

$$(SP{*}RE + ci{*}cn \leftrightarrow EM){*}(ci + EM \leftrightarrow SP) \tag{10}$$

That is, if the behavior of EM is regulated by the first *asf* in solution #3 and (10), *co* in #3—for pure logical reasons—cannot make a difference to SP and, hence, is redundant. That partial structural redundancy, however, is not visible in the data `d.autonomy` where EM alone (i.e. without *co*) is not sufficient for SP with consistency 0.9, which is the threshold chosen for the above analysis:

```
R> condTbl("EM -> SP", fsct(d.autonomy))
```

```
  outcome condition consistency coverage complexity
1 SP      EM -> SP       0.891    0.863          1
```

Hence, the data suggest that *co* makes a difference to SP, to the effect that meeting `con` $= 0.9$ for all *msc* requires $EM{*}co$ (and not EM alone) to be treated as cause of SP. At the same time, that (10) logically follows from solution #3 implies that it is logically excluded that *co* is a difference-maker of SP in the context of solution #3. The result is a contradiction: the data call for including *co* as cause of SP, whereas the structure inferred from that data entails not to include *co*.

The case of solution #6 is analogous. Solution #6 not only entails (10) but is logically equivalent to it. The upshot is the same: the data determine that some causal relevance relation obtains, which is logically excluded by the very structure inferred from the data. Such inconsistencies cannot be resolved by modifying solutions #3 and #6; rather, these solutions are not, and cannot be transformed into, well-formed MINUS-formulas that would meet `con`. They must be eliminated from the output. This is exactly what happens if `cna()` and `csf()` are run with the default `inus.only = TRUE`:

```
R> csf(cna(d.autonomy, ordering = "AU", con = .9, cov = .94,
+     maxstep = c(2, 2, 8), inus.only = TRUE))[printCols]
```

```
  condition                                    consistency coverage inus
1 (SP*RE + CO*cn <-> EM)*(ci + EM*co <-> SP)       0.938    0.947 TRUE
2 (SP*co + CO*cn <-> EM)*(ci + EM*co <-> SP)       0.938    0.947 TRUE
3 (SP*RE + CO*cn <-> EM)*(ci + EM*RE <-> SP)       0.928    0.947 TRUE
4 (SP*co + CO*cn <-> EM)*(ci + EM*RE <-> SP)       0.928    0.947 TRUE
```

In sum, as of version 3.0 of the **cna** package, both structural and partial structural redundancies are automatically resolved and eliminated. The functions `cna()` and `csf()` now exclusively output MINUS-formulas (i.e. INUS solutions).

5.6. Cycles

Detecting causal cycles is one of the most challenging tasks in causal data analysis—in all methodological traditions. One reason is that factors in a cyclic structure are so highly interdependent that, even under optimal discovery conditions, the diversity of (observational) data tends to be too limited to draw informative conclusions about the data-generating structure. Various methods in fact assume that analyzed data-generating structures are acyclic (most notably, Bayes nets methods, cf. Spirtes *et al.* 2000).

`cna()` and `csf()` output cyclic *csf* if they fit the data. The optional solution attribute `cyclic` identifies those *csf* that contain a cyclic substructure. A causal structure has a cyclic substructure if, and only if, it contains a directed causal path from at least one cause back to itself. The MINUS theory spells this criterion out more explicitly as follows:

Cycle A complex MINUS-formula **m** has a cyclic substructure if, and only if, **m** contains a sequence $\langle Z_1, Z_2, \ldots, Z_n \rangle$ such that every Z_i is contained in an atomic MINUS-formula of Z_{i+1} and $Z_1 = Z_n$ in **m**.

To illustrate, the *csf* inferred from the `d.autonomy` data in the previous section contain the cyclic sequence $\langle SP, EM, SP \rangle$ and, thus, represent causal cycles. Typically, when cyclic models fit the data, the output of `cna()` and `csf()` is very ambiguous. Therefore, if there are independent reasons to assume that the data are not generated by a cyclic structure, both `cna()` and `csf()` have the argument `acyclic.only`, which, if set to its non-default value `TRUE`, prevents solutions with cycles from being returned and, thereby, reduces model ambiguities. For example, by switching `acyclic.only` from `FALSE` to `TRUE` in the following analysis, the solution space is reduced from 31 to 9:

```
R> csf(cna(d.irrigate, con = .77, cov = .77, acyclic.only = F)) |> nrow()
```

```
[1] 31
```

```
R> csf(cna(d.irrigate, con = .77, cov = .77, acyclic.only = T)) |> nrow()
```

```
[1] 9
```

The `cycle.type` argument—also available in both `cna()` and `csf()`—controls whether a cyclic sequence $\langle Z_1, Z_2, \ldots, Z_n \rangle$ is composed of factors (`cycle.type = "factor"`), which is the default, or factor values (`cycle.type = "value"`). To illustrate, if `cycle.type = "factor"`, (11) counts as cyclic:

$$(A + B \leftrightarrow C) * (c + D \leftrightarrow A) \tag{11}$$

The factor A (with value 1) appears in an *asf* of C (i.e. C=1), and the factor C (with value 0) appears in an *asf* of A. But if `cycle.type = "value"`, (11) does not pass as cyclic. Although A appears in an *asf* of C=1, that same value of C does not appear in an *asf* of A; rather, C=0 appears in the *asf* of A.

