97-13

A Description and User's Guide for CLUSTER/2C++ A Program for Conjunctive Conceptual Clustering

P 97-13

MLI 97-10

May, 1997

Note: This document is a revision of "A Description and User's Guide for CLUSTER/2: A Program for Conjunctive Conceptual Clustering", by Robert Stepp.

A Description and User's Guide for CLUSTER/2C++: A Program for Conjunctive Conceptual Clustering

Scott Fischthal
Machine Learning and Inference Laboratory
George Mason University

Abstract

The following is a comprehensive user's guide and reference for using CLUSTER/2C++, the latest version of the CLUSTER/2 program for constructing classifications of arbitrary objects. Given a set of object descriptions, in the form of attribute-value pairs, the program structures the objects into a hierarchy of classes. The program produces a conjunctive description of each class using selected object attributes. The program runs in a batch mode, with parameter and object descriptions input via relational tables; future versions will provide interactive and embeddable interfaces. This document contains the definitions of the relational table syntax and the values that can be entered. Also included and discussed are sample input and corresponding program output generated by CLUSTER/2C++.

Acknowledgments

This research was conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's activities are supported in part by the Defense Advanced Research Projects Agency under Grant No. F49620-95-1-0462 administered by the Air Force Office of Scientific Research, in part by the National Science Foundation under Grants No. DMI-9496192 and IRI-9020266, and in part by the Office of Naval Research under Grant No. N00014-91-J-1351.

1 Introduction

CLUSTER/2 is a program for automatically constructing conceptual classifications. It is the successor to the program CLUSTER/PAF [5][6]. CLUSTER/2 is applicable to a wide variety of problem domains. CLUSTER/2 has been applied experimentally to such problems as classifying Spanish folksongs [7][9], classifying microcomputers [8], reconstructing soybean disease categories [8][12] and is currently being used to aid in fraud detection [2]. Input to CLUSTER/2 consists of a set of attribute vectors (one vector per input event), definitions of the types of variables and their value sets, and a specification of the clustering quality criteria. The program generates a classification hierarchy of the data events along with a conjunctive description of each cluster at each level of the hierarchy. The program forms clusters that have a conjunctive description and optimizes the given criteria of clustering quality. The program forms both clusters and their descriptions and thus behaves unlike programs that implements conventional numerical taxonomy techniques. The theoretical background for CLUSTER/2 can be found in [4], [8], and [12].

2 Input file content

CLUSTER/2C++ is currently run in batch mode with input from a single file. An interactive input method is currently in design. CLUSTER/2C++ generates an output file consisting of a report of the obtained clusters along with optional trace information showing alternative clusters considered and other pertinent statistics.

The user provides input data to the program via relational tables according to a standard syntax. Each table consists of three parts:

- A table identifier composed of one of the predefined table names
- A line of column headings composed of predefined column names
- Lines of data values arranged in the same order as the column headings

The header of the table can be arranged in any convenient order.

Data items ion one line of a table are separated by "white space" (spaces, commas, tabs, exclamation points). There are three types of data values: *integer, real* and *character.* Each type has a required format described below. In the table definitions that follow, the symbols `<i>, <r>, <c>` denote columns that take integer, real and character data respectively. These symbols are for descriptive purposes for the user's guide only and should not be entered into the input file by the user.

Some tables have default value which are used whenever a column is absent or when a row contains no value in a particular column. Since the relational table syntax uses no punctuation, a place holder symbol must be used whenever a value in a row is missing. '$' and '?' mean "not applicable" and "unknown", respectively.

2.1 Integer data values

Integer data consists of signed integer numbers such as 10 and -3. For very large integers, scientific notation may be used in the following form:

<sign><mantissa>E<exponent>

Examples of the scientific form are: 1E6 (which represents one million) and -2.3E7 (-23 million). In the scientific form, the given value must be an integer after the application of the exponent; otherwise, an error message is produced and the program terminates.

2.2 Real data values

Real data is entered just like integer data, except that the forms may also have decimal parts, i.e. 2.3 and -3.487E2.

2.3 Character data values

Character data may be entered in several ways.

- Single words may be entered without surrounding quote marks provided that the word contains no white space and does not begin with '$' or '>'
- Quoted (by " or ') data must start and end with the same quote mark (and the other quote may be contained within the string). White space and other characters may are allowed in quoted strings. In the current implementation, some parameters receive their value from only the first 10 characters in the string. However, it is not recommended that this feature be depended upon for future releases.

Examples of valid strings are:

onewordwithoutwhitespace

"A string containing 'apostrophe' characters"

'A string containing "quote" characters'

3 Relational table formats

The following are the formats of the various types of relational tables accepted by CLUSTER/2C++.

3.1 Title table

The *title* table is used to provide the caption for the data analysis experiment. Each title line is written at the top of the output report. A sample title table is shown below. The word entered under `TYPE` is ignored in the present implementation. You may enter as many lines of title data as you want.

TITLE		
TYPE	TEXT	{table identifier}
<c>	<c>	{column headings}
MAIN	"Analysis of Disease Data"	{1st data row}
SUBTITLE	'Charcoal rot and root rot'	{2nd data row}

CLUSTER/2 assumes that the first rows of the file comprise a title table, i.e. the title table identifier and column headings are implicit. For example, the following title table is valid at the beginning of a file:

```
MAIN      "This is my main title"
SUB       "This is my subtitle"
```

3.2 Parameters table

The *Parameters* table indicates controls various features of CLUSTER/2. This table must precede all other tables (except the Title table). Each line of the Parameters table requests a full iteration of the clustering algorithm, which may indicate different cluster organizations (e.g. hierarchical or flat), numbers of clusters, clustering criteria and different methods in which the clustering algorithm proceeds.

```
PARAMETERS

K       CRITERION
<i>     <c>
3       ALPHA
4       ALPHA
```

{This table implies a flat (single level) cluster organization and asks for two such clusterings, one with 3 clusters and one with 4 clusters. The user must define the "ALPHA" clustering quality criterion (see section 3.3).}

```
PARAMETERS

COVERTYPE     CRITERION
  <c>           <c>
Hierarchical    Q23
```

{This table requests a hierarchical cluster organization. CLUSTER/2 will build a classification hierarchy using the clustering quality criterion "Q23" (which must be further defined). The number of clusters formed at each level of the hierarchy will be in the default range of 2-4 clusters and no further partitioning will be done for clusters that contain no more than 4 events).

