



RepGrid • WebGrid • RepGrids
RepNet • RepDoc • RepScript

Rep Plus

Conceptual Representation Software

RepGrid Manual

Eliciting, Entering, Editing and
Analyzing a Conceptual Grid

May 2021

Brian R Gaines and Mildred L G Shaw

<http://cpsc.ucalgary.ca/~gaines/repplus/>

Contents

Contents	i
1 Conceptual Grids	1
1.1 RepGrid, WebGrid and RepGrids	1
2 Opening, saving and creating grid files	3
2.1 Opening a grid file	3
2.2 Save, Save As	4
2.3 Undo	4
2.4 New	4
2.5 Copy, Exchange, Elements, Constructs—Open dialogue or drag and drop	5
3 Editing a grid	6
3.1 Options pane	7
3.1.1 Grid description	7
3.1.2 Grid terminology and defaults	7
3.1.3 Default rating scale	7
3.1.4 Grid annotation	8
3.1.5 Rating scale data types: ratings, categories, integers, numbers	8
3.1.6 Metavalues in ratings: open, unknown, any, none, inapplicable	8
3.2 Elements pane	10
3.2.1 Element annotation and weights	11
3.2.2 Editing element data	12
3.2.3 Sorting elements	13
3.3 Constructs pane	13
3.3.1 Construct names	14
3.3.2 Construct annotations, weights and reversal	15
3.3.3 Editing the ratings of elements on constructs	16
3.3.4 Assigning categories to rating scale ranges	16
3.3.5 Categorical, numeric and integer construct rating scale types types	18
3.3.6 Representing ordinal relations between constructs	22
3.4 Classes—intersects and anticipation	24
3.4.1 Classes Pane	25
3.4.2 Anticipation: classes as intersects, templets, cases, rules	27
3.4.3 Editing class meanings	27
3.4.4 Using classes for classification	29

3.4.5	Exporting classes as descriptions, logical expressions and conceptual nets . . .	30
3.4.6	Using the Classes pane analyses with grids where classes have not been speci- fied	32
3.4.7	Classes as ideal elements or compound constructs in grids	33
3.5	Items pane	34
3.5.1	User-defined items	35
3.6	Scripts pane	36
4	Grid entry, elicitation and export scripts	39
4.1	Enter Grid	39
4.2	Elicit Grid	44
4.3	Export grid data	51
4.4	Analyze grid data	51
4.5	Modifying scripts	51
5	Grid display and analysis	52
5.1	Display: Plotting the grid as a matrix of ratings of elements on constructs	52
5.1.1	Display plot output	53
5.1.2	Including classes as elements or constructs in the analyses	54
5.2	Synopsis: Histograms and scree plot	55
5.2.1	Synopsis histogram and scree plots output	57
5.3	Focus: Sorting by similarity and hierarchical clustering	58
5.3.1	Focus cluster plot output	59
5.3.2	Focus data output	61
5.3.3	Status of the Focus hierarchical clusters	62
5.4	PrinGrid Map: Spatial rotation and scree plot	65
5.4.1	PrinGrid Map plot output	68
5.4.2	PrinGrid Map with Voronoi diagram	69
5.4.3	PrinGrid Map with alternative metrics	70
5.4.4	PrinGrid Map with mixed construct types	74
5.4.5	PrinGrid 3D plot	74
5.4.6	PrinGrid text output	75
5.4.7	PrinGrid analysis of hierarchical data	77
5.5	Crossplot: Plotting elements on constructs as orthogonal axes	78

5.6	Compare: Comparison of grids with some common elements and/or constructs . . .	81
5.6.1	Methodology of grid comparison	81
5.6.2	The compare dialogue	82
5.6.3	Comparing grids with substantial numbers of elements and constructs in com- mon	83
5.6.4	Comparing grids with a substantial number of elements in common	84
5.6.5	Comparing grids with a substantial number of constructs in common	86
5.7	Match analysis: Display matches between elements and between constructs	88
5.7.1	Using ideal elements derived from classes in a Match analysis	90
5.7.2	Match analysis of Wason's card selection task	92
5.8	Analysis of selected elements and constructs	94
5.9	Analysis of weighted elements and constructs	96
6	Style—managing the font and colour scheme of analyses	100
6.0.1	Colour selection	101
7	Editing and exporting RepGrid output	102
8	RepGrid/WebGrid integration	104
8.1	Transferring grid data from RepGrid to WebGrid	104
8.2	Transferring grid data from WebGrid to RepGrid	108
9	Data Formats	109
9.1	Basic grid format	109
9.2	Spreadsheet grid entry format	112
10	Appendix: Elicit Scripts	114
10.1	Script: Elicit Grid.repscript	114
10.2	Script: Elicit/Main.repscript	115
10.3	Script: Elicit/Elements.repscript	118
10.4	Script: Elicit/EditElement.repscript	120
10.5	Script: Elicit/Constructs.repscript	122
10.6	Script: Elicit/AddConstruct.repscript	123
10.7	Script: Elicit/Match.repscript	126
11	Bibliography	129

1 Conceptual Grids

To support the application of his *personal construct psychology*, George Kelly (1955) developed a method for eliciting a person, or group's, conceptual framework that he termed a *conceptual grid* (p.301). He described it as providing a *cybernetic model* (p.302) of a person's *psychological space* (p.301)—emphasizing his rejection of the dichotomy between constructivism and behaviorism, one of the many that he explicitly transcended.

Kelly's grid elicitation and analysis techniques enable one to model personal, or social, conceptual frameworks for diverse domains by eliciting the bipolar *constructs* through which people differentiate significant *elements* in a particular domain. His primary interest was clinical psychology and he developed a particular form of the grid to model a clients' models of their immediate social world in which people were significant to them in a variety of roles. He termed this a *role repertory grid*, often abbreviated to *repertory grid* or *repgrid*. In much of the personal construct psychology literature these terms became used more generally to denote Kelly's generic "*conceptual grid*," but his general term is more descriptive.

Kelly's grid method is applicable in any domain and has been used for conceptual modelling across a very wide range of disciplines such as clinical psychology (Leach et al., 2001), psychiatry (Kirkcaldy et al., 1993), education (Pope and Denicolo, 2001), management studies (Tan et al., 2009; Bellman, 2012; Rad et al., 2013), consumer studies (Mireaux et al., 2007), architectural studies (Tofan et al., 2014), and software engineering (Young et al., 2005) including requirements elicitation (Shaw and Gaines, 1996a; Dey and Lee, 2017).

1.1 RepGrid, WebGrid and RepGrids

The *RepGrid*, *WebGrid*, and *RepGrids* tools provide the capability to elicit, enter, edit and analyze conceptual grid data, and to reflect back the underlying conceptual representations in graphic form. They can be scripted to offer interactive dialogs and analyses, and includes scripts for Shaw's (1980) conversational elicitation, and for the entry of grids that have been elicited through interviews and other methods (Jankowicz, 2004; Fransella et al., 2004; Fromm, 2004; Caputi, 2011). The analyses present grids in a way that reflects their *meaning* in order to promote discussion, understanding, decision-making, conflict mediation, and further elicitation.

In the 1980s Kelly's conceptual grids became recognized as powerful tool for eliciting knowledge from domain experts in order to develop computer-based *expert systems* and were enhanced in various ways to support such applications. Most of the enhancements are consistent with Kelly's (1955) exposition of personal construct psychology and, in particular, they may be used to exemplify his *fundamental postulate*, that psychological processes are driven by the need to *anticipate* events.

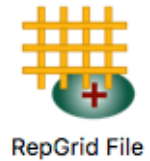
RepGrid includes such enhancements, notably a wider range of data types for grids, the capability to collect and store additional data as part of a grid file, and the ability to use the conceptual structures in grids to *classify* elements in a variety of ways through *classes* defined in terms of Kelly's (1955, p.121) "*intersect of properties*" and having subordination and preference relations to one another.

These enhancements allow RepGrid to be used for teaching and research in *cognitive science*, *artificial science* and *knowledge management*. However, the user interface has been designed such that the new capabilities are not intrusive and the program can be used as a conventional grid elicitation and analysis tool without involving them. For researchers in *personal construct psychology*, the enhancements also enable Kelly's (1955, p.121) notions of how people anticipate events to be explored.

RepGrid is designed to be *open architecture* through its scripting capabilities which enable its functionality to be used by other programs, and its own capabilities to be modified and extended by researchers. Like the other Rep Plus tools it incorporates the *RepScript* tool and provides programmable access to its own functionality, statistical techniques, the graphic capabilities of *RepNet*, and so on.

2 Opening, saving and creating grid files

An orange *grid* icon shown on the right is used for a RepGrid file. It contains lists of the elements and constructs, ratings of elements on constructs, class specifications, analysis parameters and numerous ancillary items of information such as the purpose of the elicitation, name of elicitation, date and time of elicitation, and so on. It may contain user-defined data specific to a particular study, and system-defined data relating to parameters chosen, window size and position, and so on, supporting RepGrid and WebGrid operation. The data is held in a text format that is both human and machine readable.



2.1 Opening a grid file

Clicking on the *Open* button in the *RepGrid* panel of the *Rep Plus Manager* window (Figure 1) brings up a dialog for opening a file, with grid files indicated by their distinctive icons. If a file is selected RepGrid will attempt to open it as a grid, reporting an error if the data cannot be interpreted as such. RepGrid can decode grids files in most of the many formats that have been used by past and current generations of grid programs (including a WebGrid page saved as the HTML source), but those it saves are not necessarily backwards compatible.

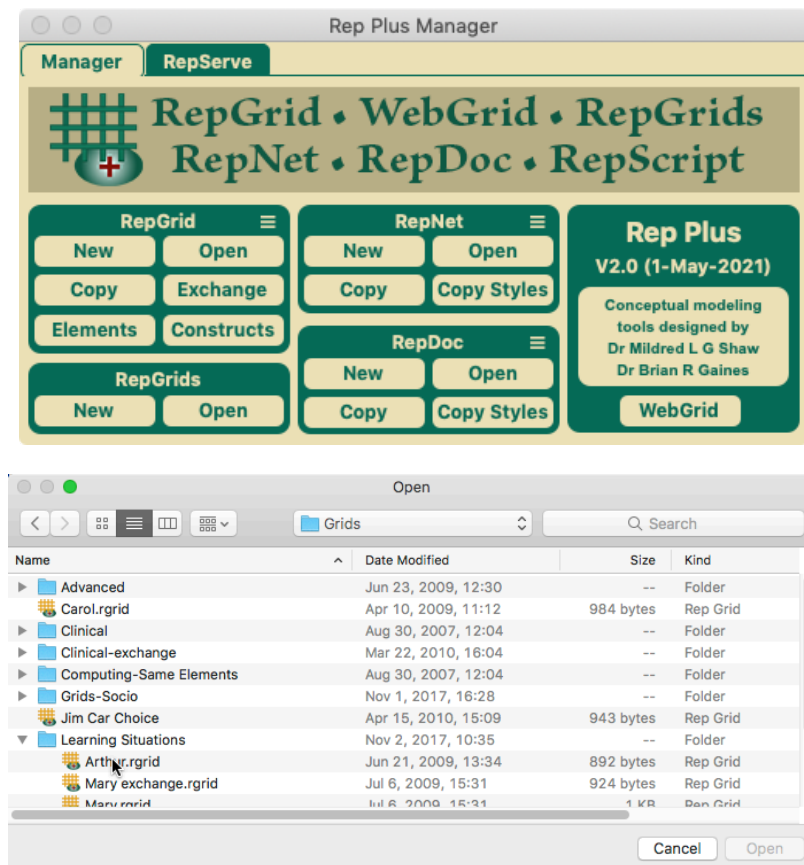


Figure 1: Opening a grid file from the Manager Window

All Rep Plus files contain a file type identifier and grid files may also be opened by using the *Open* (command-O) or *Open Recent* commands in the *File* menu, double-clicking on a grid file, or dragging it to the Rep Plus application icon or Manager window.

