1-1982

A Proposal for an Algorithm for Generating Loopless or Recursive Models of Multi-Variate Data

Klaus Krippendorff

University of Pennsylvania, kkrippendorff@asc.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/asc_papers

Part of the Communication Commons

Recommended Citation (OVERRIDE)


This paper is posted at ScholarlyCommons. http://repository.upenn.edu/asc_papers/208

For more information, please contact repository@pobox.upenn.edu
A Proposal for an Algorithm for Generating Loopless or Recursive Models of Multi-Variate Data

Abstract
Exploring multi-variate data involves searching for a simple model that simulates, explains or accounts for given data within tolerable margins of errors. Such a search requires procedures for generating suitable models that could be tested and compared. We are proposing here several rules for transforming the symbolic descriptions of a model which when inserted into a suitable algorithm will compute generations of structural models, one minimally simpler than the other. Although we are primarily concerned here with rules that generate models without loops for which algorithms are currently unavailable (Klir, 1980), we will contrast the transformation rules for this class of models with those responsible for generating all possible models and we will furthermore distinguish between rules that generate models with the same and with different covers.

Disciplines
Communication | Social and Behavioral Sciences

This book chapter is available at ScholarlyCommons: http://repository.upenn.edu/asc_papers/208
A PROPOSAL FOR AN ALGORITHM
FOR GENERATING LOOPLESS OR RECURSIVE MODELS
OF MULTI-VARIATE DATA

by

KLAUS KRIPPENDORFF
UNIVERSITY OF PENNSYLVANIA
PHILADELPHIA

ABSTRACT

Exploring multi-variate data involves searching for a simple model that simulates, explains or accounts for given data within tolerable margins of errors. Such a search requires procedures for generating suitable models that could be tested and compared. We are proposing several rules for transforming the symbolic descriptions of a model which when inserted into a suitable algorithm will compute generations of structural models, one minimally simpler than the other. Although we are primarily concerned here with rules that generate models without loops for which algorithms are currently unavailable (Klir, 1980), we will contrast the transformation rules for this class of models with those responsible for generating all possible models and we will furthermore distinguish between rules that generate models with the same and with different covers.

STRUCTURAL MODELS

The models of interest here are structural models which are characterized by a set of independent components or functional building blocks of the researcher's choice which represent different portions of observed reality. Each of these components is thus specified by a different set of observational variables, none of which is contained in another, and the dependencies between these components are indicated by the variables that two or more such components share. For example, the model:

ADF:BCDE:DEFG:GH:IJ

covers ten variables A through J. It has five components. The first is defined by the three variables ADF and is therefore said to be of ordi-
narily three. The next two components cover four variables each and are of order four, etc. The first three components are interdependent through the shared variable D, and the first and the third component are additionally interdependent through F, etc. The fifth and last component, consisting of the two unique variables I and J, is independent of the former four. We depict this model graphically:

and denote structural models more generally by:

\[ m = K_1: K_2: \ldots: K_i: \ldots: K_n \]

Although there may be many properties of models that may for various reasons appeal to a researcher (e.g. Klir, 1981b), in the following we will distinguish between two kinds of such properties yielding four classes of models altogether:

- All models
- Models without loops
- Models with loops and
- Models with same coces

Figure 1: Typology for four kinds of models

LOOPS

Probably the most important distinction a researcher may have to draw is between models that do contain loops and those that don't. In the econometric literature loopless models are called recursive models because their chain-like or linear nature allows them to be expanded by merely adding more and more components.

One and possibly the most important motivation for this distinction is computational. Models without loops can be evaluated by simple algebraical methods (specifically, there exist closed form estimates for obtaining their maximum entropy distribution) whereas for models with loops algebraical methods are unavailable and their evaluation must employ iterative procedures instead (Bishop, et al., 1975).

A second motivation for this distinction lies in the common use of structural models to explain causal dependencies. Causality cannot go backward in time and when observational variables exhibit
temporal orderings, the class of linear, chain-like or non-loop models may be more appealing than those containing circular dependencies. A researcher seeking causal explanations may want to restrict his search to models without loops which are simpler to compute and perhaps for this (not necessarily acceptable) reason preferred by many social scientists.