The function behind the solution attribute `cyclic` and behind the corresponding `cna()` arguments is also available as stand-alone function `cyclic()` (see the **cna** reference manual for details).

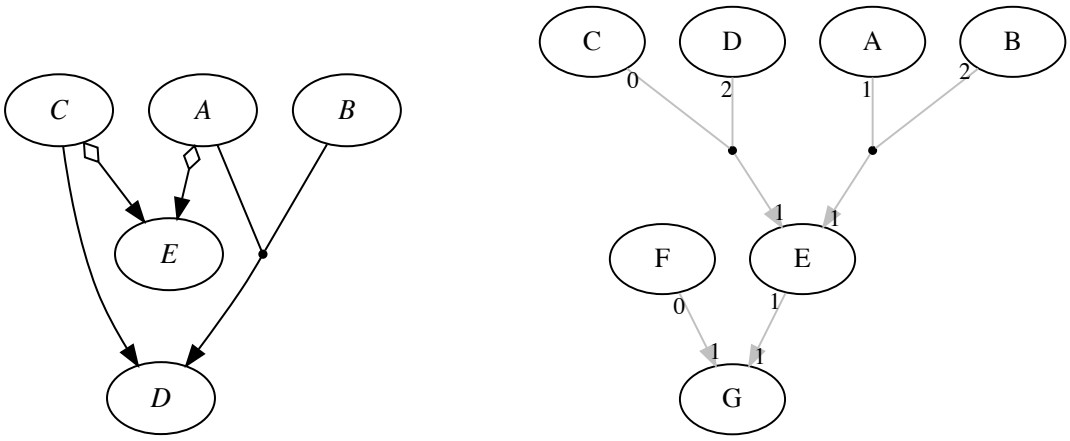
5.7. Plotting the output

MINUS-formulas can be visualized as causal hypergraphs, which are related to directed acyclic graphs (DAGs; Greenland *et al.* 1999; Spirtes *et al.* 2000), the most widely used tool for visualizing causal structures. But while edges in DAGs connect exactly two nodes, indicating the direction of causation, edges in hypergraphs can connect more than two nodes and, thereby, represent more than just the direction of causation. Causal hypergraphs can merge nodes into bundles and then connect these bundles to other nodes. This allows for representing conjunctive and disjunctive groupings of causes and, accordingly, for capturing the causal complexity encoded in MINUS-formulas. Furthermore, while DAGs are assumed not to contain cycles, causal hypergraphs may include cycles to accommodate the fact that MINUS-formulas may express causal feedback structures.

For convenience, we use the acronym CHG to refer to causal hypergraphs. A CHG is a pair $(\mathbf{F}, \mathbf{E})$, where $\mathbf{F}$ is a set of nodes and $\mathbf{E}$ is a set of ordered pairs of subsets of $\mathbf{F}$. Each of these ordered pairs $\langle \mathbf{A}, \mathbf{B} \rangle \in \mathbf{E}$ is called an *edge*, more specifically, an edge directed from $\mathbf{A}$ to $\mathbf{B}$. The subset $\mathbf{A}$ is called the *tail* of the edge, $\mathbf{B}$ is its *head*. The heads of edges in CHGs representing MINUS-formulas are always singleton sets, whereas their tails can contain multiple elements. Just as in DAGs, edges in CHGs represent the relation of direct causal relevance. But while nodes in DAGs represent factors or variables such as A and B, the nodes in CHGs represent factor values such as A and b. Hence, DAGs represent causal relationships between factors or variables, whereas CHGs represent causal relationships between factor values.

There are two types of CHGs: *set-CHGs* for causal structures involving values of crisp-set or fuzzy-set factors and *mv-CHGs* for structures of multi-value factors. Besides nodes and directed edges, set-CHGs contain two further graphical elements: "•" for bundling nodes in a conjunction and "◇" at tails of edges for negating factor values. Mv-CHGs also symbolize conjunction through "•", but instead of a negation sign, they feature numeric values directly assigned to the tails and heads of edges indicating the factor values that are connected by the edge. In both set- and mv-CHGs, edges with the same head form a disjunction. Figure 3a features an example of a set-CHG, and Figure 3b an example of an mv-CHG.

The R package **causalHyperGraph** provides functions to visualize the output of `cna()` as CHGs. The most basic plotting function is `plot(x)`, which takes a solution object `x` of `cna()`



(a) (b)

Figure 3: (a) is the causal hypergraph produced by `plot(ana.dat.redun)`. (b) is the output of `causalHyperGraph("(A=1*B=2 + C=0*D=2 <-> E=1)*(E=1 + F=0 <-> G=1)")`.

as input an draws the solution formulas contained in `x` as CHGs. For example, the following code draws the CHGs in Figure 3a from the *csf* inferred from `dat.redun` (see p. 30 above):

```
R> library(causalHyperGraph)
R> ana.dat.redun <- cna(dat.redun)
R> plot(ana.dat.redun)
```

There is also a function `causalHyperGraph(x)` that takes a character vector `x` expressing MINUS-formulas as input and draws the corresponding CHGs. To illustrate, the CHG in Figure 3b is drawn as follows:

```
R> causalHyperGraph("(A=1*B=2 + C=0*D=2 <-> E=1)*(E=1 + F=0 <-> G=1)")
```

6. Interpreting the output

The ultimate output of `cna()` and `csf()` is a set $\mathbf{csf_O}$ of *csf*—which may be identical to *asf*, if the data comprise only one endogenous factor. The causal inferences that are warranted based on the data input `x` relative to the chosen `con` and `cov` thresholds and the provided `outcome`, `ordering`, and `maxstep` have to be read off that set $\mathbf{csf_O}$. This section explains this final interpretative step of a CNA analysis.