The Parameters table has numerous other optional columns. The following table shows all the available columns along with their default values.

```
PARAMETERS
K MINK MAXK CRITERION TRACE H1 H2 H3 INITMETHOD NIDSPEED COVERTYPE BASE PROBE MAXHEIGHT MINSIZE BETA
<i><i><i> <c>  <c><i><i><i> <c>      <c>       <c>  <i><i><i>  <i>  <r>
2 2  4   ALPHA OFF 3 2 3 RANDOM   FAST    DISJOINT 2  2  99     4   3.0
```

The signs denoting type are, as previously mentioned, not part of the table. The table can contain as many lines as desired. The columns are defined below:

BASE BASE and PROBE control the clustering stopping criteria. For each number of clusters considered in finding a solution, CLUSTER/2 always forms BASE number of clusterings. Then PROBE more clusterings are generated. If any of the PROBE number of clusterings is better than any previous clustering, the better cluster description is retained and another PROBE number of clusterings are generated. Under normal circumstances, the algorithm selects a unique set of seed events for each clustering attempt. When PROBE clusterings are generated without producing a "better" cluster than the best previously found, the algorithm stops and reports the BASE best clusterings

obtained. Increasing BASE increases the number of reported alternative clusterings.

BETA Specifies the relative importance of good fit of the cluster descriptions to the data for different numbers of clusters formed. The system chooses the optimized number of clusters by measuring the fit multiplied by the number of clusters raised to the BETA power. This measured, denoted as *S*, is defined as:

$$S = sparseness * k$$

where *sparseness* is the measure of fit and *k* is the number of clusters. The optimal clustering is the one with the minimal value of *S*. Increasing BETA increases the improvement of fit demanded for greater numbers of clusters (i.e. tends to bias CLUSTER/2 towards selecting clusterings with fewer clusters).

COVERTYPE Specifies the desired organization of clusters. The specifications DISJOINT causes the generation of a flat, single-level clustering having a particular number of clusters (i.e. k classes, where k is either set by the user or chosen based on BETA). INTERSECTING causes a flat, single-level clustering where the clusters may intersect. DISJOINT clusters do not overlap; each input event occurs in at most one cluster. HIERARCHICAL generates a multilevel hierarchy of clusters. The clustering algorithm is applied recursively to clusters formed until either the hierarchy contains a number of levels equal to the parameter MAXHEIGHT or the cluster to be partitioned fails to have more than MINSIZE events.

CRITERION Specifies the name of the relational table defining a criterion of clustering quality. These tables are described in Section 3.3.

H1 H1 and H3 control search heuristics. If these parameters were set to infinity (which cannot actually be done in CLUSTER/2), then the results of the search for the best clustering are optimal. Unfortunately, only the most trivial problems can be handled with an exhaustive search due to combinatorial explosion. H1 and H3 are assigned small values and the heuristic search process yields a good, usually near-optimal solution. In terms of the actual search heuristic (see [9]), H1 represents the number of complexes kept in each partial star. Increasing H1 causes more descriptions to be considered. Values greater than 10 are usually unwarranted and rarely yield improved solutions. H1 is sometimes described in the literature as *Maxstar*. Setting H1 to 1 will provide fastest clustering.

H2 Specifies the maximum number of potential cluster descriptions produced for each cluster. The actual number maintained may be smaller than H2 as the search tree is tapered on one side and has fewer alternative solutions. In terms of the actual clustering algorithm, H2 gives the maximum number of complexes in a completed star. The effects of H2 are similar to those of H1. Typical values are less than 10.

H3 Controls the persistence of the search for optimized clusterings. The

method inspects clusterings according to their Path Rank Order (the search heuristic, as described in [8] and [9]). When the number of successively ordered clusterings inspected without finding a new, best clustering reaches H3, the search terminates. Increasing H3 extends the search, possibly enabling it to find better results (but also making the algorithm run more slowly).

INITMETHOD Specifies how to pick the very first seed events. This is always RANDOM in the current implementation, but support for heuristics for finding good starting points is contemplated.

K The desired number of clusters. When specified, MAXK and MINK are both set to K.

MAXK The maximum number of clusters to form when generating a clustering. The algorithm will try clusterings containing each number of clusters between MINK and MAXK.

MINK The minimum number of clusters to form.

MAXHEIGHT The maximum allowed height of a cluster hierarchy (when COVERTYPE is HIERARCHICAL).

MINSIZE The minimum size (in number of events) of a cluster that is subject to recursive clustering into smaller parts. Only clusters that contain more than MINSIZE events will be further partitioned.

NIDSPEED The technique used to make Nondisjoint clusters Into Disjoint clusters. The value FAST causes CLUSTER/2 to use seed events as well as previously determined cluster descriptions to form the "against set" for building alternative cluster descriptions for clusters. This improves the speed of the program but places high weight on clusters that happen to be formed first. The value slow causes CLUSTER/2 to use a cluster adjusting algorithm (NID) whenever the clusters created using only seed events as the against set intersect one another. This algorithm adds to the solution time, sometimes by a large amount, but the clusters are not weighted according to the order in which they are formed and can be more optimized.

PROBE Controls the persistence of the cluster formation process. The system forms a candidate clustering and compares it (according to the clustering quality criteria) to previously generated clusterings. The system will perform PROBE number of cycles looking for a better clustering. Every time a new "best" clustering is found, another PROBE cycles are performed until PROBE successive cycles fails to find an improved solutions, at which time the algorithm halts. The algorithm also halts if all combinations of events have been considered as seeds, with the message "Panic finding new seeds". Increasing PROBE permits the program to explore more combinations of seed events and may allow the program to discover better clusterings, though the Path Rank Ordered search heuristic tends to minimize such occurrences.

TRACE Controls the generation of intermediate clustering descriptions in the

output report file. OFF limits output to just the final cluster descriptions; ON requests the output of supplemental descriptions for each intermediate clustering produced.

The Parameters table directs the work performed by CLUSTER/2. Any combination of parameter values that is to be performed must be entered as a separate row. The clustering algorithm is repeated for each row, using the parameters given on that row.

3.3 Criterion table

The -CRITERION table form may appear more than once in the input file. Each occurrence is given a unique name which consists of a specific part followed by a general part (-CRITERION). There must be Criterion table for each name specified in the CRITERION column of the Parameters table. From the previous Parameters examples, the table ALPHA=CRITERION and Q23-CRITERION tables would need to be entered, depending on which names were used in the Parameters table. Either one of the following two tables could be used to define the "ALPHA" criterion.

ALPHA-CRITERION

#	CRIT#	TOL
<i>	<i>	<r>
1	8	0.3
2	2	0.0

ALPHA-CRITERION

#	CRITERION	TOL
<i>	<c>	<r>
1	PSPARSENESS	0.3
2	DISJOINTNESS	0.0

The program applies the elementary evaluation criteria in a user defined order. The column # specifies the order of application of these criteria as specified in the **CRIT#** or **CRITERION** column. The numbers in this column must be in numerical order, starting with 1.