A third motivation is that a two-step search through the lattice of possible models—first among the easily computable loopless models and then among the circular models in the neighborhood of the best loopless-model—offers a short-cut towards finding a model with the best fit (Krippendorff, 1981a):

To test whether a model contains loops, we use a decision procedure (Krippendorff, 1981a) that is somewhat simpler than Bishop, et al.'s, (1975) yet yields the same results:

Given \( K_1, K_2, \ldots, K_i, K_j, \ldots \) of a model:

1. remove all variables that are unique to any \( K_i \)
2. remove any \( K_j \) that is equal to or contained in any other \( K_j \) of the set.

repeat (1) and (2) until either

(a) no variables remain in which case loops are absent
(b) the remainder is unalterable by the above in which case loops exist.

We note that our distinction does not coincide with Cavallo and Klir's (1979) definition of \( \gamma \)-structures that do not contain loops and non-\( \gamma \)-structures all of which are intended to contain loops but according to our distinction contain loopless models as well. An example, discussed by Cavallo and Klir (1981) to highlight the difference, is the model:

\[
\begin{array}{ccccccc}
 & A & B & D & E & C \\
K_1 & & & & & \\
K_2 & & & & & \\
K_3 & K_4 & & & & \\
\end{array}
\]

According to our decision procedure:

1. remove unique variables \( ABD: BDE: BCE \)
2. remove redundant components \( BB: BCE: BE \)
3. remove unique variables \( BD \)

in which case no loop is present. According to Cavallo and Klir (1979, 1981), since \( K_1 \) and \( K_3 \) is connected both directly via \( B \) and indirectly via \( D \) and \( E \) of \( K_4 \), the criterion for \( \gamma \)-structures is not met and a loop should exist. We maintain the position that in this instance \( B \) is shared by all components and its inclusion or exclusion does not alter the status of the model which is otherwise perfectly linear or chain-like at least as far as its recursive construction and its algebraic computability is concerned.

COVERS

The second distinction in our typology concerns whether the models of interest are all required to have the same cover:

\[
\begin{array}{ccccccc}
K_1 & K_2 & \ldots & K_i & \ldots & K_n \\
1 & 2 & \ldots & \gamma & \ldots & \nu \\
\end{array}
\]

or whether their cover is merely confined by a set of variables within which data are available for exploration. Regarding the role of this cover there are two distinct schools of thought.

The epistemological framework, adopted by Klir (e.g., 1976) and associates focuses attention on observed systems whose subsystems need to be reconstructed from the data they emit. In such a systems analysis, it is desirable that the hypothetical structures (our models) to be tested cover all variables that are to simulate, with the least complex or most simple structure being constituted by the set of statistically independent variables. Consequently, the work by Cavallo and Klir (1979) and associates accepts as "meaningful" only those structures whose cover is the same throughout, involving all variables of the observed system.

In contrast, just as we are willing to concede that the data we analyze may be representative of only a portion of reality (regardless of whether this reality conforms to the notion of a system), so we allow a model's cover to be less than the full range of observational variables. Variables may not only be either statistically related or independent of each other but, moreover, the distribution of observations within them may be so unremarkable that nothing needs to be said about them. From this epistemological perspective, which is the perspective of the modeler, irrelevant variables may well be omitted from a model that is to simulate, explain or account for what is truly unexpected or exceptional.

Clearly, the class of models with the same cover is a special case of the general class of all models whose cover is merely confined to being a subset of a specified set of variables.

GENERAL FORM OF THE ALGORITHM

In search of a model with the best fit, efficient procedures for generating the hypothetical models to be tested are essential. Conant's (1981) strategy is to start with a situation equivalent to what might be called the null-hypothesis of statistical independence among variables and iteratively computes components of increasing complexity. These separately computed components are then assembled to form a (by some defined criterion acceptable) model of the data. Although the efficiency of this strategy is promising, we follow a top-down strategy which starts with a sufficiently if not maximally complex model from which constraints, starting with that of the highest order, are successively removed, thus yielding alternative simplifications. Coupled with suitable procedures for evaluating the fit of a model, the strategy selects a path through the lattice of all models of a certain class. Klir and Uttenboe (1976), Broeksta (1978) and Krippendorff (1981a) have followed this strategy as well.
To compute alternative simplifications from a given model we are using here transformation rules or rewrite rules similar to those employed in constructing transformational grammars of natural languages. These rules are part of an algorithm which is stated here in general terms so as to allow us to concentrate on these rules in subsequent sections:

Given the model \( m = K_1 \cdot K_2 \cdot \ldots \cdot K_x \cdot K_y \cdot \ldots \cdot K_z \).