There are three possible types of outputs:

1. $\mathbf{csf_O}$ contains no *csf* (and, correspondingly, no *asf*);
2. $\mathbf{csf_O}$ contains exactly one *csf* (and, correspondingly, exactly one *asf* for each endogenous factor);
3. $\mathbf{csf_O}$ contains more than one *csf* (and, correspondingly, more than one *asf* for at least one endogenous factor).

6.1. No solution

As indicated in section 3.4, a null result can have two sources: either the data are too noisy to render the chosen `con` and `cov` thresholds satisfiable or the selected `maxstep` is too low. If increasing `maxstep` does not yield solutions at the chosen `con` and `cov` thresholds, the latter may be lowered, preferably with a concomitant robustness analysis as described in section 3.2. If no solutions are recovered at `con = cov = .7`, the data are too noisy to warrant reliable causal inferences. Users are then advised to go back to the data and follow standard guidelines (known from other methodological frameworks) to improve data quality, e.g. by integrating further relevant factors into the analysis, enhancing the control of unmeasured causes, expanding the population of cases or disregarding inhomogeneous cases, correcting for measurement error, supplying missing values, etc.

It must be emphasized again (see section 2.4) that, under normal circumstances, an empty $\mathbf{csf_O}$ does not warrant the conclusion that the factors contained in the data input `x` are causally irrelevant to one another. The inference to causal irrelevance is much more demanding than the inference to causal relevance. A null result only furnishes evidence for causal irrelevance if there are independent reasons to assume that all potentially relevant factors are measured in `x` and that `x` exhausts the space of empirically possible configurations.

6.2. A unique solution

That $\mathbf{csf_O}$ contains exactly one *csf* is the optimal completion of a CNA analysis. It means that the data input `x` contains sufficient evidence for a determinate causal inference. The factor values on the left-hand sides of "↔" in the *asf* constituting that *csf* can be interpreted as causes of the factor values on the right-hand sides. Moreover, their conjunctive, disjunctive, and sequential groupings reflect the actual properties of the data-generating causal structure.

Plainly, as with any other method of causal inference, the reliability of CNA's causal conclusions essentially hinges on the quality of the processed data. If the data satisfy homogeneity (see section 2.4), a unique solution is guaranteed to correctly reflect the data-generating structure. With increasing data deficiencies (noise, measurement error, etc.), the (inductive) risk of committing causal fallacies inevitably increases as well. For details on the degree to which the reliability of CNA's causal conclusions decreases with increasing data deficiencies see (Baumgartner and Ambühl 2020) and (Parkkinen and Baumgartner 2023).

6.3. Multiple solutions

If $\mathbf{csf_O}$ has more than one element, the processed data underdetermine their own causal modeling. That means the evidence contained in the data is insufficient to determine which of the solutions contained in $\mathbf{csf_O}$ corresponds to the data-generating causal structure. An output set of multiple solutions $\{csf_1, csf_2, ..., csf_n\}$ is to be interpreted *disjunctively*: the data-generating causal structure is

$$csf_1 \text{ OR } csf_2 \text{ OR } ... \text{ OR } csf_n$$

but, based on the evidence contained in the data, it is ambiguous which of these csf is actually operative.

That empirical data underdetermine their own causal modeling is a very common phenomenon in all methodological traditions (Simon 1954; Spirtes *et al.* 2000, 59-72; Kalisch *et al.* 2012;

Eberhardt 2013; Baumgartner and Thiem 2017). But while some methods are designed to automatically generate all fitting models, e.g. Bayes nets methods and configurational comparative methods, other methods rely on search heuristics that zoom in on one best fitting model only, e.g. logic regression or regression analytic methods, more generally. Whereas model ambiguities have long been a thoroughly investigated topic in certain traditions, such as Bayes nets methods, they are only beginning to be studied in the literature on configurational comparative methods.

CNA—on a par with any other method—cannot disambiguate what is empirically underdetermined. Rather, it draws those and only those causal conclusions for which the data *de facto* contain evidence. In cases of empirical underdetermination it therefore renders transparent all data-fitting models and leaves the disambiguation up to the analyst.

That `cna()` and `csf()` issue multiple solutions for some data input `x` does not necessarily mean that `x` is deficient. In fact, even data that are *ideal* by all quality standards of configurational causal modeling can give rise to model ambiguities. The following simulates a case in point:

```
R> dat7 <- selectCases("a*B + A*b + B*C <-> D")
R> printCols <- c("condition", "consistency", "coverage", "inus",
+    "exhaustiveness")
R> csf(cna(dat7, details = c("exhaust", "inus")))[printCols]

  condition             consistency coverage inus exhaustiveness
1 a*B + A*b + A*C <-> D           1        1 TRUE               1
2 a*B + A*b + B*C <-> D           1        1 TRUE               1
```

`dat7` induces perfect consistency and coverage scores and is free of fragmentation; it contains all and only the configurations that are compatible with the target structure, which accordingly is exhaustively and faithfully reflected in `dat7`. Nonetheless, two models can be inferred. The causal structures expressed by these two models generate the exact same data, meaning they are *empirically indistinguishable*.