2.2 Save, Save As

Selecting the *Save* option in the *File* menu or keying command-S, may be used to save the grid data to its existing file. The *Save As* option may be used to save the grid to a new file.

When a grid is saved the analysis parameters last used and the states of the analysis tools and scripts menus are saved with it. They are restored when the grid is re-opened so that if particular analyses or scripts are being used for that grid they do not have to be set up each time it is opened. The RepGrid window size and position is also saved and restored, as are the column widths of the various tables of grid data.

2.3 Undo

When grid data is changed the state of the grid before the change is recoded, the *Undo* item in the *Edit* menu becomes active, and selecting it or keying command-Z restores the state of the grid before change. The last fifty state changes since the grid was opened are recorded supporting multiple undoing. This information is discarded when the grid is closed.

2.4 New

Selecting the *Grid* option in the *New* submenu of the *File* menu, or keying command-N, creates a new grid with standard default values. Clicking in the *New* button in the *RepGrid* area of the *Rep Plus Manager* window has the same effect

The grid that is created is a copy of the current default grid stored in the *GridScripts* directory in the *RepPlus* directory in the *Documents* or *MyDocuments* directory, or in the *GridScripts* directory in the same directory as the *Rep Plus* application where a default grid is preinstalled.

Users may store as many potential default grids as they wish in the *GridScripts* directory in the *RepPlus* directory in the *Documents* or *MyDocuments* directory. This enables a user to set up a default grid with different parameter defaults, such as terms for elements and constructs, analysis parameters and graphic styles, or even default initial elements and constructs. They can select which one is to be copied when a new grid is created by clicking in the *Menu* symbol (≡) in the *RepGrid* panel of the *Manager* window (Figure 2).

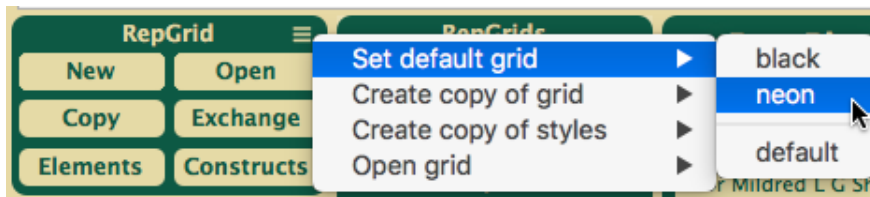


Figure 2: Setting the default grid to be copied when creating a new grid

The hierarchical popup menu shows the available grids in the submenu on the right. The grids *black* and *neon* are user-developed for purposes of presentation. The grid *default* is system installed and designed for use in publications that will be published in colour but often printed in monochrome.

The main menu on the left provides three options:

- Set default grid** Select which of the grids on the right is the default to be copied when a new grid is created;
- Create copy of grid** create a copy of the selected grid on the right—a useful alternative to changing the default grid when only one new grid of a different type is required;
- Create copy of styles** create a copy of the selected grid on the right containing only the styles—useful if some default grids contain elements and/or constructs;
- Open grid** Open the selected grid for editing—making it simple to access the possible default grids in order to make changes to them.

Users do not need to access these capabilities and can simply use the *New Grid* command in the *File* menu or *Manager* window to create an empty grid that is a copy of the system-installed default grid. However, in complex projects the capability to switch between default grids may be very useful to help organize the workflow and help to ensure uniformity in the styles of presentations.

2.5 Copy, Exchange, Elements, Constructs—Open dialogue or drag and drop

It is often required to open a copy of a grid with only the elements or constructs or with both but unrated. These options are available through four buttons in the *RepGrid* panel of the *Manager* window:

- Copy** Open a copy of a grid;
- Exchange** Open a copy with the ratings all set to be open;
- Elements** Open a copy with only the elements;
- Constructs** Open a copy with only the constructs.

If the button is clicked then a file selection dialogue appears that may be used to specify the grid. Alternatively, a grid file may be dragged to the button (Figure 3)..

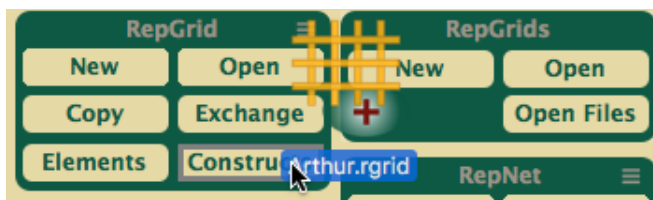


Figure 3: Dragging a grid file to one of the buttons in the *RepGrid* panel

Drag and drop is also effective with the *New* and *Open* buttons, in the first case opening a copy and in the second the original grid.

3 Editing a grid

Clicking on the file named *Arthur* in the file open dialog opens that grid file in the *Options* tab of a RepGrid window (Figure 4).

The screenshot shows the RepGrid Options pane for a grid named 'Arthur'. The window has a title bar with the name 'Arthur' and standard macOS window controls. Below the title bar is a tab ribbon with six tabs: 'Options' (selected), 'Elements', 'Constructs', 'Classes', 'Items', and 'Scripts'. The 'Options' tab is active, showing the following sections:

- Grid description:** Includes text fields for 'Name' (Arthur), 'Note' (empty), and 'Purpose' (exploring the nature of learning situations).
- Grid terminology and defaults:** Includes text fields for 'Element' (situation), 'Elements' (situations), 'Construct' (quality), and 'Constructs' (qualities). It also has a 'Default rating scale' from 1 to 5.
- Types:** A row of checkboxes for 'Ratings', 'Categories', 'Integers', and 'Numbers', all of which are currently unchecked.
- Metavalues:** A row of checkboxes for '? Open', '! Unknown', '* Any', '^ None', and '~ Inapplicable', all of which are currently unchecked.
- Grid annotation:** A large text area containing the text 'after class discussion'.

At the bottom of the pane, there is a 'WebGrid' button with a hamburger menu icon, followed by two rows of buttons: 'Compare', 'Match', 'Crossplot', 'PrinGrid Map' in the first row, and 'Style', 'Synopsis', 'Display', 'Focus Cluster' in the second row. All buttons have a small square icon to their right.

Figure 4: Opening a grid file—the RepGrid *Options* pane

The tab ribbon along the top lets one to select any one of the panes: *Options*, *Elements*, *Constructs*, *Classes*, *Items*, and *Scripts*. The two rows of buttons at the bottom provide access to a range of grid analyses (§5).

When the data in a RepGrid window is changed one can save the changed data in its original file through the *Save* command (or command-S) in the *File* menu. The *Save As...* command in the *File* menu allows one save the grid in a different file. RepGrid supports multi-level undo through the *Undo* command at the top of the *Edit* menu (or command-Z) which becomes active if grid data is changed. Only changes in grid data are recorded as undoable and, since the grid files also keep track of the window position, the *Save* command is always available. If the *Undo* command is active and one attempts to close a RepGrid window without saving the grid data then a warning is given providing the opportunity to save the data before the window is closed.

The RepGrid window opens with the *Options* pane showing, and clicking on one of the other tabs brings its pane into view. The following sections describe the functions of each pane.

3.1 Options pane

The *Options* pane shows some overall features of the grid and allows them to be entered or edited. It has three panels: *Grid Description*, *Grid Terminology and Defaults*, and *Grid Annotation* (Figure 4).

3.1.1 Grid description

The *Grid Description* panel comprises three items. The *Name* field is intended to identify the person from whom the grid was elicited. This is used to identify the grid in the titles of the analyses.

Since several grids might be elicited from the same person, a *Note* field is provided to allow further identification. The note, and other identifiers such as the date and time when the grid was created, may be specified to be included in parentheses after the name *Arthur* in analyses (§6).

The *Purpose* field is used to express the purpose or context for eliciting the grid. Computer-based elicitation tools use this field to remind the user of the reason why they are developing a grid, and the analysis tools add this to the title to show the purpose or context of the elicitation.

3.1.2 Grid terminology and defaults

The *Grid Terminology and Defaults* panel comprises seven items. In grids the entities being construed are termed *elements* and the dimensions of construing, their perceived characteristics, are termed *constructs*. These are somewhat technical terms, and it is often better for users to substitute more colloquial terms appropriate to the domain or topic under consideration. In this grid Arthur, or the person facilitating the elicitation, has chosen to use the term *situation* for *element* and *quality* for *construct*.

3.1.3 Default rating scale

The default scale for the *Ratings* data type in RepGrid may be set to range over a set of integers from a minimum of 100 to a maximum of +100. Scales of 1 to 5, 7 or 9 are commonly used. A two-valued rating scale of 1 to 2 forces the user to make binary distinctions without ‘shades of gray.’ Using 1 to 3 provides a middle option that allows a user to express neutrality between the two poles of a construct. However, users often prefer to have the further gradations of a 1 to 5, 1 to 7 or 1 to 9 scale available to them. In the literature one will also find a variety of other scales used, such as a -7 to +7 scale or a 1 to 11 scale.

One may change the rating scale as one adds constructs, and thus have constructs with different rating scales in the same grid. RepGrid will analyze such grids correctly since it rescales the constructs to a common range as part of its analyses. However, having different rating scales for different constructs is not a common usage of grids, and we do not recommend that this feature be used except with experienced users when it is appropriate to their project.

3.1.4 Grid annotation

The *Grid Annotation* panel at the bottom provides a multi-line text field to store any annotation that may be useful. It is not shown in any of the analyses and may be used to help keep track of any data relevant to the grid. If there is extensive annotation the window may be enlarged to display it.

3.1.5 Rating scale data types: ratings, categories, integers, numbers

The *Data Types* check boxes control what data types are offered. If none are checked then conventional rating scales only are offered, and advanced features such as construct names and rating scale categories, weights, and so on, are not made available so as to simplify usage. If any type or combination of types is checked then those types are offered together with the more advanced features.