CRIT# Either this column or the **CRITERION** column is used to specify which elementary criterion is applied. Only one of these two columns may be present in the Criterion table. If this column is used, the criteria must be indicated by criterion numbers. If the **CRITERION** column is used, the criteria must be indicated by criterion name. The elementary criterion numbers may be taken from the definitions described in section 3.3.1. This table thus specifies the *Lexicographic Evaluation Function (LEF)*[4] used to evaluate the clustering quality.

CRITERION Used in lieu of the **CRIT#** column. Specifies the elementary criteria by name rather than number (see section 3.3.1).

TOL A positive real number, usually less than or equal to 1.0. If larger than 1.0, it is an absolute tolerance. When evaluating the LEF, clusterings within the amount of the best one are judged to be equivalent. If less than or equal to 1.0, the tolerance is taken as the fraction of the difference between the best and worst clustering scores for the elementary criterion. For example, a TOL of 0.3 means that clusterings scoring within 30% of the best clustering (on a scale from the best to worst scores) are judged to be equivalent. A tolerance of 1.0 means all clusterings are judged to be equivalent; a tolerance of 0.0 means no clusterings are judged the same

unless their scores are identical.

TOLERANCE Synonym for **TOL.**

3.3.1 Elementary Criteria Available in CLUSTER/2

The following criteria are defined in CLUSTER/2. These criteria are component parts of a user specified Lexicographic Evaluation Function (LEF) [4]. They are specified in Criterion tables by index number of name. CLUSTER/2 optimizes the generated clusterings by favoring clusters whose descriptions have the lowest scores for the elementary criteria specified in the LEF. If a negative elementary criterion index is given, or if "-" is placed directly in front of the name, the negative of the score is used. As CLUSTER/2 favors clusters with descriptions that have the lowest negatives, the effect is the same as favoring clusters whose descriptions have the highest scores.

Additional criteria will be added in the future, as will support for user-defined criteria.

The elementary criteria are presented below in order by criterion index number. The "input specification" annotation gives the keyword used in the **CRITERION** column; only the unique part of the name, which is capitalized in the input specification, need be entered.

Index	Criterion	Input Spec	Description
1	**Sparseness**	SParseness	The sparseness of a clustering is the sum of the sparsenesses of the complexes which comprise the clustering. The sparseness of a complex is the number of events it covers which are not given in the input data. Such events are "unobserved" events. The sparseness is computed by calculating the "area" of the complex and subtracting from it the number of points which correspond to the observed events. The "area" of a complex is the number of points (each corresponding to a possible event) in the subset of the event space covered by the complex: *Suppose that complex* $[x_1=2,4][x_3=4..7]$ *covers 4 observed events.* *(assume the variables are* x_1, x_2, x_3 *and the domain of* x_2 *contains 3 values).* The "area" of the complex is the product of the numbers of values in the references for each variable. In this example, $area = 2 \times 3 \times 4 = 24$ *(2 values for* x_1, *3 values for* x_2 *(its entire domain) and 4 values for* x_3*).*

Index	Criterion	Input Spec	Description
			The sparseness of this complex is "area" - #observed events:
			$sparseness = 24\text{-}4 = 20$
2	**Disjointness**	DISjointness	The disjointness is the degree of intersection of a complex. It is determined as the sum of the degrees of intersection of each pair of complexes in the clustering. The degree of intersection of two complexes is the total number of selectors that involve the same variable and have intersecting reference sets. The following example involves three variables x_1, x_2, x_3
			a_1: $[x_1=1..3][x_2=4]$
			a_2: $[x_1=2..5][x_2=2]$
			a_3:$[x_1=4][x_2=3][x_3=2]$
			The degree of intersection of complex a_1 with a_2 is 4 because the references used with variable x_1 in each complex contain intersecting values and the selectors omitted for x_3 also intersect (a "dropped" selector intersects with anything, as it is equivalent to one with all possible domain values in the reference list). The degree of intersection of complex a_3 with a_1 is 2 because two selectors (for x_3) intersect.
3	**Number of events in >1 complex**	Multcov	This criterion is for use only when considering clusters. It is not used in the current implementation.
4	**Balance**	Balance	Measures the unevenness in the cluster populations. It is determined as the sum of the deviations from uniform distribution of observed events in each complex. For one complex, this deviation is calculated as the absolute value of the difference between the actual number of observed events covered and 1/k of the total number of observed events, where k is the number of complexes (clusters):
			Balance = |#covered events - (#observed events/k)|
5	**Commonality**	Commonality	Measures the number of common attributes in the cluster descriptions by counting the number of selectors in all clusters (not

Index	Criterion	Input Spec	Description
			including dropped selectors). The negative of the number of selectors is used in order to maximize the commonality by minimizing the numerical score. *Commonality = -#undropped selectors*
6	**Dimension-ality**	DIMensionality	Measures the number of variables which take on different values in every complex. The number reported for each complex is the number of variables in the complex which occur in selectors in all complexes of the clustering with mutually disjoint reference values:
7	**Simplicity**	SImplicity	The sum of the simplicities of all selectors contained in all complexes in the clustering. The simplicity of a selector is the sum of the cost of the variable plus the cost of the reference list. Variable costs are provided by the user via the Variables table (section 3.4); otherwise, they have a default cost of 1. Reference list costs are computed according to the type of the domain: Structured 0 Linear 0 (1 value), 1 (otherwise) Nominal #vals in ref list - 1
8	**Projected Sparseness**	Psparseness	Calculated just like regular sparseness, but not all dimensions of the event space are considered. When calculating the "area" used in calculating projected sparseness, only the variables which are present (not dropped) in at least one complex of the clustering are considered. This is equivalent to imagining that the domain sizes of all universally dropped variables is just 1: *Suppose that $[x_1=2,4][x_3=4..7]$ covers 4 observed events.* The factors used in computing the "area" of the complex depend, in part, on other complexes in the clustering. *If the variable x_5 has a domain containing 6 values and appears in some cluster description, then it is used to calculate the "area" of a complex as follows:*

Index	Criterion	Input Spec	Description
			"area" = 2 x 4 x 6
			where 2 and 4 are the number of values in the reference lists for x_1 and x_3, respectively, and 6 is the number of levels in variable x_5.
			Psparseness = (2x4x6) - 4 = 44.
			Projected sparseness may be negative.
9	**Number of exceptional events**	Except	Under some circumstances, the formation of a clustering fails to cover all events. Uncovered events are placed into an exceptions category, which is displayed in the output. Most elementary criteria are applied to each cluster description (i.e. to each complex) and the sum over all cluster descriptions is used to evaluate the entire clustering. The number of exceptional events is only applicable to the entire clustering. When applied to individual complexes, this criterion produces a score of 0 as the exceptional events list is a property of the clustering, not an individual cluster.

