2004

A Software Architecture for Reconstructability Analysis

Kenneth Willett
Portland State University

Martin Zwick
Portland State University, zwick@pdx.edu

Citation Details
Kenneth Willett, Martin Zwick, (2004) "A software architecture for reconstructability analysis", Kybernetes, Vol. 33, No. 5/6, pp. 997 - 1008

A SOFTWARE ARCHITECTURE FOR RECONSTRUCTABILITY ANALYSIS

Kenneth Willett and Martin Zwick
Systems Science Ph.D. Program, Portland State University, P.O. Box 751, Portland, OR 97201
kwillett@ignisys.com

KEYWORDS: reconstructability analysis, log-linear modeling, information theory, data-mining, Occam, Python

ABSTRACT

Software packages for Reconstructability Analysis (RA), as well as for related Log Linear modeling, generally provide a fixed set of functions. Such packages are suitable for end-users applying RA in various domains, but do not provide a platform for research into the RA methods themselves.

A new software system, *Occam3*, is being developed which is intended to address three goals which often conflict with one another: to provide (1) a general and flexible infrastructure for experimentation with RA methods and algorithms; (2) an easily-configured system allowing methods to be combined in novel ways, without requiring deep software expertise; and (3) a system which can be easily utilized by domain researchers who are not computer specialists.

Meeting these goals has led to an architecture which strictly separates functions into three layers: the Core, which provides representation of datasets, relations, and models; the Management Layer, which provides extensible objects for development of new algorithms; and the Script Layer, which allows the other facilities to be combined in novel ways to address a particular domain analysis problem.

I. INTRODUCTION

Reconstructability Analysis (RA) is a technique based on set theory and information theory for the modeling and analysis of datasets involving discrete variables. It derives from Ashby (1964), and was developed by Broekstra, Cavallo, Cellier, Conant, Jones, Klir, Krippendorff, and others (see, e.g., Klir 1985 1986 1996 2000; Krippendorff 1986; Zwick 2001a) It differs from other techniques in that it can identify high-dimensional multi-component relationships among the variables.

RA encompasses a number of related methods and variations:

- *Information-Theoretic(probabilistic) vs. Set-Theoretic (crisp possibilistic) modeling*. The information-theoretic approach deals with a frequency or probability distribution over the states of the system, while the set-theoretic approach deals only with the occurrence or non-occurrence of each possible state. These two approaches can be seen as two distinct methods within the framework of Generalized Information Theory (Klir 98),which also includes fuzzy methods (currently outside the scope of the software system described in this paper). Information-theoretic RA is mathematically equivalent to Log-linear Modeling (Bishop et al 1978; Knoke and Burke. 1980), where the two overlap. Set-theoretic RA is related to techniques used in logic design and machine learning (Perkowski et al 1997; Files and Perkowski 1998).
- *Variable-Based*, *Latent Variable-Based* and *State-Based* modeling. Traditionally RA has used a variable-based approach, where the constraints of the model are associated with sets of variables and the corresponding state subspaces. Latent variable-based modeling (Hagenaars 1993) takes a similar approach, but introduces new (unobserved) variables to represent relationships among the primary variables. State-based modeling (Zwick and Johnson, 2002) defines the constraints of the model (that is, the specifics of how the model must match the observed data) in terms of specific states of the data or its margins, rather than being defined by complete margins only. Thus, state-based modeling is a generalization of variable-based modeling.
- *Directed vs Neutral Systems*. In a directed system one or more of the variables is assumed to be dependent on the others, and the problem is typically to find a reduced set of variables which adequately predicts the dependent variable(s). In neutral systems all variables are treated as interdependent, and the problem is to identify the relationships among the variables which most account for the relation or joint probability distribution.