Four data types are currently supported:-

Ratings: integer ratings in the range -100 to +100 with the default scale as specified, where one may specify category labels (including the pole names) for ranges of ratings;

Categories: labeled categories, implemented as a subtype of *Ratings* where the rating scale is derived from the number of categories specified;

Numbers: floating point numbers representing the value along a dimension such as height, weight or time;

Integers: integer numbers, implemented as a subtype of *Numbers* restricted to have no decimal places, a significant distinction conceptually but not technically.

3.1.6 Metavalues in ratings: open, unknown, any, none, inapplicable

RepGrid also supports the analysis of grids having meta-values for some of the ratings. The *Metavalues* check boxes control what meta-values will appear on the rating scale popup menus that are made available to users for grid data entry. The possible meta-values are:-

? **Open:** indicating that a rating has not yet been requested or entered;

* **Any:** indicating that any of the metavalues or values might apply, usually used in the specification of an *ideal element* to indicate that a construct is irrelevant;

^ **Inapplicable:** indicating that the construct is not applicable to the element, perhaps because the element does not fall under the pole of a another construct superordinate to that under consideration;

! **Unknown:** indicating that the construct is applicable but the specific rating is unknown or irrelevant;

~ **None:** indicating that the construct is applicable but none of the available ratings is appropriate, usually only used with *Categories* when a suitable one has not yet been made available.

The single characters such as ? and ! before the names provide a succinct representation of a metavalue when a grid is displayed. The attached description is intended a reminder of the meaning of the metavalue, understandable to those facilitating the use of grids.

For purposes of display to users, both the single character and the description may be edited by the facilitator to be more meaningful to users. The description that appears to end users may be modified by the facilitator to be more meaningful to them, e.g. *Open* might be changed to *Enter a rating*. When the grid is stored the standard character for the metavalue is used, but when it is displayed the alternative specified is used.

The meta-values and rating types are related through a logical structure that may itself be represented as a tiered structure of constructs (Figure 5).

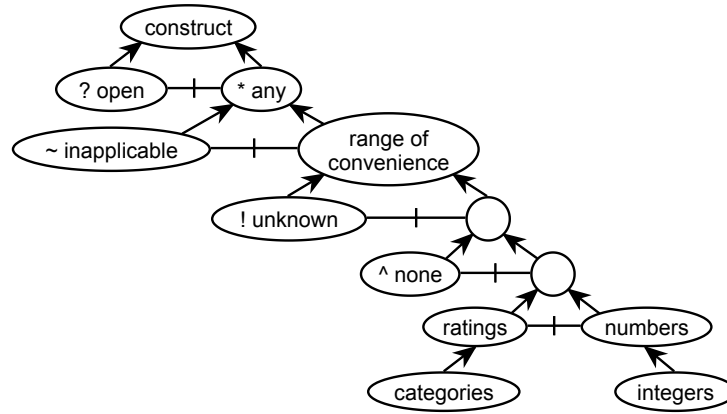


Figure 5: Type hierarchy of meta-values and rating types

The *construct* node at the top represents a partial definition of a construct that is completed through the specification of its *range of convenience* below (supporting Kelly's (1955) notion that constructs with the same name may have differing ranges and foci of convenience). The first three metavalues reflect this structure allowing a construct to be *mentioned* but possibly not *used*: *open* when no data has been entered; *any* when the construct is not relevant to a distinction because it is inapplicable or its value does not matter, typically used in specifying *ideal elements*; and *inapplicable* when the construct is not applicable to an element, a distinction important to representing the ordinal, or *laddered* (Reynolds and Gutman, 1988; Corbridge et al., 1994; Korenini, 2014), structure of constructs in grid form. e.g. in a grid on *holiday resorts*, the construct *poor beaches—good beaches* may be inapplicable to inland resorts but still very significant in comparing waterside resorts.

The final two metavalues indicate that an element is within the range of convenience of a construct but: its rating is *unknown* or, possibly, irrelevant to the purpose of the elicitation; or corresponds to *none* of the ratings, usually categories, made available.

In computing matching scores, *any* matches any value or metavalue, *inapplicable* matches *inapplicable*, and *none* is treated as an alternative middle value in relation to the poles but at the greatest possible distance from the actual middle value of the scale. Otherwise, metavalues match no other value, returning the greatest possible distance as if they were opposite poles.

In most conventional grid elicitation, the only metavalue used is ? to indicate that a rating has not yet been entered. However, sometimes clients are given options such as *any* and *none*, or *both* and *neither*, and RepGrid supports such metavalues.

If metavalues are not used then clients typically use the middle value of a rating scale as a neutral value representing all the possible metavalues above, and the distinction between them is not captured in the grid (Yorke, 1978, 1983). This seems to to be adequate in many situations, but the capability to refine grids with more explicit metavalues is significant when using them to develop computational models of anticipatory processes (Gaines and Shaw, 1993a,b) and to represent hierarchical knowledge structures (Shaw and Gaines, 1998).

3.2 Elements pane

Clicking on the *Elements* tab brings up the elements pane (Figure 6). This lists the numerical order of the elements in the grid in the column on the left, and their names in the next column.

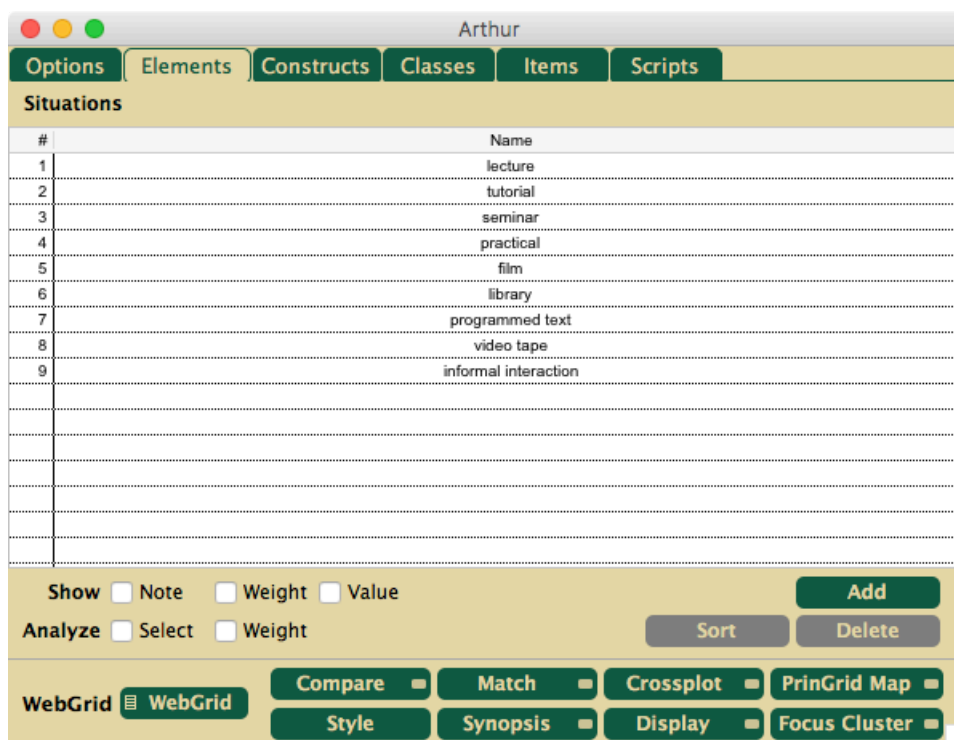
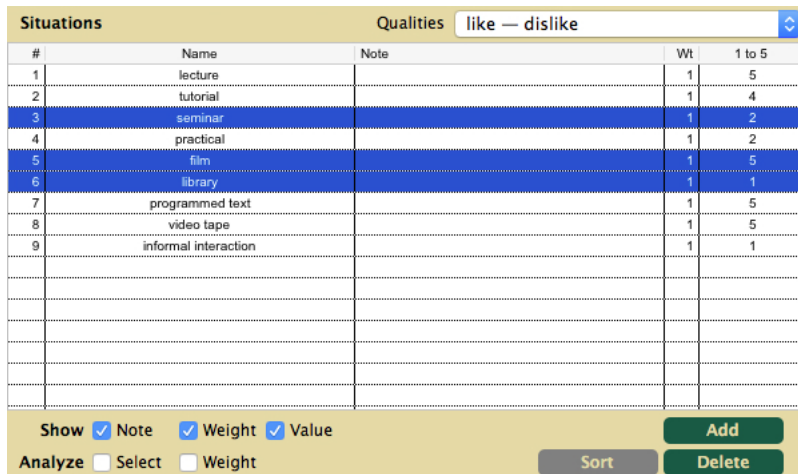


Figure 6: RepGrid *Elements* pane showing element numbers and names

Rows may be selected by clicking in the number column on the left which selects and highlights the row clicked. Multiple rows may be selected by holding down the *shift* key for contiguous selections and the command key for non-contiguous selections, consistent with the normal conventions of the Mac and Windows operating systems. One purpose of selecting rows is to allow them to be deleted by pressing the *delete* or *backspace* key. Others are the selection of only part of a grid for display and analysis (§5.8), and to enable the data in them to be dragged to another application such as a net in RepNet or a document in a word processor.

3.2.1 Element annotation and weights

The element names are always shown, and the associated note, weight and value fields may also be shown dependent on which boxes are checked in the row on the left under the data. Figure 7 shows the *Elements* pane with all the fields activated and some elements selected.

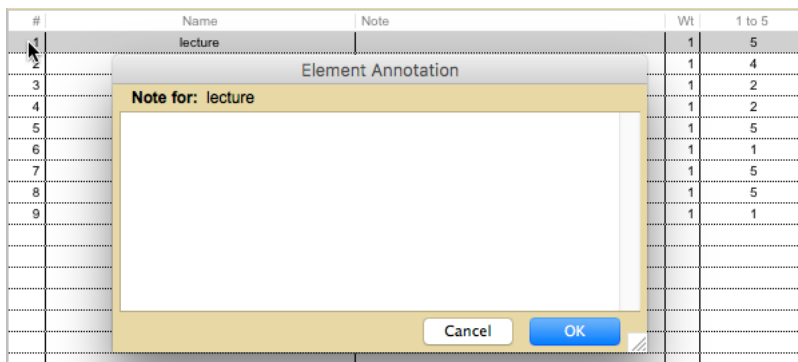


Situations			Qualities like — dislike	
#	Name	Note	Wt	1 to 5
1	lecture		1	5
2	tutorial		1	4
3	seminar		1	2
4	practical		1	2
5	film		1	5
6	library		1	1
7	programmed text		1	5
8	video tape		1	5
9	informal interaction		1	1

Show ☒ Note ☒ Weight ☒ Value
Analyze ☐ Select ☐ Weight Add Sort Delete

Figure 7: RepGrid *Elements* pane with all fields showing

The note field may be used to annotate an element, and the first line may be displayed in the output from analysis. If the annotation in the note field is long it may be edited separately by double-clicking on the element number to open its note field in an edit pane (Figure 8).