Although, a unique solution is more determinate and, thus, preferable to multiple solutions, the fact that `cna()` and `csf()` generate multiple equally data-fitting models is not generally an uninformative result. In the above example, both models feature $a{*}B + A{*}b$. That is, the data contain enough evidence to establish the joint relevance of $a{*}B$ and of $A{*}b$ for D (on alternative paths). What is more, it can be conclusively inferred that D has a further complex cause, *viz.* either $A{*}C$ or $B{*}C$. It is merely an open question which of these candidate causes is actually operative.

That different model candidates have some *msc* in common is a frequent phenomenon. Here's a real-life example, where two alternative causes, *viz.* $C{=}1 + F{=}2$, are present in all solutions:

```
R> csf(cna(d.pban, cov = .95, maxstep = c(3, 5, 10)))["condition"]

  condition
1 C=1 + F=2 + C=0*F=1 + C=2*V=0 <-> PB=1
2 C=1 + F=2 + C=0*T=2 + C=2*V=0 <-> PB=1
3 C=1 + F=2 + C=2*F=0 + C=0*F=1 + F=1*V=0 <-> PB=1
4 C=1 + F=2 + C=2*F=0 + C=0*T=2 + F=1*V=0 <-> PB=1
```

```
5 C=1 + F=2 + C=0*F=1 + C=2*T=1 + T=2*V=0 <-> PB=1
6 C=1 + F=2 + C=0*F=1 + T=1*V=0 + T=2*V=0 <-> PB=1
7 C=1 + F=2 + C=0*T=2 + C=2*T=1 + T=2*V=0 <-> PB=1
8 C=1 + F=2 + C=0*T=2 + T=1*V=0 + T=2*V=0 <-> PB=1
```

Such commonalities can be reported as conclusive results.

Moreover, even though multiple solutions do not permit pinpointing the causal structure behind an outcome, they nonetheless allow for constraining the range of possibilities. In a context where the causes of some outcome are unknown it amounts to a significant gain of scientific insight when a study can show that the structure behind that outcome has one of a small number of possible forms, even if it cannot determine which one exactly.

However, the larger the amount of data-fitting solutions and the lower the amount of commonalities among them, the lower the overall informativeness of a CNA output. Indeed, if data fragmentation is high, meaning if there are many unobserved possible configurations, the ambiguity ratio in configurational causal modeling can reach dimensions where nothing at all can be concluded about the data-generating structure any more. Hence, a highly ambiguous result is on a par with a null result. A telling example of this sort is `d.volatile` which was discussed in section 3.4 above (cf. also Baumgartner and Thiem 2017).

As the problem of model ambiguities is still under-investigated in the CNA literature, there do not yet exist conventionalized guidelines for how to proceed in cases of ambiguities. The model fit scores and solution attributes reported in the output objects of `cna()` and `csf()` often provide some leverage to narrow down the space of model candidates. For instance, if, in a particular discovery context, there is reason to assume that data have been collected as exhaustively as possible, to the effect that most configurations compatible with an investigated causal structure should be contained in the data, the model space may be restricted to *csf* with a high score on exhaustiveness. By way of example, for `d.pban` a total of 14 *csf* are built at `cov = .95`:

```
R> ana.ban.1 <- cna(d.pban, cov = .95, maxstep = c(6, 6, 10), details = T)
R> csf.ban.1 <- csf(ana.ban.1)
R> length(csf.ban.1$condition)

[1] 14
```

If only *csf* with `exhaustiveness >= .85` are considered, the amount of candidate *csf* is reduced to 2:

```
R> csf.ban.1.ex <- subset(csf.ban.1, exhaustiveness >= .85)
R> length(csf.ban.1.ex$condition)

[1] 2
```

To also resolve this final ambiguity, complexity may be brought to bear. Among equally data-fitting models the less complex ones are generally preferable because they are less likely to be overfitted and make less causal claims, resulting in a lower error risk. In the above example, if complexity is required to be as low as possible, only one model remains:

```
R> subset(csf.ban.1.ex, complexity == min(csf.ban.1.ex$complexity))

  outcome condition                                consistency coverage
1 PB=1    C=1 + F=2 + C=0*F=1 + C=2*V=0 <-> PB=1           1    0.952
  complexity inus exhaustiveness faithfulness coherence redundant cyclic
1          6 TRUE          0.889        0.941         1     FALSE  FALSE
```

Clearly though, the fit parameters and solution attributes provided by `cna()` and `csf()` will not always provide a basis for complete ambiguity elimination. The evidence contained in data is often insufficient to draw determinate causal conclusions. In such instances, data-external sources of information, such as prior causal knowledge or background theories, may be available that can be used to suitably constrain the search space of `cna()` via the arguments `outcome`, `ordering`, or `exclude` (see section 3.3 above). This tends to bring down ambiguities significantly. Moreover, the next section will show how knowledge about individual cases in the data can be leveraged to select among the model candidates. Nevertheless, it may also be impossible to resolve all ambiguities when the evidence in the data is complemented by data-external sources of information.