There exist several RA software packages, e.g., GSPS by Klir (1976), Elias (1988), and coworkers, CONSTRUCT and SPECTRAL by Krippendorff (1981), SAPS by Uyttenhove (1984) and Cellier (1987), EDA by Conant (1988), Jones' k-systems analysis (Jones, 1989) and a recent program by Dobransky and Wierman (1995). However, no package fully encompasses the variations of RA discussed above. Some programs are not easily used by researchers outside the systems field; others do not incorporate statistical tests; only EDA can handle large numbers of variables. These packages are in limited use. A number of statistics packages (SAS, SPSS, Statistica,

Willett, K. & Zwick, M. 2004. "A Software Architecture For Reconstructability Analysis." *Kybernetes*, vol. 33, No. 5/6, pp. 997-1008.

LEM) provide a basic Log-Linear (LL) capability which can be used to perform RA in a confirmatory mode, where a model can be selected and its fit to the data evaluated. However, the set of possible models for a dataset of N variables grows hyper-exponentially with N. Because of this, and because RA is most useful for modeling complex (and therefore non-intuitive) relationships, a simple confirmatory approach has limited value. Instead, one requires the ability to rapidly search the space of all possible models, applying heuristics to narrow the search.

A previously developed software package, *Occam* (Zwick 2000), was developed at Portland State University to provide a range or RA capabilities. This package was derived from a set of predecessor programs, the first of which was written by Zwick in 1985. Under his direction, programs (Occam0) were written by Hosseini, Anderson, and Shu which improved these computations and performed additional RA functions, and several of these programs were then integrated by Daniels, who also introduced heuristic search (Occam1). Grygiel added new search procedures and an improved user interface (Occam2). This most recent version has been used to analyze medical, healthcare, satellite, linguistic, and other data, as part of a general research program in discrete multivariate modeling (Zwick, 2001b). A desire to increase the accessibility of RA methods to a broader range of researchers, and also make these methods both more flexible and more automated, has led to development of *Occam3*.

II. KEY REQUIREMENTS FOR OCCAM3

Ideally, a tool such as *Occam* should address the needs for three types of researchers in the RA community:

1. Those developing variations on RA methods, as mentioned above. These researchers require the ability to *extend the representation* of core RA entities, such as data tables and model definitions.
2. Those developing new algorithms within a given RA framework, such as novel search heuristics, different statistical measures, etc. These researchers need to be able to extend the *processing rules* used by the system.
3. Those using RA for analysis in various domains. These researchers need a *flexible analysis framework*, which allows them to combine various methods according to the needs of their particular problems.

Previous versions of *Occam* addressed item number 3 only; the system architecture was rigid and provided no simple means of extension. For this, a different architecture was needed.

III. OVERVIEW OF RA

Both information-theoretic and set-theoretic analyses begin with a table of sampled data, over a set of discrete variables (continuous variables can be discretized by *binning*.). The *cover* is the set of variables in the data, and the data defines a distribution (information-theoretic) or a set-theoretic relation over the system states defined by the Cartesian product of all the variables.

Given this data, one is interested in evaluating simpler approximations of the data. In RA such an approximation is defined by a *structure*, which is a set of *relations*. (From now on, the the word *relation* will encompasses both distributions and set-theoretic relations.) Each relation is in turn a set of variables from the cover. For a four variable problem with a cover{A,B,C,D}, an example of a relation is ABC and of a structure is ABC:BCD.

Given a structure, a *model* is constructed by first producing a *projection* for each relation in the structure, and then *reconstructing* a fitted relation over the cover, which agrees with all the projections. For both set-theoretic and information-theoretic RA, one chooses the relation which maximizes uncertainty subject to the constraints of the projections.

A structure may be *loopless* or may contain loops. For example, AB:BC:BD is loopless, while AB:BC:ACD contains one loop (i.e., A→B→C→A). Loopless models are more efficient to process because fitted distributions can be determined algebraically, while distributions for models with loops must be fitted iteratively. (For set-theoretic relations, however, even models with loops have closed form solutions).