#	Name	Note	Wt	1 to 5
1	lecture		1	5
2	tutorial		1	4
3	seminar		1	2
4	practical		1	2
5	film		1	5
6	library		1	1
7	programmed text		1	5
8	video tape		1	5
9	informal interaction		1	1

Element Annotation
Note for: lecture
Cancel OK

Figure 8: RepGrid *Elements* pane note field opened by double-clicking on number field

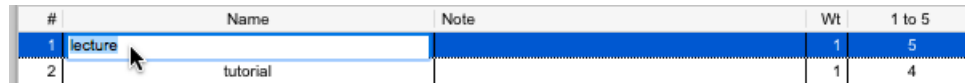
Element annotation is particularly useful if the relevant name is not sufficiently descriptive of the element or the reason for including it, for example, a person referenced by name might have a note *best friend* or *manager*, and one referenced by role might have a note giving their actual name.

The weight value may be used to give some elements more influence than others in various analyses (§5.9). **If integer values are used the effect of weighting is the same as if the element data had been entered the number of times specified in the weight.**

The *Weight* checkbox determines whether the element weights are used in analyses, and the *Select* checkbox determines whether only selected elements are used in analyses.

3.2.2 Editing element data

Any field except the number in the first column and the values in the last column may be edited by clicking on it, which selects it for editing (Figure 9). When the name has been edited, pressing the *return* or *enter* key or clicking elsewhere deselects it for editing and centres the name again.



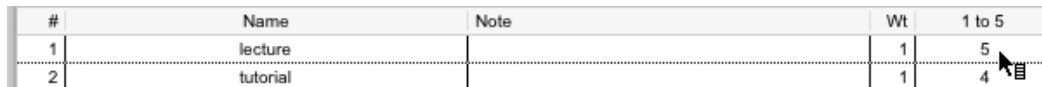
#	Name	Note	Wt	1 to 5
1	lecture		1	5
2	tutorial		1	4

Figure 9: Clicking in a field selects it for editing

Pressing the *tab* key makes next visible field editable, stepping from name to note and weight if these are visible, and then to the name in the next row, creating a new row if at the end of the table. This supports rapid entry of whatever fields are being used for a particular grid. The *Add* button at the bottom right may also be used to add an additional row.

The popup menu of constructs at the top right appears when the *Value* box is checked allows a construct to be selected for which the rating values are provided in the rightmost column. The column header shows the range of possible values.

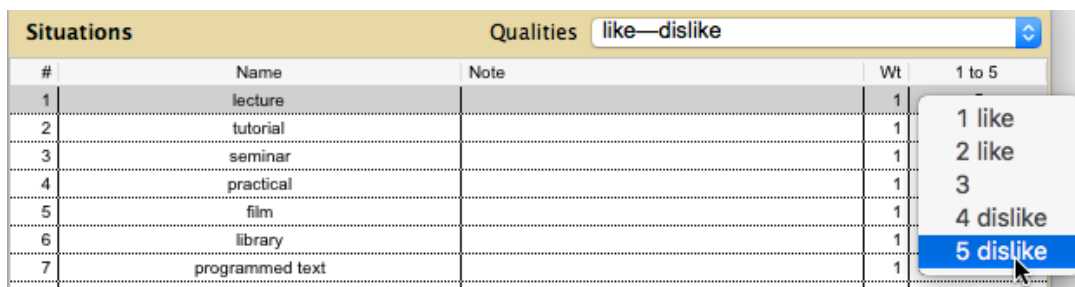
The ratings of the elements on the construct may be entered through a popup menu whose availability is indicated by the *menu cursor* that appears as one mouses over a rating (Figure 10).



#	Name	Note	Wt	1 to 5
1	lecture		1	5
2	tutorial		1	4

Figure 10: Mouse cursor changes to indicate a popup menu is available to edit a rating

Clicking in the rating field activates the popup menu allowing one to select a rating (Figure 11).



Situations		Qualities like—dislike		
#	Name	Note	Wt	1 to 5
1	lecture		1	5
2	tutorial		1	
3	seminar		1	
4	practical		1	
5	film		1	
6	library		1	
7	programmed text		1	

Figure 11: Popup menu editing a rating in the *Elements* pane

Ratings may also be edited as text in the same way as other fields by pressing the *tab* key to make the first rating editable, or by clicking in a rating and releasing the popup menu without selecting a new rating.

When editing the ratings as text, the *tab* key sequences down through the ratings, and has no effect if pressed in the last one.

3.2.3 *Sorting elements*

The list of elements may be sorted by clicking on the element number and dragging it to a different position, or by clicking on a column heading which sorts textual columns alphabetically and numeric ones numerically. Clicking on the heading of an already sorted column reverses the sort order. Figure 12 shows the elements sorted by ratings on the construct *like—dislike* after the ratings column header has been clicked. The elements will remain sorted by ratings if the construct selected is changed.

[illegible]

Figure 12: Sorting elements

Sorting in this way, or by dragging, does not renumber the elements and hence is only temporary. Clicking on the **Sort** button at the lower right makes it permanent and renumbers the elements accordingly.

3.3 Constructs pane

Clicking on the *Constructs* tab in the RepGrid window brings up the constructs pane (Figure 13). This lists the numerical order of the constructs in the grid in the column on the left, their left hand pole (LHP) names in the next column, and their right hand pole (RHP) names in the next column.

Editing constructs in this pane is similar to editing elements as described above. Rows may be selected for deletion with the *delete* or *backspace* key by clicking in the number field. They may be added by clicking the *Add* button or keying *tab* when the last (non-value) field of a construct is editable. The construct pole name fields are always shown, and the associated note, weight, value, level and output fields may also be shown dependent on which boxes are checked in the row under the data. Figure 14 shows the *Constructs* pane with all the fields activated and some constructs selected.

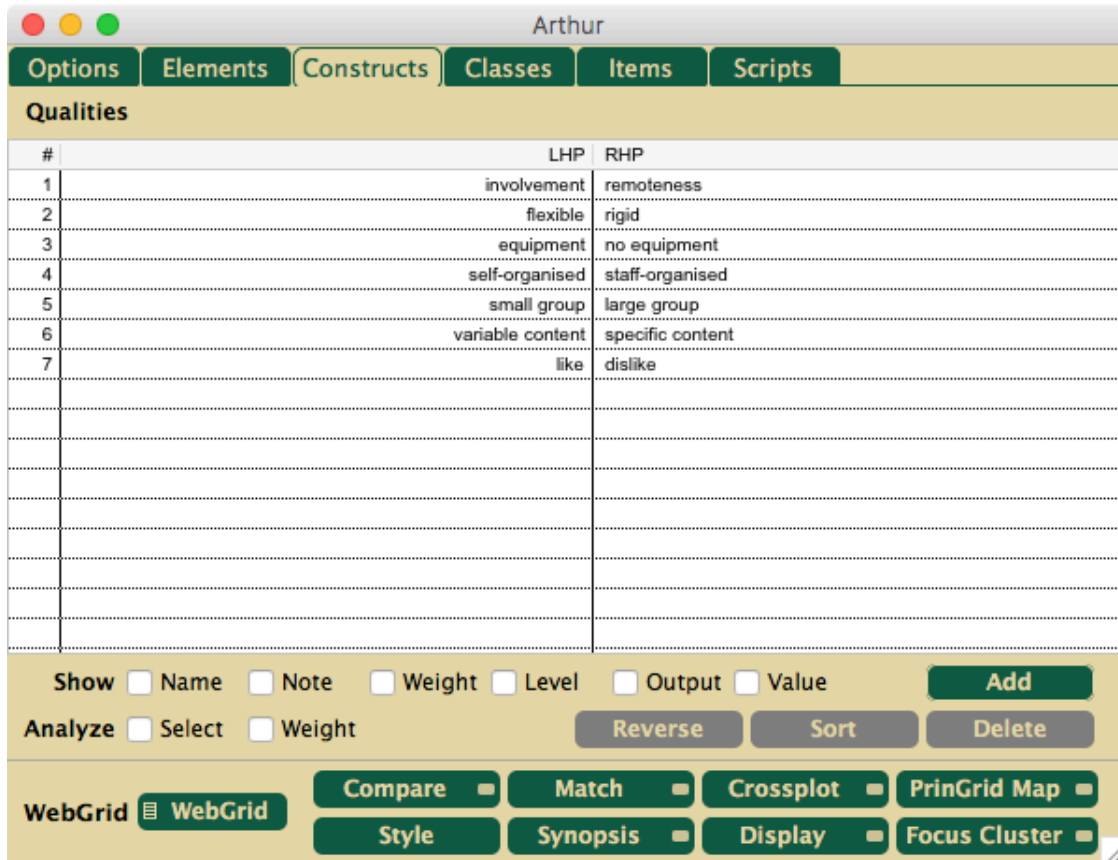


Figure 13: RepGrid *Constructs* pane showing construct numbers and poles

3.3.1 Construct names

Construct *names* may be used a way of naming the range of convenience of the construct separately from that of its poles. When it is displayed in graphic or textual output, the construct name is treated as a noun and the pole name as an adjective qualifying that noun, e.g. *golfer* as construct name and *good—poor* as its a pole names will display as *good golfer—poor golfer*. The name column is placed to the right of the pole columns in the Constructs pane to encourage this usage. In the literature constructs are often named through their pole names, e.g. *flexible—rigid*, and RepGrid defaults to this convention if a name is not supplied.

Construct names are uncommon in normal grid data but are useful in applications where the pole names are insufficient to identify the construct, e.g. with “Q-sort” (Stephenson, 1953) data where the elements are statements, all the constructs are *agree—disagree*, and the name field is used to identify the person responsible for rating that construct.

Construct names can also be used to represent *ordinal relations* between constructs (Kelly, 1955, p.56) by entering a construct name that is the same as a pole name of a superordinate construct (§3.3.6).

[illegible]

Figure 14: RepGrid *Constructs* pane with all fields showing

3.3.2 Construct annotations, weights and reversal

Each construct can be annotated, and the first line of the annotation may be shown in analyses. Lengthier annotation can be edited as described for elements, by double-clicking on the construct number to open its note field in an edit pane.

The weight value enables some constructs to be given greater weight than others in various analyses (§5.9). If integer values are used the effect of weighting is the same as if the construct data had been entered the number of times specified in the weight.

The *Weight* checkbox determines whether the construct weights are used in analyses, and the *Select* checkbox determines whether only selected constructs are used in analyses.

The capabilities of specifying a construct as an *output* or giving it a *level* relative to other constructs are provided to support algorithms for modelling the conceptual structure represented in a grid, such as various forms of machine learning.