The most important course of action in the face of ambiguities is to *render them transparent*. By default, readers of CNA publications should be informed about the degree of ambiguity. Full transparency with respect to model ambiguities, first, allows readers to determine for themselves how much confidence to have in the conclusions drawn in a study, and second, paves the way for follow-up studies that are purposefully designed to resolve previously encountered ambiguities.

6.4. "Back to the cases"

When CNA is applied to small- or intermediate-N data, researchers may be familiar with some or all of the cases in their data. For instance, they may know that in a particular case certain causes of an outcome are operative while others are not. Or they may know why certain cases are outliers or why others feature an outcome but none of the potential causes. A proper interpretation of a CNA result may therefore require that the performance of the obtained models be assessed on the case level and against the background of the available case knowledge.

The function that facilitates the evaluation of recovered *msc*, *asf*, and *csf* on the case level is `condition(x, ct)`. Its first input is a character vector `x` specifying Boolean expressions—typically *asf* or *csf*—and its second input a data frame or configuration table `ct`. In case of *cs* or *mv* data, the output of `condition()` then highlights in which cases `x` is instantiated, whereas for fs data, the output lists relevant membership scores in exogenous and endogenous factors. Moreover, if `x` is an *asf* or *csf*, `condition()` issues their consistency and coverage scores.

To illustrate, we re-analyze `d.autonomy`:

```
R> dat.aut.2 <- d.autonomy[15:30, c("AU","EM","SP","CO","RE","DE")]
R> ana.aut.3 <- cna(dat.aut.2, outcome = c("EM","AU"), con = .91, cov = .91)
R> condition(csf(ana.aut.3)$condition, dat.aut.2)
```

That function call returns a list of three tables, each corresponding to one of the three *csf* contained in `ana.aut.3` and breaking down the relevant *csf* to the case level by contrasting

the membership scores in the left-hand and right-hand sides of the component *asf*. A case with a higher left-hand score is one that pulls down consistency, whereas a case with a higher right-hand score pulls down coverage. For each *csf*, `condition()` moreover returns overall consistency and coverage scores as well as consistency and coverage scores for the component *asf*.

The three *csf* in `ana.aut.3` differ only in regard to their component *asf* for outcome *AU*. The function `group.by.outcome(condlst)`, which takes an output object `condlst` of `condition()` as input, lets us more specifically compare these different *asf* with respect to how they fare on the case level.

```
R> group.by.outcome(condition(asf(ana.aut.3)$condition, dat.aut.2))$AU
```

```
          SP EM*RE+re*DE EM*RE+CO*DE  AU | n.obs
ENacg1   1.0         1.0         1.0 1.0 |     1
ENacg2   0.6         0.4         0.4 0.4 |     1
ENacg3   0.8         0.6         0.9 0.8 |     1
ENacg4   0.6         1.0         1.0 1.0 |     1
ENacg5   0.4         0.4         0.4 0.4 |     1
ENacg6   0.6         0.7         0.7 0.6 |     1
ENacg7   1.0         0.8         0.8 1.0 |     1
ENacg8   1.0         1.0         1.0 1.0 |     1
ENacto1  0.4         0.4         0.6 0.4 |     1
ENacto2  0.4         0.4         0.4 0.4 |     1
ENacosa1 0.4         0.4         0.2 0.4 |     1
ENacosa2 0.4         0.4         0.4 0.2 |     1
ENacosa3 0.4         0.4         0.4 0.6 |     1
ENacat1  0.4         0.4         0.4 0.2 |     1
ENacat2  0.4         0.4         0.4 0.6 |     1
ENacat3  0.4         0.6         0.4 0.4 |     1
```

The first three columns of that table list the membership scores of each case in the left-hand sides of the *asf*, and the fourth column reports the membership scores in *AU*. The table shows that the first *asf* ($SP \leftrightarrow AU$) outperforms the other *asf* in cases ENacg3/6/7, ENacto1, ENacosa1, and ENacat3, while it is outperformed by another *asf* in cases ENacg2 and ENacg4. In all other cases, the three solution candidates fare equally. If the analyst is closely familiar with some of these cases, performance differences on the case level can help to choose among the candidates. For instance, if it is known that there are no other factors operative in case ENacg7 than the ones contained in `dat.aut.2`, it follows that ENacg7's full membership in *AU* must be brought about by *SP*—which, in turn, disqualifies the other solutions. By contrast, if the absence of other relevant factors can be assumed for case ENacg4, the *asf* featuring *SP* as cause of *AU* is disqualified.

7. Benchmarking

Benchmarking the reliability of a method of causal inference is an essential element of method development and validation. In a nutshell, it amounts to testing to what degree the benchmarked method recovers the true data-generating structure Δ or proper substructures of Δ

from data of varying quality. As Δ is not normally known in real-life discovery contexts, the reliability of a method cannot be assessed by applying it to real-life data. Instead, reliability benchmarking is done in so-called *inverse searches*, which reverse the order of causal discovery as it is commonly conducted in scientific practice. An inverse search comprises three steps:

(1) a data-generating causal structure Δ is presupposed/drawn (as ground truth),

(2) artificial data δ is simulated from Δ, possibly featuring various deficiencies (e.g. noise, fragmentation, measurement error etc.),

(3) δ is processed by the tested method in order to check whether its output meets the tested reliability benchmark.