Structures (and their corresponding models) can be arranged in a lattice by definition of a *parent-child relationship*. Given a structure, each child structure is created by removal of a relation and reinsertion of all the embedded relations within that relation which are not already present in other relations of the structure. For example, the relation ABC has embedded relations AB, BC, and AC. Thus the children of ABC:BD are AB:AC:BC:BD and ABC:D.

The topmost structure (model) in the lattice is the *saturated structure (model)*, defined by all the variables occurring in a single relation, e.g. ABCD. The bottom of the lattice is defined by the *independence structure (model)*, which is defined by only first-order relations, e.g. A:B:C:D. To be more precise: this is the bottom of the lattice if one wishes to maintain the same cover in all structures, so that every structure contains every variable.

A researcher using RA is typically interested in one or more of the following questions:

1. What are all possible structures for a given cover? What are all the loopless structures, structures of non-overlapping relations, etc.? This might be a prelude to further exploration.
2. Which subset of variables in a directed system is most predictive of the dependent variable(s)? This problem is equivalent to bottom-up search of the sublattice of loopless models.
3. Which model(s), allowing loops, provide the simplest representation of the data, while still fitting the data with adequate accuracy? This question involves *searching* the lattice of models.
4. What are the characteristics of each of a specific set of models? The models may have been identified by some *a priori* criteria. Models can be *fit* to the data and then compared based on predictive power, differences from the data, chi-squared statistics, etc. Details of the model, such as the residual error in each specific state of the system, can also be computed.

IV. OVERVIEW OF OCCAM3 ARCHITECTURE

The three requirements described above are addressed by separation of the architecture into three distinct layers. Each of these layers is extensible to address the needs of a particular constituency.

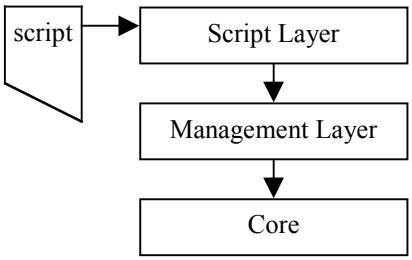
The Occam Core provides the representation of basic RA entities. The implementation of the core must be done with careful attention to performance, memory usage, and robustness. Implementation of this layer is done in C++.

The Occam Management Layer provides the manipulation of Core entities. This layer provides the basic mechanisms for searching, caching, computing statistics of interest, and reporting. This layer is implemented in C++ and uses class inheritance to provide extensibility.

Assembly of particular functions into an analysis strategy is done using the *Occam Script Layer*. There are standard scripts for common situations (such as detailed analysis of a single model, or simple search heuristics), but it is possible to create more complex scripts for special cases. The scripts are written in the *Python* language (Beazley 99), and an adapter interface between the Python interpreter and the rest of *Occam3* is written in C++.

Python was chosen because it is readily available on a wide variety of systems, including Windows and various forms of Unix. It is powerful enough to develop complex algorithms, but does not require mastery of programming details such as variable declarations and memory management. Even though Python is an interpreted language, most of the computation is done at lower levels of the system, so interpreter performance is not an issue.

The specific approaches used for these three layers are described further in the sections below.

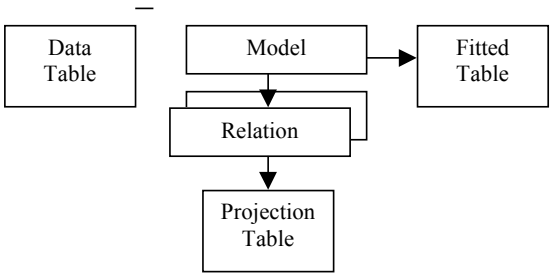


Architecture of Occam3

V. THE OCCAM CORE

Occam Core Objects

The Occam Core provides efficient implementations of three key objects: *Tables*, *Relations*, and *Models*. These objects are described further below.