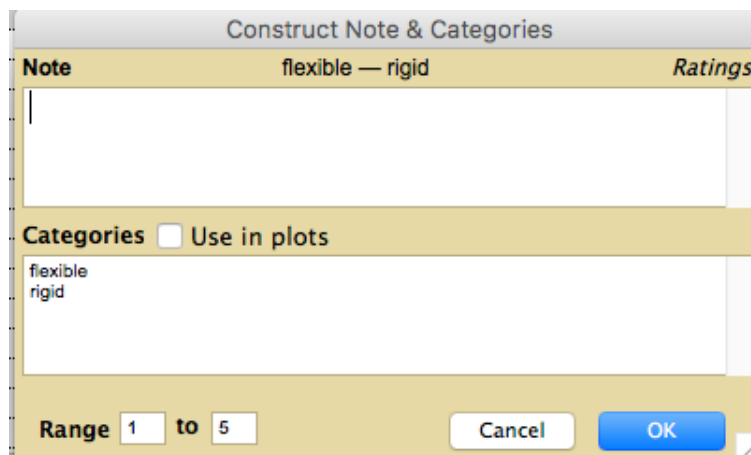
Constructs may be sorted by clicking the column headers or by clicking on the construct number and dragging it to a different position. Clicking on the *Sort* button renumbers the constructs to make the sort order permanent.

Clicking on the *Reverse* button reverses any selected constructs. That is the poles are interchanged and the ratings complemented such that there is no difference in meaning. Such reversal occurs automatically in analyses but it is sometimes appropriate to adjust it in the original grid, for example, to show all the preferred poles on the left or right.

3.3.3 Editing the ratings of elements on constructs

Ratings may be edited through the keyboard or popup menu as for elements, and the *tab* key behaviour is the same as for elements.

Double-clicking on a construct number opens up the note field of a construct for lengthier annotation as it does for an element (Figure 15). It also makes the construct's rating scale range available for editing in case it does not correspond to the default range given in the *Status* pane. RepGrid supports the editing and analysis of grids containing constructs with differing rating scales by rescaling all ratings to the same range (a floating point double in the range -1.0 to +1.0) before analysis. If a rating scale is changed after element ratings have been entered then the ratings are rescaled proportionately along the new scale (note that this rescaling may lose information).



The figure shows a dialog box titled "Construct Note & Categories". It has a yellow background. At the top, there is a "Note" field with a text area and a label "flexible — rigid". Below this is a "Categories" section with a checkbox labeled "Use in plots" and a list box containing the items "flexible" and "rigid". At the bottom, there is a "Range" field with "1" and "5" in input boxes separated by the word "to". To the right of the range field are "Cancel" and "OK" buttons.

Figure 15: RepGrid *Constructs* pane note and categories field for a rating scale

3.3.4 Assigning categories to rating scale ranges

The points, or ranges, on a rating scale may be assigned textual labels termed *categories*. The left and right hand pole names are treated as initial categories (Figure 15) and, if no further categories are allocated, will be assigned non-overlapping ranges from the lowest rating value to the mid-point of the scale, and from the mid-point to the highest rating value, respectively. The mid-point itself will be omitted to avoid overlap if there is an odd number of rating values. This is apparent in the popup menu shown on the right of Figure 11.

The *Categories* text field in the bottom half of the note field editor allows the pole names to be edited and additional categories to be allocated to rating scale values or ranges as shown in the in Figure 16 where a label has been specified for each value of the 5 point scale. These labels are used in the popup menus (Figure 17), and are also available to conceptual modelling algorithms.

More complex specifications of the relations between ratings and categories are possible by specifying an explicit numeric range for a category as one or two numbers separated by a space before the category label as shown in Figure 18. A single number specifies the rating associated with the category, and a pair the range of ratings.

Construct Note & Categories

Note for flexible — rigid Ratings

Categories ☐ Use in plots

flexible
 fairly flexible
 somewhat flexible
 fairly rigid
 rigid

Range to

Figure 16: Specification of intermediate categories

2	flexible	rigid			1	0	<input type="checkbox"/>
3	equipment	no equipment			1	0	<input type="checkbox"/>
4	self-organised	staff-organised			1	0	<input type="checkbox"/>
5	small group	large group			1	0	<input type="checkbox"/>
6	variable content	specific content			1	0	<input type="checkbox"/>
7	like	dislike			1	0	<input type="checkbox"/>

1 flexible
 2 fairly flexible
 3 somewhat flexible
 4 fairly rigid
 5 rigid

Figure 17: Popup menu showing specified values and categories

1 very flexible
 1 3 flexible
 3 5 rigid
 5 very rigid

Figure 18: Specification of overlapping category ranges

The ranges for categories may overlap and, if two or more categories apply to the same rating, all of those that apply are shown separated by commas (Figure 19).

2	very flexible	very rigid			1		
3	equipment	no equipment			5		
4	self-organised	staff-organised			1		
5	small group	large group			3		
6	variable content	specific content			1		
7	like	dislike			1		

1 very flexible, flexible
 2 flexible
 3 flexible, rigid
 4 rigid
 5 very rigid, rigid

Figure 19: Popup menu showing overlapping categories

Bare labels and numerically specified labels may be freely mixed, and the RepGrid labelling algorithm will attempt to achieve what seems to be intended. Any problems with label specification can be made apparent by checking the popup menu it generates. The first and last label are used as the pole names in the plots from the various analyses.

The capability to specify labels indicating the *meanings* intended to be associated with intermediate rating values is particularly useful in communal studies where a grid is developed by a core group

reflecting expected community constructs and provided without ratings to members of the target community to provide their individual ratings. In such studies it may be useful to attempt to promote a common understanding of the use of intermediate values on the rating scales.

For example, the overlapping labelling of a scale illustrated in Figure 19 may be more appropriate for some purposes than one which allocates a different label for each scale point. It indicates that the stronger terms, *very flexible* and *very rigid*, apply only to the extreme of the scale but the more general term, *flexible* and *rigid*, encompasses a range of ratings, and this representation may be closer to common linguistic usage.

The *Use in plots* check box indicates that the category labels rather than the numeric rating values should be used in the output of analyses that show rating values, such as Synopsis, Display and Focus.

3.3.5 Categorical, numeric and integer construct rating scale types


As well as conventional rating scales, RepGrid supports the use of categorical, integer and numeric (floating point) data types in grids (§3.1). If all four construct types are checked in the *Options* pane then the *Constructs* construct *Add* button at the lower right changes to one with menu icon on the left that allows the type of construct to be added to be selected. Figures 20 and 21 illustrate this for a grid about house choices that uses multiple construct types.

House Choice

Options Elements **Constructs** Classes Items Scripts

Qualities Homes **Ideal home**

#	LHP	RHP	Name	Value
1	mall near	long way from stores		1
2	mountain views	town views		1
3	friendly	oppressive		1
4	good study	poor study		1
5	completely remodeled	older style		3
6	needs redecorating	good decorations		5
7	extensive	inadequate	modernization	extensive
8	low	very high	price (000s)	400.00
9	old	recent	year	1999

Show ☒ Name ☐ Note ☐ Weight ☐ Level ☐ Output ☒ Value Add  Ratings

Analyze ☐ Select ☐ Weight


WebGrid  WebGrid

Figure 20: RepGrid *Constructs* pane offering multiple construct types

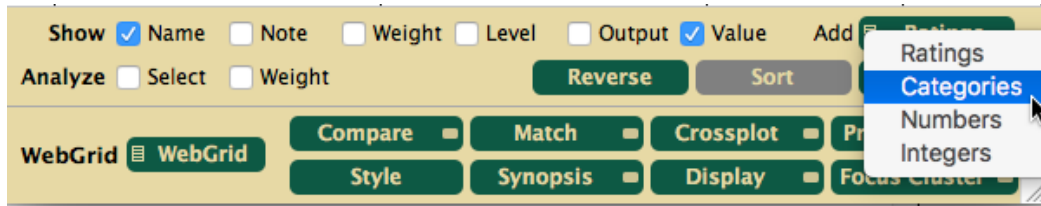


Figure 21: Popup menu for selecting a construct type

In this grid the first six constructs are rating scales and the last three are categories, integers and numbers, respectively. Double-clicking on the seventh construct opens up the note and categories fields for editing (Figure 22).

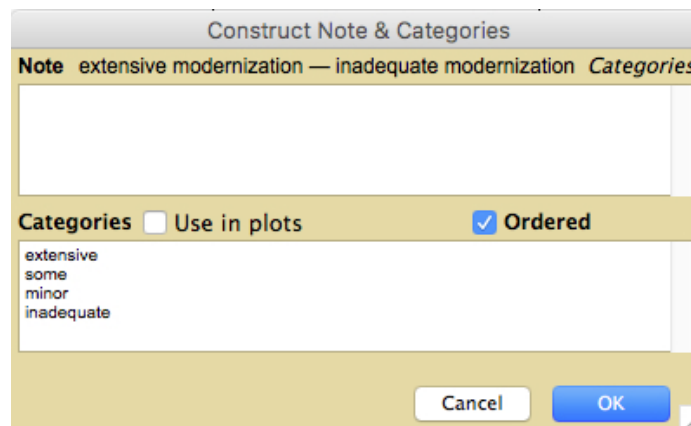


Figure 22: RepGrid *Constructs* pane note and categories fields for a categorical construct

The category names are arbitrary strings entered when the *Category* button is clicked to add a categorical construct. The number of points on the associated rating scale is from 1 to the number of categories entered and the *Use in plots* option is set by default. If it is unset then the categories will be represented by their associated rating values in the output from analyses. In addition to the bare categories that determine the rating scale, numerically specified categories may be entered as for the *Ratings* type.

The ratings of elements on categorical constructs can be entered and changed through the *Value* field, textually or using a popup menu, in the same way as for ratings (Figure 23).

7	extensive	inadequate	modernization	
8	old	recent	year	
9	low	very high	price (000s)	

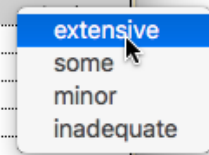


Figure 23: Popup menu showing specified categories

Categories are generally assumed to be ordered so that they may be treated as a conceptual dimension, but unordered categories may also be represented by unchecking the *Ordered* check box.

Some analyses can take this account, but distance-based analyses, such as Focus and PrinGrid, will still treat the categories as ordered.

The category labels can be edited after ratings have been entered, and additional categories can be added without existing ratings being affected. The order of the labels can also be changed without the ratings being affected. **However, if both the labels and the order need to be changed this should be done as two separate edits as, otherwise, the intended outcome is ambiguous.**

If a category label is deleted then any elements rated with that category will have their ratings set to be open.

The *Construct Note and Categories* fields opens up automatically when a new construct, other than ratings, is added in order to allow the category labels and/or the range of the numeric rating scales to be specified.