3.4 Domains table

The Domains table is used to define variable domains that are likely to apply to several variables. Variables may be defined entirely with a Variables table (described after the Domains table description) or with the help of the Domains table. When many variables have exactly the same domain type and number of values, it is often more convenient to define the domain first, then just reference it in the Variables table, rather than redefining the domain over and over again. A simple domain definition is shown below.

DOMAINS

NAME	TYPE	LEVELS
<c>	<c>	<i>
Color	NOM	4
Size	LIN	3

NAME Specifies the name of the domain. In the sample table above, two different domains are defined: *Color* and *Size*.

TYPE Specifies the type of the domain. Only the first letter of the word given is significant: *N* for nominal, *L* for linear and *S* for structured. The discussion of **TYPE** below further describes what these types mean.

LEVELS The number of distinct values in the domain. If the number of levels is *j*, the domain consists of the integer values from *0* to *j-1*.

COST Not shown in the above table. Indicates the cost of a variable having this domain, used in some of the clustering quality criteria (see section 3.3). This column is optional (default cost = 1.0) and takes a real (<r>).

3.5 Variables table

The Variables table describes the attribute variables used to represent each event. A simple Variables table is shown below.

VARIABLES

#	TYPE	LEVELS
<i>	<c>	<i>
1	NOM	4
2	LIN	3
3	LIN	5

Gives the variable ordinal. The above table defines the variables X_1, X_2, and X_3. The ordinal values must be in ascending order, starting with 1.

TYPE Specifies the domain type of each variable. Only the first letter of each word given under **TYPE** is significant. Nominal (N) denotes an unordered qualitative domain (e.g. Color would have values red, green, blue, et al.). Linear (L) denotes an ordered quantitative domain (e.g. Height, which simply has numeric values). When Structured (S) is specified a STRUCTURE table must be entered (see section 3.9 below).

LEVELS Gives the number of distinct values in the domain. If the number of levels is *j*, CLUSTER/2 requires that the data values be integers from *0* to *j-1*. The event descriptions are entered as a vector of integers, one integer for each variable.

In printing the descriptions for each cluster, CLUSTER/2 refers to the variables as X1, X2, X3, etc. A more descriptive variable name can be specified by adding a **NAME** column to the Variables table. For example, the following table defines the variables *SHAPE*, *SIZE* and *WEIGHT*.

VARIABLES

#	TYPE	LEVELS	NAME
<i>	<c>	<i>	<c>
1	NOM	4	SHAPE
2	LIN	3	SIZE
3	LIN	5	WEIGHT

As with the Domain table, there is an optional **COST** column which can accept any real value. **COST** is currently used in calculating the Simplicity clustering quality criterion. If no **COST** is specified, the default cost is 1.

To indicate that a variable has a domain identical to a predefined domain (defined via a Domains table), the name of the variable is entered as <variablename>-<domainname>, i.e. as a character string composed of the desired variable name, a hyphen and the predefined domain name. When this is done, the **TYPE**, **LEVELS** and **COST** for the variable are taken from the domain definition regardless of whether they are specified in the Variables table. If any domain has associated Names, Onames or Structure tables (as defined in sections 3.7-3.9), then those tables must precede the Variables table.

3.6 Events table

The Events table holds the attribute vectors for all objects or situations being studied. Each object or situation is defined by one row of integer values (subsequent sections describe how to handle symbolic attribute values). Each attribute value must be placed in its corresponding column and must be in the range from *0* up to *LEVELS-1*, where LEVELS denotes the number of levels for that attribute as specified in the Variables table. The headings for the Events table are the names of the variables. If the variables are defined without the NAME column option; the variables names are in the form of X*n*, where *n* is the variable ordinal (e.g. X1). If the variable is defined with the NAME column, the given name is used without any hyphen or domain name part (e.g. Shape). The following Events table shows the form used with variables that are not given a name.

EVENTS

#	X1	X2	X3
<i>	<i>	<i>	<i>
1	0	1	1
2	2	1	0
3	1	3	3
4	1	2	3

The same table may be entered in the following form if the variables are given names. The heading keywords must match the values from the **NAME** column of the Variables table (they need not be in the same order).

EVENTS

#	Shape	Size	Weight
<i>	<i>	<i>	<i>
1	0	1	1
2	2	1	0
3	1	3	3
4	1	2	3

The # column gives the event number. The values in this column must be in sequence and must begin with the value 1.

When working with events described by very long attribute vectors, the number of columns in the event table (one column per attribute) may make the tables unmanageably wide. CLUSTER/2 permits such tables to be split into multiple pieces. To make use of this feature, the Events table is specified several times, with different column headings (for different variables) used in each Events table. Each table still must contain a # column. All of the separate Events tables must have the same number of rows and the rows must be in exactly the same order. CLUSTER/2 collects the values from the *i*th row of each tables into the attribute description of the *i*th event.

The optional column **WT** may be entered to provide a weight for each event. The weight may be used to indicate the relative frequency of observation of the event. The use of weighted events alters the calculation of sparseness. Each observed event counts according to its weight, so a complex with an "area" of 4 and covering 2 observed events receives a sparseness of (4 minus the combined weights of the covered events). With large weights, the sparseness calculation may produce negative values (this is perfectly valid). Negative sparseness should be interpreted as a measure of fit something like weight density. The more weight of covered events, the more negative the sparseness.

3.7 Names table

A Names table may be entered for any variable or domain definition. The specific part of the table name is the name of the variable (either X*n* or as given in the **NAMES** column of the Variables table) or domain. The Names table specifies a symbolic name to be used in lieu of the integer variable value in both the reported cluster descriptions **and** the input of the event data (i.e. values for this variable in an Events table must be symbolic rather than integral). The Names table is illustrated below.

SHAPE-NAMES

VALUE	NAME
<i>	<c>
0	Square
1	Rectangle
2	Circle
3	Oval

For linearly ordered variables and domains, the **VALUE** establishes the ordering of the symbols within the domain. If a single value denotes a range of observed features (e.g. the value 3 denotes a temperature range 68 to 72) the name should be written *lowname..highname* (e.g. *68..72*). The use of the .. range indicator allows CLUSTER/2 to combine names appropriately when reporting an interval value of a linearly ordered variable.