Objects in the Occam Core

Tables

A key goal of *Occam3* is to be able to handle larger problems, with larger numbers of variables, than was possible with *Occam2*. (*Occam2* has been used to analyse data involving 10's of variables.) The state space of a particular problem grows exponentially with the number of variables, while the amount of available data for analysis is likely to grow at a much slower rate. This means that, for large problems, the data is very sparse (not every feasible state is represented in the data sample).

The Core takes advantage of this sparseness by avoiding the need to represent the entire state space explicitly. A Table, whether it represents the input data or a computed projection, consists of a sorted list of tuples. Each tuple contains a *key*, which encodes the states of the individual variables associated with that tuple; and a *value* which may be either Boolean (for set-theoretic analysis), a frequency, or a probability (depending on the type of analysis being performed).

Tables are used in three contexts. A *Data Table* contains observation data, typically read from a data file. Data tables might also be produced by preprocessing data in some way, such as by binning quantitative variables, performing mask analysis on time series data, etc.

A *Projection Table* is computed from a Data Table, and is associated with a particular Relation by summing over the missing variables for probabilistic distributions or taking a logical "or" for crisp possibilistic relations.

A *Fitted Table* is a computed table associated with a Model, which defines a maximum uncertainty relation subject to the constraints of the model for all variables in the cover. Fitted tables for distributions are produced by algorithms such as Iterative Proportional Fitting (IPF) (Krippendorf 19xx). Research into other, more efficient methods for fitting is being done as part of the overall Occam project (Zwick 2002).

For large analysis problems, the memory space requirements become dominated by storage of these data tables. Typically the cardinality of individual variables is small (binary variables are common; most variables have only a few states); so bit-packing of the key is used to represent the tuple in the smallest possible number of bytes.

Sorting the tuples allows binary searching to be used to find specific states, and allows operations on multiple tables to be performed in linear time. Sorting eliminates the need for any additional indexing structure for the data tables.

Each variable has an additional state, "absent", which is used in computing projection tables. For example, in a problem with binary variables A, B, and C, the tuples of the projection associated with AB are 00*, 01*, 10*, and 11*, where "*" represents the "absent".

Computation of a Projection Table over one or more variables is done in time which is linear with the amount of data, and storage of this table is typically much smaller than the initial data set due to combination of tuples.

Relations

A Relation stores a list of variables defining the relation, an associated projection table produced from the initial data table, and a list of *Attributes*, computed from the relation (e.g., Degrees of Freedom, Uncertainty, etc.)

Relations are reusable, because the projection table and relation attributes are independent of the model currently being evaluated. Also, the storage space for a relation, and the high computational cost of computing the projection for that relation, makes this reuse a critical performance strategy. A cache of Relations is maintained so that each relation is computed only once.

Models

An RA model is defined be a set of relations; thus each Model object contains references to its constituent relations, but not a separate copy. A Model also contains a list of Attributes, with statistics of interest computed for the model. A Model may also have a Fitted Table in cases where this must be explicitly computed (see below).

VI. THE OCCAM MANAGEMENT LAYER

While the Occam Core is concerned primarily with the representation of various data objects, analysis methods are implemented in the Occam Management Layer. Finding the best models, and computing and displaying statistics about models, involves three types of operations:

- Navigation – this involves traversal of the lattice of all possible models, producing new candidate models from existing ones.
- Reconstruction – for probabilistic systems, this involves construction of a fitted distribution for a given model. Reconstruction requires generation of a projection for each relation in the model, then the combination of those projections to produce the fitted distribution. For models without loops, the fitted distribution can be represented explicitly in terms of the projections of the individual relations, and key attributes such as uncertainty and degrees of freedom are computed algebraically. For models with loops it is necessary to construct a Fitted Table, using an algorithm such as Iterative Proportional Fitting (IPF). Reconstruction for crisp possibilistic systems is done using a closed form set-theoretic equation for models both without and with loops.
- Evaluation – attributes of the model are computed and used to rank models for further processing. Attributes may be computed from the model structure (such as Degrees of Freedom), or may be computed from the fitted distribution (e.g., Uncertainty, Transmission).