1. All permissible transformation rules:

   - All components of the model \( K_i \) are from \( m \)...

2. All ways of applying that transformation rule on \( K_i \):

   (1) replace \( K_i \) of the model \( m \) by the string \( K'_i \cdot K''_1 \cdot \ldots \cdot K''_y \).
   (2) remove any \( K_i \) of the string that is equal to or contained in any other \( K_j \) of \( m \).
   (3) consider the remaining \( K_i \) and \( K_j \) as the components of \( m \)’s simplifications.

(3) is merely clerical. (2) removes redundancies so that each of the simpler model’s components represents a different portion of reality without duplication as is required by structural models. The key to the algorithm lies in the permissible transformation rules according to which (1) breaks one component of the model into several simpler parts (that are thereby no longer capable of representing certain constraints embodied in the original component). Applied iteratively, on all simplifications, descendents or successors of a model, the algorithm generates a lattice of models from an initial and most complex (or least simple) model \( m \) as upper bound to a final and least complex (or most simple) model \( m = \o \) as lower bound. Obviously, the properties of the models of such a lattice depend on the choice of the upper bound of this lattice and on the nature of the transformation rules deemed permissible. In our algorithm, any decision on the kinds of models that a researcher might want to examine must find its reflection in the choice among appropriate transformation rules to which we now turn.

TRANSFORMATION RULE FOR ALL POSSIBLE MODELS

The least complex transformation rule is the one responsible for generating the class of all possible models (which may or may not contain loops and which cover a given set of variables and all subsets thereof). The transformation rule simply breaks a component into as many parts as there are variables in that component and thereby removes only the highest order interaction within that component. Let the component \( K_i \) have \( w \) variables \( 1', 1'', \ldots, 1''^w \), the rule is simply:

\[ K_i \rightarrow (K_i-1') \cdot (K_i-1'') \cdots \cdot (K_i-1''^w) \]  

where \( (K_i-1) \) denotes \( K_i \) with one variable removed. The \( w \) resulting parts which may become components, of the simpler model are of ordinality one less than the original. Save for the possibility of redundancies, by this rule, components with three or more variables yield loops. The variables of components with two variables become separated, and a one-variable component is deleted from the simpler model. Naturally, there is only one way of applying (1) to a component.

For four variables, \( A, B, C, D \), the algorithm may start with \( A \cdot B \cdot C \cdot D \) from which it generates one immediate simplification with four components:

\[ A \cdot B \cdot C \cdot D \rightarrow A \cdot B \cdot C \cdot D \]

The second iteration yields four alternative simplifications with three components each (the cancelled parts being redundant):

\[ A \cdot B \cdot C \cdot D \rightarrow A \cdot B \cdot C \cdot D \rightarrow A \cdot B \cdot C \cdot D \]

The third iteration yields three alternative simplifications from each of the above. e.g.:

\[ A \cdot B \cdot C \cdot D \rightarrow A \cdot B \cdot C \cdot D \]

and six in total, etc. The process of stepwise simplification terminates when all variables have been removed in which case \( m = \o \). Part of Figure 2 shows the resulting lattice in block diagrams that ignore mere permutations of the four variables.

TRANSFORMATION RULE FOR MODELS WITH THE SAME COVER

Models with the same cover differ from the general and unrestricted case merely in that their cover is unvarying and fixed. The transformation rule generating this kind of models must therefore not delete variables from the simpler models. It is a restricted version of (1):

\[ \begin{align*}
\{=1: & K_i \rightarrow K_i' \\
\rightarrow: & K_i \rightarrow (K_i-1') \cdot (K_i-1'') \cdots \cdot (K_i-1''^w) \}
\end{align*} \]

The resulting lattice of models in block diagrammatic form may be compared with the general case in Figure 2.

We note that this algorithm has also been used by Conant (1980). It generates what Cavallo and Kliir (1979) call G-structures which are seen in Figure 2 to be a special case of our set of all structural models. The explicit reliance on transformation rules makes our procedure not merely different from the one used by Kliir and Uyttenhove (1976) but, as we think, also much simpler both conceptually and computationally.