3.8 Onames table

The Onames table may be entered in lieu of a Names table. The Onames table specifies output-only names for the integer values of the domain of the variable. If the input data is numerical, an Onames table will allow the output to be symbolic while the input remains numerical, while using the Names table requires that the input be symbolic. An Onames table contains the same columns as a Names table.

3.9 Structure table

Variables defined as a Structured **TYPE** have a tree structured (or graph structured) domain. The **LEVELS** value given in the Variables (or Domains) table refers to the number of discrete leaf values in the tree or graph. Higher order node values (generalized values of the variable) are defined by the Structure table. The specific part of the table name is the name of the variable (either X*n* or as given in the **NAME** column of the Variables table).

The Structure table takes two forms, depending on the use of an associated Names table. If no Names table is used for the variable, the Structure table appears as follows:

SHAPE-STRUCTURE

VALUE	SUBVALUE	(SUBVALUE)	(SUBVALUE)	...
<i>	<i>	<i>	<I>	...
4	2	3	...	
5	0	1	...	

The **VALUE** column and the first **SUBVALUE** column are required. Additional **SUBVALUE** columns may be used by including **SUBVALUE** several times in the column heading list. In the above illustration, the parentheses around SUBVALUE denote that the column is optional; they are not used in the actual input file. For each row in the Structure table, the value in the **VALUE** column is defined as the parent of the values in the **SUBVALUE** columns. Thus each row of the Structure table is defining one or more links in a tree structured domain. Any nodes or leaf values that have no explicitly defined parent are implicitly parented by the root of the tree. The value that occur under **VALUE** must be greater than any regular (i.e. leaf) value for the variable. If the **LEVELS** column of the Variables table specifies *j* levels, the leaf values in the domain are denoted by values from *0* to *j-1* and the values *j*. *j+1*, *j+2*, ... could appear in the **VALUE** column of the Structure table to denote generalized domain values. In the example above, the variable SHAPE has four levels (values 0 to 3). Two new values (4 and 5) are defined as generalizations of values 2 and 3 and values 0 and 1, respectively.

If a Names table is given for the variable (the Names table must precede the Structure table for the same variable), the following form of the Structure table is to be used:

SHAPE-STRUCTURE

NAME	SUBNAME	SUBNAME
<c>	<c>	<c>
4-Sided	Square	Rectangle
Circular	Circle	Oval

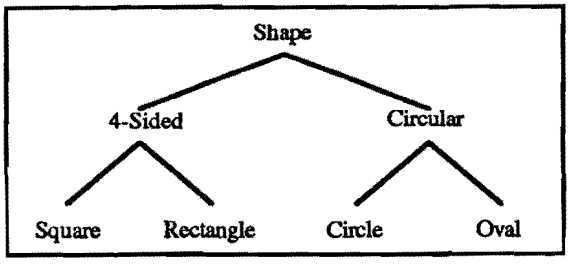

Figure 1. Semantics of Shape structure

The tree shown in Figure 1 above illustrates the semantics of this Structure table. This form of the Structure table has all the options of the previous one, but with symbolic (character data) names for all values. The names given to the defined generalized domain values must be unique among all names (including leaf names) in the same domain.

If a generalized value must be the parent to a great number of subvalues (and subnames), the table may become unmanageable. In such cases it is permitted to repeat the **VALUE** (or **NAME**) entry on adjacent lines, giving part of the subvalue (subname) list on each line. CLUSTER/2 merges the definitions together into one specification of a generalized value for all of the subvalues mentioned.

4 A Sample Input File

The following sample input file makes use of the relational table forms described above. Some comments about the interpretation of this example are given below.

```
MAINTITLE "Microcomputers"

PARAMETERS
K       TRACE       CRITERION       NIDSPEED       COVERTYPE
2       ON          DS              SLOW           DISJOINT
$       OFF         FC              FAST           HIERARCHICAL

DS-CRITERION
#           CRITERION       TOLERANCE
1           DIS             0.3
2           COM             0.0
3           PSPARS          0.0

FC-CRITERION
#           CRITERION       TOLERANCE
1           PSPARS          0.3
2           SIM             0.0

VARIABLES
#           NAME            TYPE            LEVELS
1           MP              STR             13
2           RAM             LIN             4
3           ROM             LIN             7
4           DISP            STR             4
5           KEYS            LIN             5

MP-NAMES
VALUE       NAME
0           8080A
1           6502
2           Z80
3           1802
4           6502C
5           6502C
6           68000
7           6800
8           6805
```

```
9          6809
10         8048
11         Z8000
12         HP

MP-STRUCTURE
NAME              SUBNAME          SUBNAME          SUBNAME          SUBNAME
6502X             6502             6502C            6502A            $
8080X             8080A            Z80              8048             $
8BIT              6502X            8080X            1802             6800
8BIT              6805             6809             $                $
16BIT             68000            Z8000            HP               $

RAM-ONAMES
VALUE      NAME
0          16K
1          32K
2          48K
3          64K

ROM-ONAMES
VALUE      NAME
0          1K
1          4K
2          8K
3          10K
4          11K..16K
5          26K
6          80K

DISP-ONAMES
VALUE      NAME
0          TERMINAL
1          "B/W TV"
2          "COLOR TV"
3          BUILT-IN
```

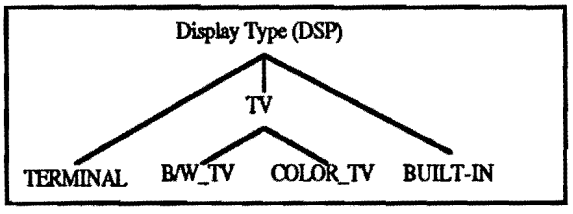

Figure 2. Display structure

```
DISP-STRUCTURE
NAME     VALUE          SUBVALUE       SUBVALUE
TV       4              1              2

KEYS-ONAMES
VALUE      NAME
0          52
1          53..56
2          57..63
3          64..73
4          92
```

EVENTS

#	MP	RAM	ROM	DISP	KEYS
1	6502	2	3	2	0
2	6502	2	3	2	2
3	6502A	1	4	2	3
4	Z80	2	1	1	2
5	8080A	3	0	3	3
6	Z80	3	2	3	3
7	HP	1	6	3	4
8	Z80	3	2	0	2
9	6502	1	3	1	1
10	6502C	2	3	1	1
11	Z80	2	4	1	1
12	Z80	2	4	3	3

<END OF FILE>

The tables used in the above file are PARAMETERS, DS-CRITERION, FC-CRITERION, VARIABLES, MP=NAMES, MP-STRUCTURE, RAM-NAMES, ROM-NAMES, DISP-NAMES, DISP-STRUCTURE, KEYS-NAMES and EVENTS. The entire clustering algorithm will be performed twice (because there are two rows in the Parameters table). The two different user-defined criterion names (DS and FC) have Criterion tables which define the elementary criteria used. The variables MP, RAM, ROM, DISP and KEYS have names associated with their integer data values; there is a Names table for each variable. The variables MP and DISP have structured domains; there is a Structure table for each of these two variables.