The Management Layer uses the entities of the Core (Tables, Relations, and Models) and basic operations on these entities (constructing projections, computing uncertainties, etc.) to build these higher-level RA methods.

The foundation for building extensions in this layer (such as different heuristic search methods, of different evaluation criteria) is the *Base Manager*. This object provides facilities for caching and reusing relations and models.

The Base Manager also provides for basic operations such as construction of a fitted table for a model; computation of degrees of freedom (DF) and uncertainty (H) for models with overlapping components; and generation of all child relations of a given relation. These basic operations are used in a variety of RA methods.

More specialized managers can be constructed using *class inheritance* in C++. Such a manager will augment and/or replace operations provided by the Base Manager. In

particular, the *Variable Based Modeling (VBM) Manager* has been developed to provide support for classical RA methods.

In the lattice of models the basic operations are to move upward (find all parents of a given model); or move downward (find all children).

It is also useful to be able to navigate within the sublattice defined by models meeting a certain criterion, such as those without loops. Restricting attention to loopless models allows much larger problems to be considered. For directed systems, loopless models correspond to models with a single predictive component. For example, the model ABC:AZ is loopless, while model ABC:AZ:BZ which has two predictive components contains a loop (A→B→Z→A)

Most model evaluation methods are based on some measure that combines the complexity of the model and the quality of its fit to the data. In the lattice of models, quality of fit is highest at the *saturated model* at the top of the lattice, which exactly fits the data. Quality of fit decreases monotonically as one navigates down through the lattice, with the fit being poorest with the *independence model* which assumes all variables are independent. (This is true among models which use the full *cover*, i.e. the set of variables in the data. It is possible to analyze even simpler models, where variables are removed from the model entirely, ending with the uniform distribution model.) Quality of fit is typically measured by either the transmission between the model and the saturated model, or by the reduction in uncertainty of the dependent variables in the model relative to their uncertainty in the independence model.

Model complexity, on the other hand, is lowest in the independence model, and increases monotonically upward. Information-theoretic complexity is typically measured by DF; similar measures are available for set-theoretic models (Grygiel 2000).

Because the set of possible navigation operators is open-ended, the VBM Manager has a facility for navigation "plug-ins". At this point plug-ins have been developed for full upward and downward search, as well as loopless upward and downward search (loopless upward search is currently limited to directed systems).

The VBM Manager has algorithms defined for about 40 relation and model attributes (Uncertainty, Transmission, standard and Pearson chi-square statistics, etc.). Additional attributes can be defined either as extensions to the Management Layer, or in the Scripting Layer.

The result of the search process is typically a report of the best models encountered during search, sorted by some criterion (typically the same criterion which guided the search). Depending on the analysis problem, different statistics of the models may be of interest. The Management Layer contains a report generator, which can be provided with a list of models, sort them according to a specified attribute, and display the results as a table. Tables can be generated in a textual format, a format compatible with desktop applications (e.g., spreadsheets), or in HTML.

VII. THE SCRIPT LAYER

One of the most serious limitations of previous versions of *Occam* was that minor variations in analysis technique required changes to the underlying software. The researcher was restricted to a fixed set of options, and adding additional options not only required rebuilding the software but also led to a proliferation of options and controls.

In the planning for *Occam3*, it was clear that more flexibility and programming power in the user interface was required. Users who want to run analysis of a number of data sets, or preprocess the data in multiple ways before analysis, were interested in a way to automated this processing. For this reason, instead of being developed as a stand-alone application, *Occam3* was developed as a loadable module which can be used in conjunction with the script language *Python*.

The Python language was initially developed by Guido van Rossum at BWI in Amsterdam in 1990, and its continued development is performed by an Open Source development community. Python is becoming common in information system management applications as well as in web systems.