Double-clicking on the eighth construct, named *Price (000s)*, show that it has numeric values in the range 200.00 to 600.00, and that categories have been assigned to ranges of prices (Figure 24). The precision required is inferred from the maximum number of decimal places specified in the *Range* fields.

Figure 24: RepGrid *Constructs* pane note, range and categories fields for a numeric construct

The *Use in plots* option is unchecked by default but can be checked to indicate that categories rather than numeric values should be used in analyses. The *Bins* field allows the number of bins to be set in the Synopsis analysis histograms if categories are not being used in plots.

The ratings of elements on numeric constructs can be entered and changed through the *Value* field, textually or by using a popup menu, in the same way as for ratings (Figure 25).

The first and last category will be used as the pole names and it is normal for the first to have the same lower bound as the overall range and the last to have the same upper bound as the range.

Double-clicking on the ninth construct, named *Year*, show that it has integer values in the range 1960 to 1999, and that categories have been assigned to ranges of years (Figure 26). Subranges have been allocated category labels as shown and the other features are the same as those for the more general *Numbers* type, e.g. the popup rating menu of Figure 27.

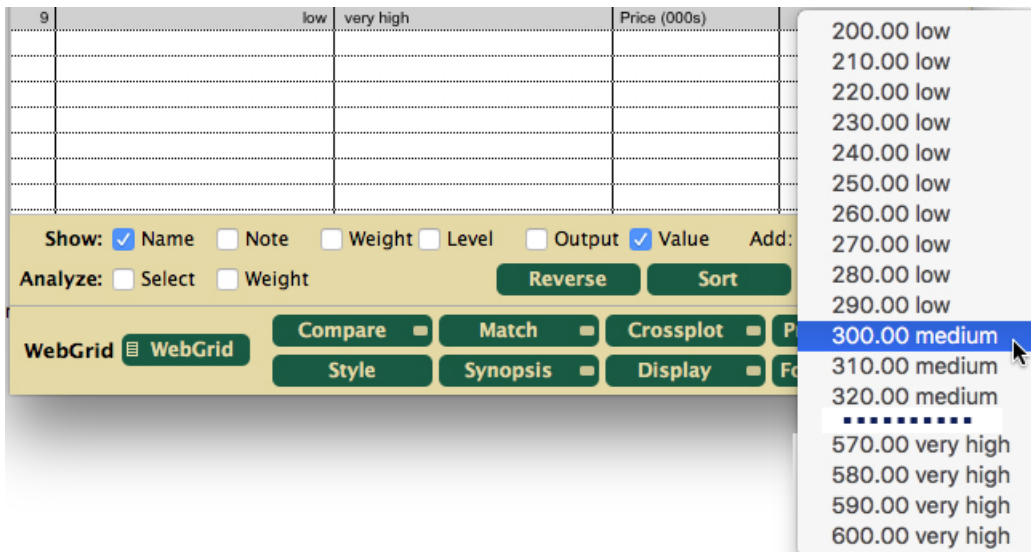


Figure 25: Popup menu (truncated) showing some possible values and categories

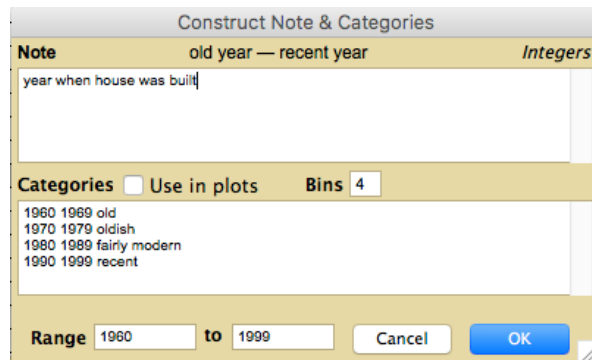


Figure 26: RepGrid *Constructs* pane note, range and categories fields for an integer construct

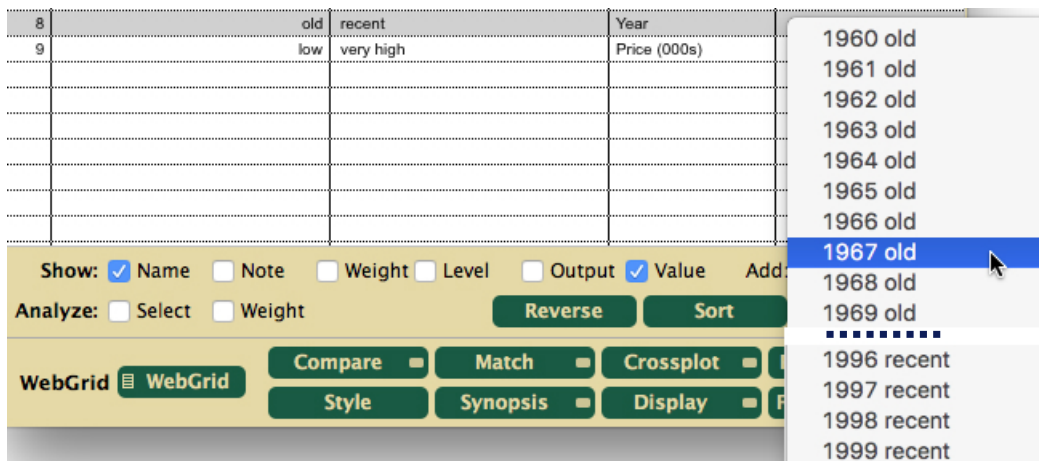


Figure 27: Popup menu (truncated) showing possible values and categories

3.3.6 Representing ordinal relations between constructs

The availability of *metavalues* (§3.1.6) and the capabilities to specify construct *names* (§3.3.1) and *categories* (§3.3.4) combine to enable Kelly's (1955, p.56) *ordinal relations* between constructs to be represented in a grid. Entering a construct name that is the same as a pole name or category of a superordinate construct is recognized as specifying that the first construct is subordinate to the second. The metavalue *~ Inapplicable* may be used to rate a construct as *irrelevant* if it is a subordinate to the opposite pole of one applicable to the element. When elements are *split* into two differently named elements to avoid an inconsistency arising from ordination, the previous name may be represented in a categorical construct so that the linkage between the new elements is still represented in the grid.

Figure 28 illustrates how this may be done for the example discussed in §3.1.6 of the construct *poor beaches—good beaches* being subordinate to the *waterside* pole of the construct *inland—waterside*. At the top are the element names and construct poles and names entered in RepGrid; in the center the ratings of three exemplary elements; and bottom a conceptual net formally derived from the grid. The elements are vacation locations: Arrimoor, construed as an inland town where the *poor beaches—good beaches* construct is inapplicable; Stonyside, construed as a seaside town with poor beaches; and Sandyside, construed as a seaside town with good beaches.

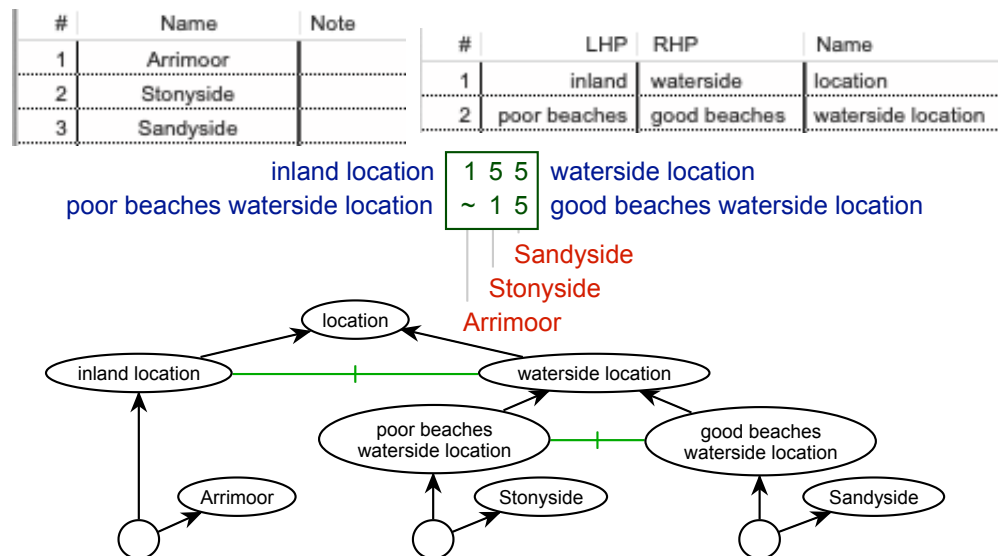


Figure 28: Representing ordinal relations between constructs in a grid

The net at the bottom may be derived from the grid (§3.4.5) and *vice versa*—conceptual nets and conceptual grids provide alternative, but equivalent, representations of conceptual structures (Gaines and Shaw, 2012). The concepts represented by the poles *poor beaches* and *good beaches* are *false* for experiences of Arrimoor because they are *irrelevant*, whereas for those of Stonyside and Sandyside they are relevantly false and true, respectively. The figure illustrates Kelly's (1955) notions that: a grid represents a conceptual “network” (p.304); there is an essential difference between a property being “false” and being “irrelevant” (p.60).

The concepts corresponding to the elements are the anonymous nodes at the bottom of the net with the element names shown underneath them. Each of these nodes represents a constellation of *experiences*, direct and mediated, that have been attributed to an entity identified by the element name. These experiences could each be represented by further anonymous nodes subordinate to the named one, and further nodes subordinate to these. The bottom nodes could be understood as representing *individual* events, or states of affairs, that is, as events which are construed as the *states* of individual entities (Gaines, 2015). All of this is subsumed into the notion of an *element* in a grid.

The element names are, from a logical perspective, arbitrary labels providing a unique identifier for what is construed as an entity, but, for purposes of communication, are usually terms that are generally used to indicate the constellation of experiences associated with the hypothesized entity or descriptive terms sufficient to identify it in the context of their usage. Such considerations address Kelly's (1955) concerns deriving from Korzybski (1951) and Whitehead (1929) that one should recognize that the elements are not concrete entities given by *reality* but are themselves constructed by carving events from experiences and construing certain collections of events to attributable to some entity. The interpretation of the *entity* construct from a realist perspective might be that we evolved in a corner of the universe that naturally partitions into entities in a way relevant to our survival, and from a constructivist perspective that, perhaps as a consequence, we tend to structure our social and built environments in the same way.

The named node, its subordinates and superordinates may be seen as constituting what has come to be termed the *mental file* (Récanati, 2013) that a person creates to group their experiences of an *object* in their environment. The superordinate nodes shown are some of those in the mental file of the person from whom the grid has been elicited that she or he deems relevant to the topic of *selecting a vacation location*.

Ordinal relations between constructs are often elicited by *laddering techniques* (Reynolds and Gutman, 1988; Corbridge et al., 1994; Korenini, 2014). Their explicit representation in the grid supports elicitation techniques that provide feedback to users on the consistency between the ordinal relations that have been elicited by laddering and the rating values they have specified (Korenini, 2014). For example, in figure 28, rating Arrimoor as having poor or good beaches would be inconsistent with it being rated as inland. Inconsistency may indicate error in the data entered, that a particular element is an anomaly that does not conform to normal expectations or that it is heterogeneous with different parts, or aspects, having different characteristics.