The table MP-STRUCTURE presents a graph-structured domain in which the generalized values 6502x (members of the 6502 family of microprocessors), 8080x, 8bit and 16bit are defined. The inset shows the graph-structured domain of the DISP variable as defined by the DISP-STRUCTURE table.

5 Program Output Corresponding to the Sample Input File

The beginning of the output listing shows the Parameters table and the two Criterion tables. Since the Parameters table contains two rows of data, CLUSTER/2 will be invoked twice, once for k=2 and a "flat" clustering, once for hierarchical clustering in which the program will select the best k, $2 \leq k \leq 4$, using the sparseness calculation described in section 3.2 with the default b (3.0). Parts of the remaining listing that are of particular interest have been marked with numbers.

Location 1 in the listing marks the beginning of the first clustering, which is the first experiment. The criterion applied is "ds", as defined by the DS-CRITERION table. The number of clusters is set, via the Parameters table, to exactly 2. Since trace mode is on for this experiment (also from the Parameters table), each clustering that is formed is described in the listing. Each clustering is described as a "COVER". The complexes in the clustering are denoted by "Complex <complex#>" followed by a rule describing the cluster as a series of variable selectors. Following are the costs under each clustering criterion for the specified LEF (e.g. for Complex 1 of the first cover, a value of 4 for disjointness, -3 for commonality and 3016 for projected sparseness). Next comes the seed event for that complex (e.g. event #1) and the other events covered by the complex (e.g. 1, 3, 4, 5, 6, 7, 8, 12). This is repeated for each complex. Then, the totals for the DS-CRITERION on this clustering are shown (8, -8, 3172 for each of the three criteria), followed by a list of any events that could not be covered

(labeled "Exceptions"; there are none for this clustering). For this first clustering, the first complex covers microprocessors other than the 1802, with between 32K and 64K of RAM, and between 57 and 92 keys. The second cluster covers 8 bit microprocessors with between 32K and 48K of RAM, between 10K and 16K of ROM and a TV display.

Position #2 shows the best cover, based on the DS-CRITERION. The two complexes are:

- Eight bit microprocessors with RAM between 32K and 64K, between 1K and 16K of ROM, and between 52 and 73 keys on their keyboards.
- Hewlett Packard microprocessors with 32K RAM, 80K ROM, a built-in display and a 92 key keyboard.

Position #3 shows the results of the second experiment (using FC-CRITERION as the clustering quality criterion) for k=2. Note that the best cluster found is quite different from the first experiment -- now it consists of:

- Non 8080X microprocessors with 32-48K of RAM, 10-80K of ROM and non-terminal displays.
- 8080X microprocessors with 48-64K of RAM, 1-16K of ROM, non-color-tv displays and keyboards with between 53 and 73 keys.

Positions #4 and #5 show 3-class and 4-class solutions to the clustering problem. Position #6 shows the selection of the "best solution", based on the sparseness calculation discussed earlier, and Position #7 shows a hierarchical breakdown of the best clustering.

```
maintitle  "microcomputers"

parameters
   k  print  trace  criterion nidspeed covertype
   2    a      on       ds      slow    disjoint
   $    a      off      fc      fast    hierarchical

   ds-criterion
 #  crit tolerance
 1    2      0.3
 2    5      0.0
 3    8      0.0

   fc-criterion
 #  crit  tolerance
 1    9      0.0
 2    8      0.3
 3    7      0.0

variables
   #   name      type   levels
   1    mp        str      13
   2    ram       lin       4
   3    rom       lin       7
   4    disp      str       4
   5    keys      lin       5

mp-names
```

```
 value name
   0   8080a
   1   6502
   2   z80
   3   1802
   4   6502c
   5   6502a
   6   68000
   7   6800
   8   6805
   9   6809
  10   8048
  11   z8000
  12   hp

mp-structure
 name      subname subname subname subname
 6502x       6502    6502c   6502a   $
 8080x       8080a   z80     8048    $
 8bit        6502x   8080x   1802    6800
 8bit        6805    6809    $       $
 16bit       68000   z8000   hp      $

 ram-onames
 value name
   0   16k
   1   32k
   2   48k
   3   64k

 rom-onames
 value name
   0   1k
   1   4k
   2   8k
   3   10k
   4   11k..16k
   5   26k
   6   80k

 disp-onames
 value name
   0   terminal
   1   "b/w_tv"
   2   "color_tv"
   3   built-in

 disp-structure
 name         value subvalue subvalue
  tv           4     1        2

 keys-onames
 value name
   0   52
```

```
1   53..56
2   57..63
3   64..73
4   92
```

events

#	mp	ram	rom	disp	keys
1	6502	2	3	2	2
2	6502	2	3	2	0
3	6502a	1	4	2	3
4	z80	2	1	1	2
5	8080a	3	0	3	3
6	z80	3	2	3	3
7	hp	1	6	3	4
8	z80	3	2	0	2
9	6502	1	3	1	1
10	6502c	2	3	1	1
11	z80	2	4	1	1
12	z80	2	4	3	3