Python was selected because of its simple syntax, its high level functions (such as automatic memory management with garbage collection, list processing, etc.) and the ease with which it can be used with existing C and C++ libraries.

Interfacing a C++ library to Python involves development of an adapter file, which describes how C++ facilities are made visible through Python. Currently the *Occam* adapter exposes four classes through python:

- ocVBMManager, which provides access to most of the computational and modeling functions, such as constructing models, performing search navigation operations, and computing statistics.
- ocModel, which provides read access to details of a model, such as its attributes and relations.
- ocRelation, which provides read access to the attributes and data of a single relation.
- ocReport, which provides model sorting and reporting functions.

The example below shows a simple script for performing a complete loopless top-down search, and producing a report sorted on %dH.

```
import os, sys, occam
sortName = "cond_pct_dh"          # this determines the sort order for filtering
mgr = occam.ocVBMManager()        # create a variable based modeling manager
mgr.initFromCommandLine(sys.argv) # initialize setting and read input file
mgr.setSearchType("loopless-down")# set search method

report = occam.ocReport()         # create a report generator
report.setAttributes("level, h, ddf, lr, alpha_sat, cond_pct_dh");
report.setSeparator(3);           # align columns using spaces

top = mgr.getTopRefModel()        # get the top (saturated) model

# Build a work list; go through all the models and construct their children.
# Mark models already processed so they aren't repeated. Add each model to
# report generator

modelList = [top]
while len(modelList) > 0 :
       model = modelList[0]
       modelList = modelList[1:]
       mgr.computeL2Statistics(model)
       newModels = mgr.searchOneLevel(model)
       for newModel in newModels :
              if newModel.get("processed") <= 0.0 :
                     newModel.processed = 1.0
                     modelList.append(newModel)
                     report.addModel(newModel)

report.sort(sortName, sortDir)    # sort and print report
report.writeReport("report.out")
```

VIII. USING OCCAM3 FOR LARGE DATASETS

A simple experiment was run using a subset of the OPUS data obtained from Dr. Clyde Pope of the Kaiser Permanente Center for Health Research in Portland, Oregon (Zwick and Pope, 2002). This data concerns healthcare utilization in the Kaiser member population, and is a directed system with 24 independent and 1 dependent variable. This data has previously been analyzed with *Occam2*, to (1) select a subset of the 24 variables, which is most predictive, and then (2) build a model structure from that subset of variables which is statistically significant.

With *Occam2* the first step requires exhaustive bottom-up search. Since the number of single-component models grows exponentially with distance from the bottom of the lattice, the search for the best seven-variable model required many hours.

The second step was able to take advantage of *Occam2*'s fixed width downward search, and was much faster. Even so, running more than a few experiments (e.g., changing the number of variables in the selected subset) was very expensive, requiring several hours. Also, the sequence of performing step 1, then step 2, required manual interaction.

With *Occam3* a simple script was written which defined the desired search strategy: a fixed-width single-predicting-component (i.e., loopless) upward search for the desired number of levels, followed by fixed-width downward search. All models not statistically significant were eliminated before the final report.

The upward search to select the best 7-variable component was faster by more than an order of magnitude, with a search-width of 30 (i.e., the best 30 2-variable components were kept, then the 30 best 3-variable components, etc.) This heuristic approach identified a number of single-component models as good or better than the component found with *Occam2*.

The scripting approach makes it simple to combine both the upward and downward search phases, and also allows experimentation with

search width, number of levels, and different sorting criteria.

IX. BIBLIOGRAPHY

Ashby, W. R. 1964. "Constraint Analysis of Many-Dimensional Relations." *General Systems Yearbook*, vol. 9, 99-105.

Bishop, Y. M.; S.E. Feinberg; and P.W. Holland. 1978. *Discrete Multivariate Analysis*. MIT Press, Cambridge.

Beazley, D. 1999. *Python Essential Reference*, New Riders Publishing.