A benchmark test can measure various properties of a method's output, for instance, whether it is error-free, correct or complete, etc. As real-life data are often fragmented, methods for MINUS discovery typically do not infer the complete Δ from a real-life δ but only proper substructures thereof (see section 2.4). Thus, since completeness is not CNA's primary aim, it should likewise not be the primary reliability benchmark for CNA; it is more important that its output scores high on *error-freeness* and *correctness*.

CNA's output, *viz.* the issued set $\mathbf{csf_O}$ of *csf*, is error-free iff it does not entail a causal claim that is false of the ground truth Δ (i.e. no false positive). That can be satisfied in two ways: either (i) $\mathbf{csf_O}$ is empty, meaning no causal inferences are drawn, or (ii) $\mathbf{csf_O}$ contains at least one[15] solution $\mathbf{m}_i$ that is correct of Δ, which is the case iff $\mathbf{m}_i$ is a submodel of Δ (for details on the submodel relation see section 2.4). So, $\mathbf{csf_O}$ satisfies the error-freeness benchmark iff it satisfies conditions (i) or (ii). With increasing stringency, $\mathbf{csf_O}$ can then be said to be correct of Δ iff condition (ii) is satisfied, meaning $\mathbf{csf_O}$ actually contains at least one solution $\mathbf{m}_i$ that is a submodel of Δ, and thus correct. Finally, completeness measures the informativeness of $\mathbf{csf_O}$, that is, the ratio of causal properties of Δ captured and revealed by the solutions in $\mathbf{csf_O}$.

The **cna** package provides many functionalities to conduct inverse searches that are tailor-made to benchmark the output of `cna()` and `csf()`. The functions `randomAsf()` and `randomCsf()` can be used to draw a data-generating structure Δ in step (1). `randomAsf(x)` generates a structure with a single outcome (i.e. a random *asf*) and `randomCsf(x)` an acyclic multi-outcome structure (i.e. a random *csf*), where `x` is a data frame or `configTable` defining the factors and their possible values from which the structures are drawn. The function `selectCases()`, which has already been discussed in section 3.1.2, can be employed to simulate data δ in the course of step (2). Finally, `is.submodel(x, y)` determines whether models are related by the submodel relation, which, in turn, helps in assessing whether $\mathbf{csf_O}$ is true of Δ. `is.submodel()` takes a character vector `x` of *asf* as first input and tests whether the elements of that vector are submodels of `y`, which, in turn, is a character string of length 1 representing the target *asf* (i.e. Δ). If Δ is a *csf* with multiple outcomes, the function `causal_submodel(x, y)` from the **frscore** package should be used to determine whether `x` is true of `y`. Moreover, the function `identical.model(x, y)` is available to check whether `x` (which must have length 1) and `y` are identical.

Against that background, the following might be a core of a error-freeness benchmark test that simulates multi-value data with 20% missing observations and 5% random noise (i.e.

[15] Recall from section 6.3 that an output containing multiple solutions is to be interpreted disjunctively; and a disjunction of solutions is true iff at least one solution is true.

cases incompatible with the ground truth), and that runs `cna()` and `csf()` at `con` = `cov` = 0.7 without giving the algorithm prior causal information through the arguments `outcome`, `ordering`, or `exclude`.

```
R> # Draw a ground truth.
R> fullData <- allCombs(c(4,4,4,4,4))
R> groundTruth <- randomCsf(fullData, n.asf = 2, compl = 2)
R> # Generate ideal data for groundTruth.
R> idealData <- ct2df(selectCases(groundTruth, fullData))
R> # Introduce 20% fragmentation.
R> fragData <- idealData[-sample(1:nrow(idealData), nrow(idealData)*0.2), ]
R> # Add 5% random noise (cases incompatible with ground truth).
R> incompCases <- dplyr::setdiff(fullData, idealData)
R> x <- rbind(incompCases[sample(1:nrow(incompCases),
+     nrow(fragData) * 0.05), ], fragData)
R> # Run CNA without an ordering.
R> csfs <- csf(cna(x, con = .7, cov = .7, maxstep = c(3, 3, 12)))
R> # Check whether no causal error (no false positive) is returned.
R> if(length(csfs$condition)==0) {
+    TRUE } else {any(unlist(lapply(csfs$condition,
+    function(x) frscore::causal_submodel(x, groundTruth, fullData))))}
```

Every re-run of this code chunk generates a different ground truth and different data; in some runs CNA passes the test, in others it does not. To determine CNA's error-freeness ratio under these test conditions, the above core must be embedded in a suitable test loop. To estimate CNA's overall error-freeness ratio, the test conditions should be systematically varied by, for instance, varying the complexity of the ground truth, the degree of fragmentation and noise, the consistency and coverage thresholds, or by drawing the noise with a bias or supplying CNA with more or less prior causal information via `outcome`, `ordering`, or `exclude`. Correctness and completeness tests can be designed analogously, by suitably modifying the last line that evaluates the solution object `csfs`. For single-outcome structures (*asf*), benchmark tests with some of the above variations have been conducted in (Baumgartner and Ambühl 2020) and (Baumgartner and Falk 2023a); corresponding tests for multi-outcome structures (*csf*) have been carried out in (Parkkinen and Baumgartner 2023).