ƒƒ

Conjunctive conceptual clustering program CLUSTER/2C++
Last upgrade: 08/23/94

#1

```
**** COVER ***
Complex 1:[mp<>1802] [ram=32k..64k] [keys=57..92]                       Costs: 4 -3 3016 Seed: 1 Covered: 1 3 4 5 6 7 8 12
Complex 2:[mp=8bit] [ram=32k..48k] [rom=10k..16k] [disp=tv] [keys=52..56]      Costs: 4 -5 156 Seed: 2 Covered: 2 9 10 11
TOTALS: 8 -8 3172
Exceptions:
******************
**** COVER ***
Complex 1:[mp=8080x] [ram=48k..64k] [rom=1k..16k] [disp<>color_tv] [keys=53..73] Costs: 4 -5 264 Seed: 12 Covered: 4 5 6 8
11 12
Complex 2:[mp<>8080x] [ram=32k..48k] [rom=10k..80k] [disp<>terminal]           Costs: 4 -4 1194 Seed: 9 Covered: 1 2 3 7 9
10
TOTALS: 8 -9 1458
Exceptions:
******************
**** COVER ***
Complex 1:[mp=8080x] [ram=48k..64k] [rom=1k..16k] [disp<>color_tv] [keys=53..73] Costs: 4 -5 264 Seed: 6 Covered: 4 5 6 8 11
12
Complex 2:[mp<>8080x] [ram=32k..48k] [rom=10k..80k] [disp<>terminal]           Costs: 4 -4 1194 Seed: 1 Covered: 1 2 3 7 9
10
TOTALS: 8 -9 1458
Exceptions:
******************
**** COVER ***
Complex 1:[mp=8bit] [ram=32k..64k] [rom=1k..16k] [keys=52..73]           Costs: 2 -4 2389 Seed: 11 Covered: 1 2 3 4 5 6 8 9 10
11 12
Complex 2:[mp=hp] [ram=32k] [rom=80k] [disp=built-in] [keys=92]           Costs: 2 -5 0 Seed: 7 Covered: 7
TOTALS: 4 -9 2389
Exceptions:
******************
**** COVER ***
Complex 1:[mp=8bit] [ram=32k..64k] [rom=1k..16k] [keys=52..73]           Costs: 2 -4 2389 Seed: 4 Covered: 1 2 3 4 5 6 8 9 10 11
12
Complex 2:[mp=hp] [ram=32k] [rom=80k] [disp=built-in] [keys=92]           Costs: 2 -5 0 Seed: 7 Covered: 7
TOTALS: 4 -9 2389
Exceptions:
******************
**** COVER ***
Complex 1:[mp=8bit] [ram=32k..64k] [rom=1k..16k] [keys=52..73]           Costs: 2 -4 2389 Seed: 5 Covered: 1 2 3 4 5 6 8 9 10 11
12
Complex 2:[mp=hp] [ram=32k] [rom=80k] [disp=built-in] [keys=92]           Costs: 2 -5 0 Seed: 7 Covered: 7
TOTALS: 4 -9 2389
Exceptions:
******************
```

k=2, best clustering: **** COVER ***
Complex 1:[mp=8bit] [ram=32k..64k] [rom=1k..16k] [keys=52..73] Costs: 2 -4 2389 Seed: 11 Covered: 1 2 3 4 5 6 8 9 10 11 12
Complex 2:[mp=hp] [ram=32k] [rom=80k] [disp=built-in] [keys=92] Costs: 2 -5 0 Seed: 7 Covered: 7
TOTALS: 4 -9 2389
Exceptions:
****

22 stars built
****

#2

*** Best solution (beta=3.0): S=1.911e+04, k = 2 ***
**** COVER ***
Complex 1:[mp=8bit] [ram=32k..64k] [rom=1k..16k] [keys=52..73] Costs: 2 -4 2389 Seed: 11 Covered: 1 2 3 4 5 6 8 9 10 11 12
Complex 2:[mp=hp] [ram=32k] [rom=80k] [disp=built-in] [keys=92] Costs: 2 -5 0 Seed: 7 Covered: 7
TOTALS: 4 -9 2389
Exceptions:
****
***

#3

k=2, best clustering: **** COVER ***
Complex 1:[mp=8080x] [ram=48k..64k] [rom=1k..16k] [disp<>color_tv] [keys=53..73] Costs: 0 264 8 Seed: 12 Covered: 4 5 6 8 11 12
Complex 2:[mp<>8080x] [ram=32k..48k] [rom=10k..80k] [disp<>terminal] Costs: 0 1194 6 Seed: 9 Covered: 1 2 3 7 9 10
TOTALS: 0 1458 14
Exceptions:
****

14 stars built
****

#4

k=3, best clustering: **** COVER ***
Complex 1:[mp=8080x] [ram=48k..64k] [rom=1k..16k] [disp<>color_tv] [keys=53..73] Costs: 0 264 8 Seed: 6 Covered: 4 5 6 8 11 12
Complex 2:[mp=6502x] [ram=32k..48k] [rom=10k] [disp=tv] [keys=52..63] Costs: 0 32 7 Seed: 9 Covered: 1 2 9 10
Complex 3:[mp<>8080x] [ram=32k] [rom=11k..80k] [disp<>terminal] [keys=64..92] Costs: 0 178 7 Seed: 3 Covered: 3 7
TOTALS: 0 474 22
Exceptions:
****

28 stars built
****

#5

k=4, best clustering: **** COVER ***
Complex 1:[mp=6502x] [ram=32k..48k] [rom=10k..16k] [disp=tv] [keys=52..73] Costs: 0 91 8 Seed: 1 Covered: 1 2 3 9 10
Complex 2:[mp=z80] [ram=48k] [rom=4k..16k] [disp<>terminal] [keys=53..73] Costs: 0 33 7 Seed: 4 Covered: 4 11 12
Complex 3:[mp=hp] [ram=32k] [rom=80k] [disp=built-in] [keys=92] Costs: 0 0 5 Seed: 7 Covered: 7
Complex 4:[mp=8080x] [ram=64k] [rom=1k..8k] [disp<>tv] [keys=57..73] Costs: 0 33 7 Seed: 5 Covered: 5 6 8
TOTALS: 0 157 27
Exceptions:
****

96 stars built
****

#6

*** Best solution (beta=3.0): S=1.166e+04, k = 4 ***
**** COVER ***
Complex 1:[mp=6502x] [ram=32k..48k] [rom=10k..16k] [disp=tv] [keys=52..73] Costs: 0 91 8 Seed: 1 Covered: 1 2 3 9 10
Complex 2:[mp=z80] [ram=48k] [rom=4k..16k] [disp<>terminal] [keys=53..73] Costs: 0 33 7 Seed: 4 Covered: 4 11 12
Complex 3:[mp=hp] [ram=32k] [rom=80k] [disp=built-in] [keys=92] Costs: 0 0 5 Seed: 7 Covered: 7
Complex 4:[mp=8080x] [ram=64k] [rom=1k..8k] [disp<>tv] [keys=57..73] Costs: 0 33 7 Seed: 5 Covered: 5 6 8
TOTALS: 0 157 27
Exceptions:
****
***

Breakdown for cover

```
**** COVER ***
Complex 1:[mp=6502x] [ram=32k..48k] [rom=10k..16k] [disp=tv] [keys=52..73]      Costs:  0 91  8 Seed: 1 Covered: 1 2 3 9 10
Complex 2:[mp=z80] [ram=48k] [rom=4k..16k] [disp<>terminal] [keys=53..73]      Costs:  0 33  7 Seed: 4 Covered: 4 11 12
Complex 3:[mp=hp] [ram=32k] [rom=80k] [disp=built-in] [keys=92]                Costs:  0  0  5 Seed: 7 Covered: 7
Complex 4:[mp=8080x] [ram=64k] [rom=1k..8k] [disp<>tv] [keys=57..73]           Costs:  0 33  7 Seed: 5 Covered: 5 6 8
TOTALS:  0 157  27
Exceptions:
**********
```