Cellier, F. and D. Yandell. 1987. "SAPS-II: A New Implementation of the Systems Approach Problem Solver." *Int. J. of General Systems*, vol. 13 (4), pp. 307-322.

Conant, R. C. 1988. Extended Dependency Analysis of Large Systems. *Int. J. General Systems*, vol. 14, 97-141.

Dobransky, M. and M. Wierman. 1995. "Genetic Algorithms: A Search Technique Applied to Behavior Analysis." *Int. J. of General Systems*, 24 (1&2), pp. 125-136.

Elias, D. 1988. *The General Systems Problem Solver: A Framework for Integrating Systems Methodologies*. Ph.D. Dissertation, Dept. of Systems Science, SUNY-Binghamton.

Files, C. and M. Perkowski, M. 1998. "Multi-Valued Functional Decomposition as a Machine Learning Method,"' *Proc. ISMVL'98*.

Grygiel, S. 2000. *Decomposition of Relations as a New Approach to Constructive Induction in Machine Learning & Data Mining*. Electrical Engineering Ph.D. Dissertation, PSU, Portland OR.

Hagenaars, J. A. 1993. *Loglinear Models With Latent Variables*. (Quantitative Applications in the Social Sciences #94). Sage, Beverly Hills.

Jones, B. 1989. "A Program for Reconstructability Analysis." *Int. J. General Systems*, 15, pp. 199-205.

Klir, G. 1976. "Identification of Generative Structures in Empirical Data." *Int. J. General Systems*, vol. 3 (2), pp. 89-104.

Klir, G. 1985. *The Architecture of Systems* Problem Solving. Plenum Press, New York.

Klir, G. 1986. "Reconstructability Analysis: An Offspring of Ashby's Constraint Theory." *Systems Research*, vol. 3 (4), pp. 267-271.

Klir, G., ed. 1996. *International Journal of General Systems* Special Issue on GSPS, vol. 24.

Klir, G. and M. Wierman. 1998. *Uncertainty-based Information – Elements of Generalized Information Theory*, Physica-Verlag, Heidelberg

Klir, G., ed. 2000. *International Journal of General Systems* Special Issue on Reconstructability Analysis in China, vol. 29.

Knoke, D. and P.J. Burke. 1980. *Log-Linear Models*. (Quantitative Applications in the Social Sciences Monograph # 20). Sage, Beverly Hills.

Krippendorff, K. 1981. "An Algorithm for Identifying Structural Models of Multivariate Data." *Int. J. of General Systems*, vol. 7 (1), pp. 63-79.

Krippendorf, K. 1986. *Information Theory - Structural Models for Quantitative Data*, Sage Series: Quantitative Applications in the Social Sciences, Series 07-062.

Perkowski, M., M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, and J. Zhang. "Decomposition of Many-Valued Relations," *Proc. ISMVL '97,* Halifax, Nova Scotia, May 1997, pp. 13-18.

Uyttenhove, Hugo J.J. 1984. "SAPS - A Software System for Inductive Modelling." In: *Simulation and Model-Based Methodologies: An Integrative View,* edited by T.I Oren et al. Springer-Verlag, Berlin, Heidelberg. NATO ASI Series, Vol. F10, pp.427-449.

Zwick, M. 2000. "OCCAM: Organizational Complexity Computation and Modeling", Portland State University Systems Science Program Internal Document.

Zwick, M. 2001a. "Wholes and Parts in General Systems Methodology.". In: *The Character Concept in Evolutionary Biology*, G. Wagner, ed. Academic Press, New York, pp. 237-256.

Zwick, M. 2001b. "Discrete Multivariate Modeling" http://www.sysc.pdx.edu/res_struct.html

Zwick, M. and M. Johnson. 2002. "State-Based Reconstructability Analysis." In these *Proceedings*.

Zwick, M. 2002. "Reconstructability Analysis With Fourier Transforms." In these *Proceedings*.