TRANSFORMATION RULE FOR MODELS WITHOUT LOOPS

To generate loopless or recursive models (regardless of cover) we state the transformation rule and its condition of application and leave the justification for the later. The transformation rule is:

\[ \{\begin{align*}
\rightarrow=1: & K_i \rightarrow \o \\
\rightarrow>1: & K_i \rightarrow (K_i-1') \\
\rightarrow: & K_j \in \o \rightarrow 1'' \}
\end{align*} \]

where \( 1' \) and \( 1'' \) are two different variables of \( K_i \) which must not be shared with the same component \( K_j \).
Figure 2: Lattices of block diagrams of four kinds of models in four variables.
If $K_i$ has only one variable, it is removed from the simplification of the model.

With $w$ as the number of variables in $K_i$, $u$ as the number of $K_i$'s unique variables and $m = w - u$ as the number of variables $K_i$ shares with other components. If $w = 1$ there is only one way of applying (3). Otherwise, if $1'$ and $1''$ are both unique to $K_i$ there are $u(u-1)/2$ ways of applying (3). If one of the two variables is unique and the other is shared there are $u$ ways of applying (3). And if both of these variables are shared with other components, there are as many ways of applying (3) as there are ways of selecting two shared variables from $K_i$ that are not shared with the same component. Examples of applying (3) are:

1' and 1'' unique: \[ ABCDEF \rightarrow ABCDE : BCDEF \]

1' unique, and 1'' shared: \[ AB : BCDE, ABCD : CDEF \rightarrow AB : BCDE, ABCD : CDEF \]

1' and 1'' shared: \[ ABCD : CDEF : FG \rightarrow ABCD : CDEF : FG \]

$K_i = 1$: \[ ABCD : DE : F \rightarrow ABCD : DE : \emptyset \]

Transformation Rule for Models Without Loops and With the Same Cover

Again, if the cover of the models of interest must be kept the same for whatever reason, no variables must be deleted from the model to be simplified:

\[ w \begin{align*}
1 & \rightarrow K_i \rightarrow K_i' \\
1' & \rightarrow K_i \rightarrow (K_i - 1); (K_i - 1') \\
\forall j \neq 1' & : 1'' \rightarrow K_j \text{ implies } 1'' \rightarrow K_j
\end{align*} \]

Figure 2 facilitates comparisons of the result of (4) with that of others.

Justifications

To yield minimal simplifications or the immediate successors of a model, it is required that the transformation rules break a component into as many parts as possible all of whose ordinalities are one less than the component they replace in the simpler model. Loops come into existence when any three or more such parts are brought together. To prevent loops from being generated it is thus necessary though not sufficient that the transformation rules for loopless models yield no more than two parts of ordinality one less than the original component. The transformation rules (3) and (4) do just this.

We explore briefly the conditions under which loops do emerge when one component is replaced by two such parts and how our rules circumvent the emergence of such loops. Loops emerge (a) when variables that are shared by two or more components become separated after breaking one component into parts, e.g.:

\[ \text{ABC:BCD} \rightarrow \text{AB:AC:BCD} \]

and (b) when the variables that a set of components have in common (which can therefore have no influence on the circularity within this set) becomes shared by a smaller subset of these components so that the dependency thereby established closes the circle involving other previously loopless dependencies, e.g.:

\[ \text{ABC:BCD:DE} \rightarrow \text{ABC:BCD:DE} \]

When the variables 1' and 1'' of (3) and (4) are either both unique or one is unique and the other is shared, the two rules are unable to generate such loops because the set of shared variables in $K_i$ always remain whole and undivided in at least one of the resulting parts. The loops in examples (a) and (b) exclusively arise when both 1' and 1'' are shared with other components in which case (3) and (4) are severely restricted in application. Considering (a), since the component ABC shares both B and C with the only other shared component BCD, the rules are unable to transform ABC into
AB:AC. Similarly in (b), the rules cannot separate the two shared variables B and C in the transforms of ABC and BCD as well as C and D in the transforms of BCD and CDE. The only simplification of the loopless model in (b) (3) and (4) permits is by transforming BCD with B=1' and D=1". ABC:BCD:CDE→ABC:BC:CD:CDE which eliminates BCD in effect, leaving a model without loops behind.