8. Summary

This vignette introduced the theoretical foundations as well as the main functions of the **cna** R package for configurational causal inference and modeling with Coincidence Analysis (CNA). Moreover, we explained how to interpret the output of CNA, provided some guidance for how to use various model fit parameters for the purpose of ambiguity reduction, and supplied a benchmarking template.

CNA is currently the only method searching for (M)INUS causation in data that builds multi-outcome models and, hence, not only orders causes conjunctively and disjunctively but also sequentially. Moreover, it builds causal models on the basis of a bottom-up algorithm that is unique among configurational comparative methods and gives CNA an edge over other

methods in guaranteeing the redundancy-freeness of its models, which, in turn, is crucial for their causal interpretability. Overall, CNA constitutes a powerful methodological alternative for researchers interested in causal structures affected by conjunctivity and disjunctivity. The **cna** package makes that inferential power available to end-users.

Acknowledgments

We are grateful to Alrik Thiem, Martyna Swiatczak, Jonathan Freitas, and Luna De Souter for helpful comments on earlier drafts of this vignette, and we thank the Toppforsk-program of the Trond Mohn Foundation and the University of Bergen (grant nr. 811886), the Research Council of Norway (grant nr. 326215), and the Swiss National Science Foundation (grant nr. PP00P1_144736/1) for generous support of the research behind the **cna** package over the years.

References

Albert DZ (1992). *Quantum Mechanics and Experience*. Harvard University Press, Cambridge.

Ambühl M, Baumgartner M (2022). ***cnaOpt**: Optimizing Consistency and Coverage in Configurational Causal Modeling*. R Package Version 0.5.2. https://cran.r-project.org/package=cnaOpt.

Baumgartner M (2009a). "Inferring Causal Complexity." *Sociological Methods & Research*, **38**, 71–101.

Baumgartner M (2009b). "Uncovering Deterministic Causal Structures: A Boolean Approach." *Synthese*, **170**, 71–96.

Baumgartner M (2013). "A Regularity Theoretic Approach to Actual Causation." *Erkenntnis*, **78**, 85–109.

Baumgartner M (2015). "Parsimony and Causality." *Quality & Quantity*, **49**, 839–856.

Baumgartner M (2020). "Causation." In D Berg-Schlosser, B Badie, L Morlino (eds.), *The SAGE Handbook of Political Science*, pp. 305–321. SAGE, London.

Baumgartner M, Ambühl M (2020). "Causal Modeling with Multi-Value and Fuzzy-Set Coincidence Analysis." *Political Science Research and Methods*, **8**, 526–542. doi:10.1017/psrm.2018.45.

Baumgartner M, Ambühl M (2021). "Optimizing Consistency and Coverage in Configurational Causal Modeling." *Sociological Methods & Research*, **52**(3), 1288–1320. doi:10.1177/0049124121995554.

Baumgartner M, Falk C (2023a). "Configurational Causal Modeling and Logic Regression." *Multivariate Behavioral Research*, **58**(2), 292–310. doi:10.1080/00273171.2021.1971510.

Baumgartner M, Falk C (2023b). "Boolean Difference-Making: A Modern Regularity Theory of Causation." *The British Journal for the Philosophy of Science*, **74**(1), 171–197. doi:10.1093/bjps/axz047.

Baumgartner M, Thiem A (2017). "Model Ambiguities in Configurational Comparative Research." *Sociological Methods & Research*, **46**(4), 954–987.

Baumgartner M, Thiem A (2020). "Often Trusted But Never (Properly) Tested: Evaluating Qualitative Comparative Analysis." *Sociological Methods & Research*, **49**, 279–311. doi:10.1177/0049124117701487.

Beirlaen M, Leuridan B, Van De Putte F (2018). "A Logic For the Discovery of Deterministic Causal Regularities." *Synthese*, **195**(1), 367–399. doi:10.1007/s11229-016-1222-x.

Bowran AP (1965). *A Boolean Algebra. Abstract and Concrete*. Macmillan, London.

Brambor T, Clark WR, Golder M (2006). "Understanding Interaction Models: Improving Empirical Analyses." *Political Analysis*, **14**(1), 63–82. doi:10.1093/pan/mpi014.

Braumoeller B (2015). ***QCAfalsePositive**: Tests for Type I Error in Qualitative Comparative Analysis (QCA)*. R package version 1.1.1, https://CRAN.R-project.org/package=QCAfalsePositive.

Cronqvist L, Berg-Schlosser D (2009). "Multi-Value QCA (mvQCA)." In B Rihoux, CC Ragin (eds.), *Configurational Comparative Methods: Qualitative Comparative Analysis (QCA) and Related Techniques*, pp. 69–86. Sage Publications, London.

Csikszentmihalyi M (1975). *Beyond Boredom and Anxiety*. Jossey-Bass Publishers, San Francisco.

Culverhouse R, Suarez BK, Lin J, Reich T (2002). "A Perspective on Epistasis: Limits of Models Displaying No Main Effect." *The American Journal of Human Genetics*, **70**(2), 461–471. doi:10.1086/338759.

De Souter L (2024). "Evaluating Boolean Relationships in Configurational Comparative Methods." *Journal of Causal Inference*, **12**(1). doi:10.1515/jci-2023-0014.