If the source of the inconsistency is attributed to the element being somewhat heterogeneous and having multiple significant states then a common technique is to *split* it into two more elements representing these states, for example *Mary as a doctor* and *Mary as a mother*. Kelly's (1955, p.50) *construction corollary* states that "*a person anticipates events by construing their replications*", but the decision to treat one event as a replication of another is a significant abstraction that may itself be problematic.

Korzybski (1951) saw such abstraction of events as a powerful technique essential to the development of human knowledge and civilization, but also cited excessive abstraction as major source of psychological problems. He promoted his *chain-indexing* technique as a therapy for such problems, in which clients were taught to separate events that they attributed to the *same* entity by indexing

the name under which they subsumed the events. Problems “*may become trivial or nonexistent if we become conscious of the identifications involved*” (Korzybski, 1951, p.173).

For example, the inconsistency of Arrimoor being rated as having *poor beaches* when it has also been rated as *inland* might be resolved if it is noted that the town is predominantly remote from water but does have a small district far from the main town that is by the sea. Entering the different locations as two separate elements with notes indicating that one is *Arrimoor town* and the other *Arrimoor seaside* allows the conflict to be resolved (Figure 29). This shows in the grid and net the commonality of the two locations as being in Arrimoor but distinguishes them in terms of the *inland*—*waterside* construct.

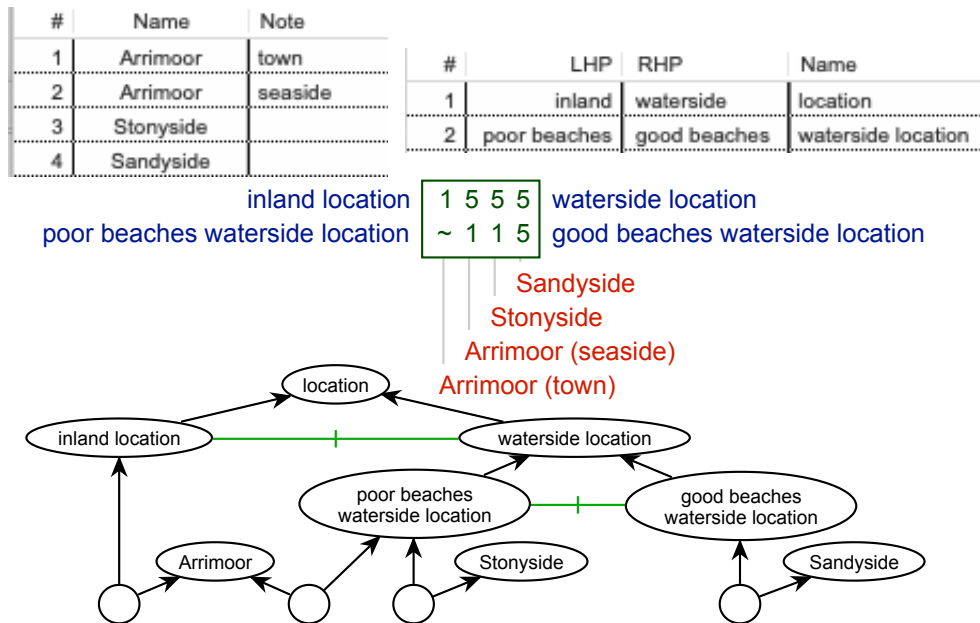


Figure 29: Splitting an element to resolve inconsistency

This example illustrates the techniques whereby structures that represent human knowledge in a way that people find understandable and adequate must be further refined to enable computational reasoning to emulate human reasoning. This process has come to be termed *knowledge engineering*: “The vocabulary initially used by the expert to talk about the domain with a novice is often inadequate for problem-solving; thus the knowledge engineer and expert must work together to extend and refine it.” (Hayes-Roth et al., 1983).

3.4 Classes—intersects and anticipation

Kelly (1955, p.121) explicates the basis of anticipation as the recognition that an event can be construed as falling at the intersect of a set of properties: “What one predicts is not a fully fleshed-out event, but simply the common intersect of a certain set of properties. If an event comes along in which all the properties intersect in the prescribed way, one identifies it as the event he expected.” A class in RepGrid pro-

vides the means to specify an intersect of the properties of elements as encoded in a grid, notably the basic constructs, categories and already defined classes.

This specification may be seen as constituting the *meaning* of the class, and can be interpreted in a variety of ways. For example, classes may be treated as *ideal elements* or as *compound constructs* dependent on their roles in analysis and application (§3.4.7). In *case-based reasoning* (Kolodner, 1993) the ideal elements are regarded as prototypical *cases*, and in *rule-based reasoning* (Buchanan and Shortliffe, 1984) the compound constructs are regarded as the premises of *rules*.

The support of metavalues (§3.1.6) already provides the means to specify a basic intersect as an *ideal element* having ratings on the constructs providing the properties of the intersection, and the metavalue * *Any* on those that are not relevant to it—where *ideal* is understood in the sense of *idealization* rather than *perfection*. An ideal element, such as *ideal house* may well represent a desired anticipation, but it may equally well be an intersect representing an undesired anticipation, such as a rugby player knowing that the intersect of handling the ball and knocking or throwing it forward is a foul play to be avoided. Classes extend the basic specification of an intersect in terms of constructs and ratings to also include specification in terms of relations with other classes, corresponding to one possible anticipation being subordinate to, preferred to, or an exception to, another.

From a personal construct psychology perspective, intersects, ideal elements, cases and rules are all subsumed by, and exemplify, Kelly's (1955, p.8-9) notion of *templates* as patterns that are fitted to events. The basic *constructs* of a conventional grid represent the dimensions of some part of the elicitee's psychological space, the compound constructs that we have termed *classes* represent intersects in that space in terms of the basic constructs and the other classes already defined. Classes may be regarded as more complex specifications of *ideal elements* and also treated as *dichotomous constructs* whose poles specify that the ideal element can *fit* (match closely), or cannot *fit* (match poorly), another element.

Classes were originally introduced in RepGrid to support the development of *knowledge-based* or *expert* systems where Kelly's (1955, p.46) fundamental insight, that construct systems evolved to support the *anticipation* of events, is operationalized by eliciting grids from domain experts and using them to emulate the anticipatory processes underlying human expertise (Boose and Gaines, 1988; Shaw and Gaines, 2005).

3.4.1 Classes Pane

Clicking on the *Classes* tab brings up the classes pane (Figure 30). This lists the numerical order of the classes specified for the grid in the column on the left, followed by the number of elements that could fall under a class, the class names, and their meanings in terms of the constructs, categories and other classes that have already been specified.

Editing classes in this pane is similar to editing elements or constructs as described above. Rows may be selected and sorted, the class number and name are always shown and the count, meaning and note fields may be shown or hidden. The *Add*, *Sort* and *Delete* buttons act as before. The remaining controls and the editing capabilities activated by double-clicking on a class number or description are specific to classes and are detailed below.

#	Cnt	Name	Meaning
1	12	lens feasible	normal tear production
2	12	lens infeasible	reduced tear production and no lens
3	6	soft feasible	lens feasible and not astigmatic
4	6	hard feasible	lens feasible and astigmatic
5	1	exception soft	soft feasible and myope and presbyopic and no lens
6	2	exception hard	hard feasible and hypermetrope and old and no lens
7	5	prescribe soft	soft feasible and soft lens but prefer exception soft
8	4	prescribe hard	hard feasible and hard lens but prefer exception hard

Figure 30: RepGrid *Classes* pane showing class numbers, element counts, names and descriptions

The grid shown represents a well-known expert system dataset devised by Cendrowska (1987) to demonstrate issues with machine learning algorithms in the 1980s, and has been widely studied because the problem is simple enough to be understood but complex enough to exhibit significant aspects of computational knowledge acquisition, representation and inference. The problem is presented as one of contact lens prescription where a hard lens is suitable for an astigmatic client and a soft one otherwise, but there are exceptional conditions making a contact lens unsuitable. One of these is reduced tear production but Cendrowska posits that this should only be tested if the other exceptions do not apply because the test is invasive and expensive.

The dataset is represented in the grid by the 5 constructs and 24 elements that Cendrowska specifies which encompass all possible prescription situations. The elements have been named to include the correct prescription, and the exception cases identified by an asterisk, in order to make it easier to assess the outcome of classification. The classes shown represent a complete solution in a form intended to emulate the expert's decision processes: the initial considerations based on astigmatism; the possible exceptions; and the final prescriptions.

The two prescription classes at the end are the recommendations for action: *prescribe soft* and *prescribe hard*. The implicit recommendation for no prescription, *prescribe none*, applies to the residual elements not covered by these classes. If it was desired to show it explicitly the class specification would be *no lens but prefer prescribe soft or prescribe hard*.

3.4.2 Anticipation: classes as intersects, templets, cases, rules

The *Case* and *Rule* radio buttons at the top right specify whether the classes should be interpreted as cases or rules. The actual class specifications do not change but the way they are described is different. Figure 31 show the classes pane above when the interpretation is changed to rules. The entailments shown for the lens construct arise because that construct has been flagged as an *Output* and hence treated as the conclusion of a rule. The need to be explicit about the separation between premise and conclusions is peculiar to rule interpretations and not required for case-based reasoning.

Classes				Interpretation <input type="radio"/> Case <input checked="" type="radio"/> Rule	
#	Cnt	Name	Meaning		
1	12	lens feasible	normal tear production		
2	12	lens infeasible	reduced tear production and entails no lens		
3	6	soft feasible	lens feasible and not astigmatic		
4	6	hard feasible	lens feasible and astigmatic		
5	1	exception soft	soft feasible and myope and presbyopic and entails no lens		
6	2	exception hard	hard feasible and hypermetrope and old and entails no lens		
7	5	prescribe soft	soft feasible but not exception soft and entails soft lens		
8	4	prescribe hard	hard feasible but not exception hard and entails hard lens		

Figure 31: Classes pane with *Rule* interpretation

3.4.3 Editing class meanings

To enter a new class click on the *Add* button and type in the new class name (Figure 32).

Classes				Interpretation <input checked="" type="radio"/> Case <input type="radio"/> Rule	
#	Cnt	Name	Meaning		
1	12	lens feasible	normal tear production		
2	12	lens infeasible	reduced tear production and no lens		
3	6	soft feasible	lens feasible and not astigmatic		
4	6	hard feasible	lens feasible and astigmatic		
5	1	exception soft	soft feasible and myope and presbyopic and no lens		
6	24	exception hard			

Figure 32: Entering the class *exception hard*

To specify the meaning of the new class, or edit that of an existing one, double-click on the class number or meaning to bring up the *Edit Class dialog* (Figure 33). The first row shows the class being edited; the second a popup menu of classes and associated editing buttons for removing the class shown in the menu from the meaning of the class being edited, or adding it in positive or negative form; the third the constructs; and the fourth the categories associated with the construct above.