From the foregoing argument it seems clear that the transformation rules (3) and (4) are unable to generate models with loops from those that contain none. Less clear is whether these rules generate all possible loopless models. The formal proof that the proposed transformation rules generate all and only loopless models is still missing. We can only report that we have applied these transformation rules to several sets of variables and found them to be working flawlessly.

REFERENCES


AUTHOR Klaus Krippendorff
The Annenberg School of Communications
University of Pennsylvania, C5
Philadelphia PA 19104

Klaus Krippendorff is professor at the University of Pennsylvania's Annenberg School of Communication. He graduated in 1961 from the Ulm School of Design and received his Ph.D. in (social) communication in 1967 from the University of Illinois, Urbana. He has been involved in and consulted with several social research projects, teaches subjects in human communication, in the methodology of social research, especially content analysis, and in cybernetics and systems theory. He is also affiliated with the Social Sciences department at the University of Pennsylvania. He just returned from a year as fellow of the Netherlands Institute of Advanced Studies (NIAS)

Klaus Krippendorff's main interest has been to apply cybernetic models to solve social problems. He edited a book on Communication and Control in Society, coedited one and wrote another book on content analysis, contributed book chapters on clustering techniques, on validity, on systems approaches to communication and numerous journal articles. One of his continuing interests has been the extension of information theory to a multivariate calculus and the development of associated analytical devices for discovering patterns in qualitative data.
PROOF FOR AND ILLUSTRATION OF AN ALGORITHM THAT DISTINGUISHES STRUCTURAL MODELS
WITH LOOPS FROM THOSE WITHOUT LOOPS

At the 1979 Delft Summer School on Systems Methodology we presented
an algorithm that distinguishes structural models that do contain loops
from those that don't, the motivation being that maximum likelihood
estimates for models without loops can be obtained algebraically, whereas
similar estimates for models with loops require iterative procedures.
We were then unfamiliar with a procedure published by Bishop, Fienberg
and Holland. This ignorance proved to be an advantage because our algor-
ithm turned out to be somewhat simpler than their's, leading to the same
decisions, however.

The algorithm is stated as follows:

Given a set of components $K_1, K_2, \ldots, K_i, \ldots$ of a model $m$,

1. remove all variables that are unique to any $K_i$,

2. remove any $K_i$ that is equal to or contained in any other $K_j$ of the
Repeat (1) and (2) until either (a) no variables remain in which case
loops are absent, or

(b) the remainder is unalterable by the
above in which case loops exist.
Two examples illustrate how the algorithm works. First example:

\[
\text{given: } \quad \text{ABC:BDE:BCEFG:EFH:HI}
\]

by step 1:
\[
\text{BC:BE:BCEF:EFH:H}
\]

by step 2:
\[
\text{BCEF:EFH}
\]

by step 1:
\[
\text{EF:EF}
\]

by step 2:
\[
\text{EF}
\]

by step 1:
\[
\emptyset \implies \text{there is no loop}
\]

Second example:

\[
\text{given: } \quad \text{ABC:BDE:BCEFG:EH:FHI:IJ}
\]

by step 1:
\[
\text{BC:BE:BCEF:EH:FHI:I}
\]

by step 2:
\[
\text{BCEF:EH:FHI}
\]

by step 1:
\[
\text{EF:EH:FH} \implies \text{there is a loop}
\]

Here we intend to prove the contention that the procedure makes the decision as claimed. For notations and background we refer to the author's paper.²

Generally, any system \( S=S_1:S_2:...:S_n:... \) may be expanded by adding a new component \( K \):

\[
S' = S:K
\]  

(1)
This expansion creates loops whenever $K$ connects two or more already connected components of $S$, e.g.:

![Diagram showing interconnected components of $S$.]

The expansion will not create loops when either $K$ connects only independent components of $S$, e.g.:

![Diagram showing independent components of $S$.]

or $K$ does not establish a new connection between different components in $S$, e.g.:

![Diagram with no new connection between components.]