Dusa A (2024). ***QCA**: A Package for Qualitative Comparative Analysis*. R Package Version 3.22. https://cran.r-project.org/package=QCA.

Eberhardt F (2013). "Experimental Indistinguishability of Causal Structures." *Philosophy of Science*, **80**(5), 684–696.

Graßhoff G, May M (2001). "Causal Regularities." In W Spohn, M Ledwig, M Esfeld (eds.), *Current Issues in Causation*, pp. 85–114. Mentis, Paderborn.

Greenland S, Pear J, Robins JM (1999). "Causal Diagrams for Epidemiologic Research." *Epidemiology*, **10**(1), 37–48.

Hájek P (1998). *Metamathematics of Fuzzy Logic*. Kluwer, Dordrecht.

Hume D (1999 (1748)). *An Enquiry Concerning Human Understanding*. Oxford University Press, Oxford.

Kalisch M, Maechler M, Colombo D, Maathuis MH, Buehlmann P (2012). "Causal Inference Using Graphical Models With the R Package **pcalg**." *Journal of Statistical Software*, **47**(11), 1–26.

Kooperberg C, Ruczinski I (2005). "Identifying Interacting SNPs Using Monte Carlo Logic Regression." *Genetic Epidemiology*, **28**(2), 157–170. doi:10.1002/gepi.20042.

Kooperberg C, Ruczinski I (2023). ***LogicReg**: Logic Regression*. R package version 1.6.6. https://CRAN.R-project.org/package=LogicReg.

Lemmon EJ (1965). *Beginning Logic*. Chapman & Hall, London.

Mackie JL (1974). *The Cement of the Universe. A Study of Causation*. Clarendon Press, Oxford.

Oana IE, Medzihorsky J, Quaranta M, Schneider CQ (2023). ***SetMethods**: Functions for Set-Theoretic Multi-Method Research and Advanced QCA*. R package version 4.0, https://CRAN.R-project.org/package=SetMethods.

Parkkinen VP, Baumgartner M (2023). "Robustness and Model Selection in Configurational Causal Modeling." *Sociolocial Methods & Research*, **52**(1), 176–208.

Parkkinen VP, Baumgartner M (2024). ***frscore**: Functions for Calculating Fit-Robustness of CNA-solutions*. R Package Version 0.4.1. https://CRAN.R-project.org/package=frscore.

Ragin CC (2006). "Set Relations in Social Research: Evaluating Their Consistency and Coverage." *Political Analysis*, **14**(3), 291–310.

Ragin CC (2008). *Redesigning Social Inquiry: Fuzzy Sets and Beyond*. University of Chicago Press, Chicago.

Rihoux B, Ragin CC (eds.) (2009). *Configurational Comparative Methods. Qualitative Comparative Analysis (QCA) and Related Techniques*. Sage, Thousand Oaks.

Ruczinski I, Kooperberg C, LeBlanc M (2003). "Logic Regression." *Journal of Computational and Graphical Statistics*, **12**(3), 475–511. doi:10.1198/1061860032238.

Schneider CQ, Wagemann C (2012). *Set-Theoretic Methods: A User's Guide for Qualitative Comparative Analysis (QCA) and Fuzzy-Sets in the Social Sciences*. Cambridge University Press, Cambridge.

Schwender H, Tietz T (2024). ***logicFS**: Identification of SNP Interactions*. R package version 2.24.0. https://doi.org/doi:10.18129/B9.bioc.logicFS.

Simon HA (1954). "Spurious Correlation: A Causal Interpretation." *Journal of the American Statistical Association*, **49**(267), 467–479.

Spirtes P, Glymour C, Scheines R (2000). *Causation, Prediction, and Search*. 2 edition. MIT Press, Cambridge.

Swiatczak MD (2021). "Different Algorithms, Different Models." *Quality & Quantity*, **56**(4), 1913–1937. doi:10.1007/s11135-021-01193-9.

Thiem A (2018). ***QCApro**: Advanced Functionality for Performing and Evaluating Qualitative Comparative Analysis*. R Package Version 1.1-2. https://CRAN.R-project.org/package=QCApro.

Thiem A, Duşa A (2013). *Qualitative Comparative Analysis with R: A User's Guide*. Springer, New York.

Whitaker RG, Sperber N, Birken S, Baumgartner M, Thiem A, Cragun D, Damschroder L, Miech E, Slade A (2020). "Coincidence Analysis: A New Method for Causal Inference in Implementation Science." *Implementation Science*, **15**. doi:10.1186/s13012-020-01070-3.

Yakovchenko V, Miech EJ, Chinman MJ, Chartier M, Gonzalez R, Kirchner JE, Morgan TR, Park A, Powell BJ, Proctor EK, Ross D, Waltz TJ, Rogal SS (2020). "Strategy Configurations Directly Linked to Higher Hepatitis C Virus Treatment Starts: An Applied Use of Configurational Comparative Methods." *Medical Care*, **58**(5). doi:10.1097/MLR.0000000000001296.

Affiliation:

Michael Baumgartner
University of Bergen
Department of Philosophy
Postboks 7805
5020 Bergen
Norway
E-mail: michael.baumgartner@uib.no
URL: https://m-baum.github.io

Mathias Ambühl
Consult AG Statistical Services
Tramstrasse 10
8050 Zürich
E-mail: mathias.ambuehl@consultag.ch