#7

```
          **********
k=4, best clustering: **** COVER ***
Complex 1:[mp=6502c] [ram=48k] [rom=10k] [disp=b/w_tv] [keys=53..56]           Costs:  0  0  5 Seed: 10 Covered: 10
Complex 2:[mp=6502] [ram=48k] [rom=10k] [disp=color_tv] [keys=52..63]          Costs:  0  1  6 Seed: 2 Covered: 1 2
Complex 3:[mp=6502a] [ram=32k] [rom=11k..16k] [disp=color_tv] [keys=64..73]    Costs:  0  0  5 Seed: 3 Covered: 3
Complex 4:[mp=6502] [ram=32k] [rom=10k] [disp=b/w_tv] [keys=53..56]            Costs:  0  0  5 Seed: 9 Covered: 9
TOTALS:  0  1  21
Exceptions:
**********
48 stars built
**********


*** Best solution (beta=3.0): S= 64, k = 4 ***
**** COVER ***
Complex 1:[mp=6502c] [ram=48k] [rom=10k] [disp=b/w_tv] [keys=53..56]           Costs:  0  0  5 Seed: 10 Covered: 10
Complex 2:[mp=6502] [ram=48k] [rom=10k] [disp=color_tv] [keys=52..63]          Costs:  0  1  6 Seed: 2 Covered: 1 2
Complex 3:[mp=6502a] [ram=32k] [rom=11k..16k] [disp=color_tv] [keys=64..73]    Costs:  0  0  5 Seed: 3 Covered: 3
Complex 4:[mp=6502] [ram=32k] [rom=10k] [disp=b/w_tv] [keys=53..56]            Costs:  0  0  5 Seed: 9 Covered: 9
TOTALS:  0  1  21
Exceptions:
**********
***
          Breakdown for cover
**** COVER ***
Complex 1:[mp=6502c] [ram=48k] [rom=10k] [disp=b/w_tv] [keys=53..56]           Costs:  0  0  5 Seed: 10 Covered: 10
Complex 2:[mp=6502] [ram=48k] [rom=10k] [disp=color_tv] [keys=52..63]          Costs:  0  1  6 Seed: 2 Covered: 1 2
Complex 3:[mp=6502a] [ram=32k] [rom=11k..16k] [disp=color_tv] [keys=64..73]    Costs:  0  0  5 Seed: 3 Covered: 3
Complex 4:[mp=6502] [ram=32k] [rom=10k] [disp=b/w_tv] [keys=53..56]            Costs:  0  0  5 Seed: 9 Covered: 9
TOTALS:  0  1  21
Exceptions:
**********
          **********
          End breakdown
End breakdown
Elapsed time: 284.42 seconds
```

6 Future Improvements

The following are improvements currently being worked on:

1. Interactive, graphical and menu driven input support. The user will no longer be required to use batch input methods, though support will remain for batch input files, and will receive more guidance in parameter setting.

2. Comprehensive, graphical output. Rather than the spartan output currently produced by the program, improved analytical tools will be provided and the user will be able to get a better read on the output. Included will be support for an ordered listing of discovered clusterings.

3. Additional built-in criteria of clustering quality

4. Support for user-defined clustering quality criteria

5. Links to popular databases, so that input need not be reformatted for introduction into CLUSTER/2C++

6. Performance improvements, both in reduction of memory requirements and in reduced running time.

7. Numerous other code improvements

References

1. Biswas, B. "A Programmer's Guide to CLUSTER: A Program for Conjunctive Conceptual Clustering", ISG report UIUDCDC-F-83-810, Department of Computer Science, University of Illinois, Urbana, July 1983.

2. Fischthal, S.M., "Fraud Detection Using a Combination of Neural Networks and Conceptual Clustering", Loral Federal Systems Technical Report, February 1995.

3. Fischthal, S.M., "Low Level Design and Code Description for CLUSTER/2C++", 1995.

4. Michalski, R.S., "Knowledge Acquisition Through Conceptual Clustering: A Theoretical Framework and an Algorithm for Partitioning Data into Conjunctive Concepts", special issue on knowledge acquisition and induction, *International Journal of Policy Analysis and Information Systems*, **4**:3, pp.219-44, 1980.

5. Michalski, R.S., Stepp, R.E., Diday, E., "A Recent Advance in Data Analysis: Clustering Objects into Classes Characterized by Conjunctive Concepts", invited chapter in *Progress in Pattern Recognition,* Vol I, Kanal, L. and Ronsenfeld, A. (Eds.), pp. 33-55, 1981.

6. Michalski, R.S., Stepp, R.E., "Revealing Conceptual Structure in Data by Inductive Inference", in *Machine Intelligence* **10**, Hayes, J.E., Michie, D., Pao, Y.-H. (Eds.), Ellis Horwoood, Chichester, Halsted Press (John Wiley), New York, 1981.

7. Michalski, R.S., Stepp, R.E., "An Application of AI Techniques to Structuring Objects into an Optimal Conceptual Hierarchy", *Proceedings of the 7th International Joint Conference on Artificial Intelligence,* Vancouver, Canada, August 24-8, pp. 460-5, 1981.

8. Michalski, R.S., Stepp, R.E., "Automated Construction of Classifications: Conceptual Clustering vs. Numerical Taxonomy", *IEEE Transactions on PAMI,* **5**:4, pp. 396-410, 1983.

9. Michalski, R.S., Stepp, R.E., "Learning from Observation: Conceptual Clustering", chapter in book *Machine Learning: an Artificial Intelligence Approach*, Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (Eds.), Morgan-Kaufmann: San Mateo, CA, 1983.

10. Srivastava, A., Murty, M.N., "A Comparison Between Conceptual Clustering and Conventional Clustering", *Pattern Recognition* **23**:9, pp.975-81, 1990.

11. Stepp, R.E., "A Description and User's Guide for CLUSTER/2: A Program for Conjunctive Conceptual Clustering", report 83-30, Department of Computer Science, University of Illinois, Urbana, November 1983.

12. Stepp, R.E., "Conjunctive Conceptual Clustering: a Methodology and Experimentation", Ph.D. Thesis, UIUCDCS-R-84-1189, Department of Computer Science, University of Illinois, Urbana, November 1984.