In an expansion of a system that does not create loops, the variables in $K$ must be of not more than two kinds, unique and shared with one component $S_1$ of $S$. Let $(S\&K)$ be unique to $K$, let $(S_1\&K)$ be shared by $S_1$ and $K$ and let $(S\&K)(S_1\&K)$ be the cartesian product of the two. Furthermore, let $p_s$ be the probability of a state $s\in S$, let $p_k$ be the probability of $k\in K$, let $p_{s\&k}$ be the probability of values $s\&k\in S\&K$ shared by $S$ and $K$. We call the extension $S'=S:K$ a recursive expansion iff:

$$
\exists i: (S\&K)(S_1\&K) = K \implies p_{s'} = \frac{p_s p_k}{p_{s\&k}} = p_s p_{s\&k|s\&k}
$$

(2)

$p_{s\&k|s\&k}$ is the conditional probability of the unique values in $K$ given the values shared by $S$ and $K$ and $p_{s'}$ is the probability of the extended system.
Three special cases of (2) will illustrate the nature of the recursive expansion. First, K and S are independent, \( S \& K = \emptyset \), \( p_{S \& K} = 1 \):

\[
S \& K = K \quad \implies \quad p_{s'} = \frac{p_s p_k}{p_{S \& K}} = p_s p_k \quad (3)
\]

Second, \( S_1 \) is contained in K, \( S \& K = S_1 \), \( p_{S \& K} = p_{S_1} \):

\[
(S \& K) S_1 = K \quad \implies \quad p_{s'} = \frac{p_s p_k}{p_{S \& K}} = p_s p_{S_1} \quad (4)
\]

Showing K to replace \( S_1 \) or to merely expand \( S_1 \)'s variables. Third, K is contained in \( S_1 \), \( S \& K = K \), \( p_{S \& K} = p_k \):

\[
S_1 \& K = K \quad \implies \quad p_{s'} = \frac{p_s p_k}{p_{S \& K}} = p_s \quad (5)
\]

showing K to be entirely redundant. This fact motivates the exclusion of redundant parameters as meaningful specifications of our notion of a structural model and justifies step 2 of our algorithm.

Since (2) prevents a new component from establishing new connections between the existing components, the recursive expansion yields systems with chain-like dependencies but cannot create loops. Thus, if S is free of loops so is its recursive extension \( S' = S:K \). One test for whether a given system, say, \( S^m \) is free of loops requires showing that it can be generated in a sequence of recursive expansions:

\[
\emptyset \rightarrow S' \rightarrow S'' \rightarrow \ldots \rightarrow S^m \quad (6)
\]

from any system known not to have loops. The most primitive system without loops and the most obvious starting point of the sequence of recursive expansions is the system that has no variables whatever and is denoted here by \( \emptyset \).
Our algorithm computes that sequence (6) by recursively removing those components and those variables of components that could have been added by recursive expansion. Specifically, in step 1 of our procedure, all unique variables are removed from any one of the system's components, including all independent components, according to (3), and mere expansions of some component according to (4). Then, in step 2, all redundant components are removed which is indicated by (5) and incorporated in our definition of a structural model. If \( S^m \) is recursively reducible to \( \emptyset \), then \( S^m \) is also recursively obtainable from \( \emptyset \) and loops do not exist.

However, the efforts to recursively reduce a system to \( \emptyset \) may fail and it will precisely when there are no variables unique to any one component that could be removed in step 1 of the algorithm and whenever there are no redundant components in the set (none is contained in another) that could be removed in Step 2. We claim this condition to indicate the presence of a loop.

In a "pure" loop all variables must link different (that is two or more) components, e.g.:

![Diagram of loops](image)

It follows that none of the variables involved in a loop is unique to any one component and none is shared by all components. Since (2) prevents the establishment of new links between components and step 1 of the algorithm can not reduce a system whose components have no unique variables,
loops can neither be generated by a recursive extension nor be broken down or into by the algorithm. Any recursive reduction of a system that ends up with something other than $\emptyset$ contains loops.

The decision procedure will remove all recursive expansions. E.g.,

the systems:

are reduced to systems with the above "pure" loops. The fact that the algorithm does not remove variables that are shared by all components forming a loop, e.g.:

is of no consequence for a decision on whether or not a loop exists.

Q.E.D.

Klaus Krippendorff, University of Pennsylvania, Philadelphia October 25, 1980
References