Under these areas is a non-editable text area showing the meaning of the class as it is defined, an editable text area where the class may be annotated, and buttons to cancel the class dialog without changing the class meaning, or to update the class meaning to be that entered or edited.

The 'Class Edit' dialog box for the class 'exception hard' contains three rows of input fields and buttons:

Field	Value	Remove	Add	Prefer
Class	lens feasible	Remove	Add	Prefer
Construct	lens	Remove	Add	Not
Category	no	Remove	Add	Not

Below these rows is a 'Note' text area and 'Cancel' and 'Update' buttons at the bottom right.

Figure 33: Editing the meaning of the class *exception hard*

The menu of classes at the top is automatically computed to be those without any dependencies on the class being edited so that entering a dependency cannot lead to a circular specification. This computation is independent of the order of the classes so that they can be reordered as the user wishes. Selecting the class *hard feasible* and clicking on *Add* in the *Class* row adds that class to the specification of the meaning of *exception hard* (Figure 34), that is the class *exception hard* is specified as a *subclass* of *hard feasible* and *inherits* the meaning of that class (*lens feasible and astigmatic*).

The 'Class Edit' dialog box for the class 'exception hard' is shown with the 'Class' field now containing 'lens feasible' and the 'Add' button highlighted. Below the input fields, the text 'hard feasible' is displayed in red.

Figure 34: Entering a dependency on the class *hard feasible*

Similarly, selecting the construct *young—old*, its category *old* and clicking *Add* in the *Category* row makes *old* part of the meaning (Figure 35). This category is defined in the construct to be either *presbyopic* or *pre-presbyopic*.

The 'Class Edit' dialog box for the class 'exception hard' is shown with the 'Construct' field now containing 'young — old' and the 'Category' field now containing 'old'. The 'Add' button for the 'Category' row is highlighted. Below the input fields, the text 'hard feasible and old' is displayed in red.

Figure 35: Entering construct category as part of a description

If *old* had not been specified as category for the construct, one could achieve the same effect in the class description by adding both *presbyopic* and *presbyopic* which be interpreted as the disjunction of the two categories, *pre-presbyopic* or *presbyopic* (Figure 36).

When the meaning of the class has been completely specified, clicking on the *Update* button sets the meaning of the class and closes the dialog.

The screenshot shows the 'Edit Class' interface for the class 'exception hard'. It has three rows: 'Class' with value 'lens feasible', 'Construct' with value 'young — old', and 'Category' with value 'presbyopic'. Each row has a 'Remove' button, an 'Add' button, and a 'Prefer' button. Below the rows, a summary text reads: 'hard feasible and pre-presbyopic or presbyopic'.

Figure 36: Entering a disjunction of construct categories as part of a description

The **Not** buttons add the negation of the class, construct or category, but their use is not recommended for constructs and categories as positive specifications are more natural and easier to understand. However negative links to classes are useful because they allow preferences (for cases) or exceptions (to the premises of rules) to be specified. The name of the button changes between *Prefer* for cases and *Not* for rules to reflect the differing interpretations of negative links between classes.

The removal of a construct is a quick way to remove the construct and associated categories. The capability to add a bare construct or its negation to a specification allows classes to be specified in terms of the relevance or irrelevance of certain constructs. For example, in the *philosophy of art* it has proved difficult to establish an agreed definition of what it is to be an *art object* and there are many competing attempts (Davies, 1991). However, consensus might be achieved about the collection of dimensions along which any art object might be construed (Gaut, 2000), and the relevance of these dimensions to the discussion of any entity might be sufficient to establish that the discussion is about an *art object* (Gaines and Shaw, 2012; Gaines, 2015). The positioning of an art object on certain dimensions would be significant for establishing its category as *primitive art*, *representational art*, *African art*, and so on, but it is the expectation that certain ways of construing will be accepted as relevant that establishes its status as an art object.

3.4.4 Using classes for classification

The **Classify** button outputs classifications based on the classes and constructs. The **Match** text allows the metavalues to be specified which will match anything, usually just ***** but sometimes it is convenient to allow **?** also. The three check boxes after **By** determine whether the classification is by class, by element or by construct. Figure 37 shows elements classified by class for the contact lens grid.

Classification by Classes

- 1: **lens feasible**: soft case 1, hard case 1, soft case 2, hard case 2, soft case 3, hard case 3, soft case 4, none case 9*, none case 11*, hard case 4, soft case 5, none case 15*
- 2: **lens infeasible**: none case 1, none case 2, none case 3, none case 4, none case 5, none case 6, none case 7, none case 8, none case 10, none case 12, none case 13, none case 14
- 3: **soft feasible**: soft case 1, soft case 2, soft case 3, soft case 4, none case 11*, soft case 5
- 4: **hard feasible**: hard case 1, hard case 2, hard case 3, none case 9*, hard case 4, none case 15*
- 5: **exception soft**: none case 11*
- 6: **exception hard**: none case 9*, none case 15*
- 7: **prescribe soft**: soft case 1, soft case 2, soft case 3, soft case 4, soft case 5
- 8: **prescribe hard**: hard case 1, hard case 2, hard case 3, hard case 4

Figure 37: Classification of elements by classes

The *Select* checkboxes for the elements, constructs and classes determine which items are included in the classification, for example, Figure 38 shows the elements classified by the last two classes.

Classification by Classes

7: **prescribe soft**: soft case 1, soft case 2, soft case 3, soft case 4, soft case 5

8: **prescribe hard**: hard case 1, hard case 2, hard case 3, hard case 4

Figure 38: Classification of elements by selected classes

3.4.5 Exporting classes as descriptions, logical expressions and conceptual nets

The *Export List* button at the bottom right of the Classes pane has a menu symbol at the left providing options to export the classes as a textual list, logical expressions, or conceptual nets (Figure 39). In each case, the *Case* and *Rule* radio buttons at the top right specify whether the classes should be interpreted as cases or rules.

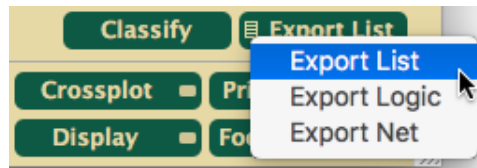


Figure 39: Class export options

Figure 40 shows the class specification exported as a list of cases or of rules.

Cendrowska Descriptions of Classes as Cases

- 1: **lens feasible** *means* contact lens client **and** normal tear production
- 2: **lens infeasible** *means* contact lens client **and** reduced tear production **and** no lens
- 3: **soft feasible** *means* lens feasible **and** not astigmatic
- 4: **hard feasible** *means* lens feasible **and** astigmatic
- 5: **exception soft** *means* soft feasible **and** myope **and** presbyopic **and** no lens
- 6: **exception hard** *means* hard feasible **and** hypermetropic **and** old **and** no lens
- 7: **prescribe soft** *means* soft feasible **and** soft lens **but prefer** exception soft
- 8: **prescribe hard** *means* hard feasible **and** hard lens **but prefer** exception hard

Cendrowska Descriptions of Classes as Rules

- 1: **lens feasible** *means* contact lens client **and** normal tear production
- 2: **lens infeasible** *means* contact lens client **and** reduced tear production **and entails** no lens
- 3: **soft feasible** *means* lens feasible **and** not astigmatic
- 4: **hard feasible** *means* lens feasible **and** astigmatic
- 5: **exception soft** *means* soft feasible **and** myope **and** presbyopic **and entails** no lens
- 6: **exception hard** *means* hard feasible **and** hypermetropic **and** old **and entails** no lens
- 7: **prescribe soft** *means* soft feasible **but not** exception soft **and entails** soft lens
- 8: **prescribe hard** *means* hard feasible **but not** exception hard **and entails** hard lens

Figure 40: Classes listed as cases or rules

Figure 41 shows the class and construct specifications exported as a conceptual net supporting case-based reasoning. A link to the data in the grid is automatically included so the elements may be used to test that the correct anticipations are made. The script *CNet* may be run to provide an inference engine for paraconsistent monadic first-order logic (FOL) and interpret the visual language representing the class and construct specifications as assertions in FOL.

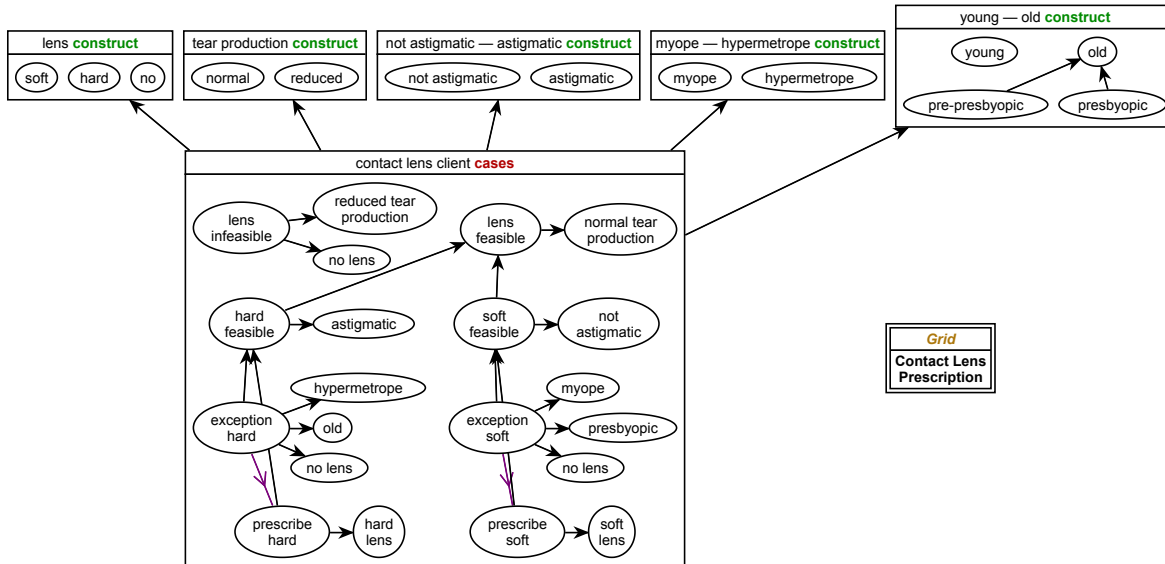


Figure 41: Classes exported as case-based reasoning net

It is difficult to design an automatic layout algorithm that can emulate as perspicuous an arrangement of the conceptual net as can a person, and the initial layout provided by RepGrid can usually be improved manually. Figure 42 shows the net of Figure 41 after manual adjustment to make it more comprehensible and where CNet has been run on a particular case.

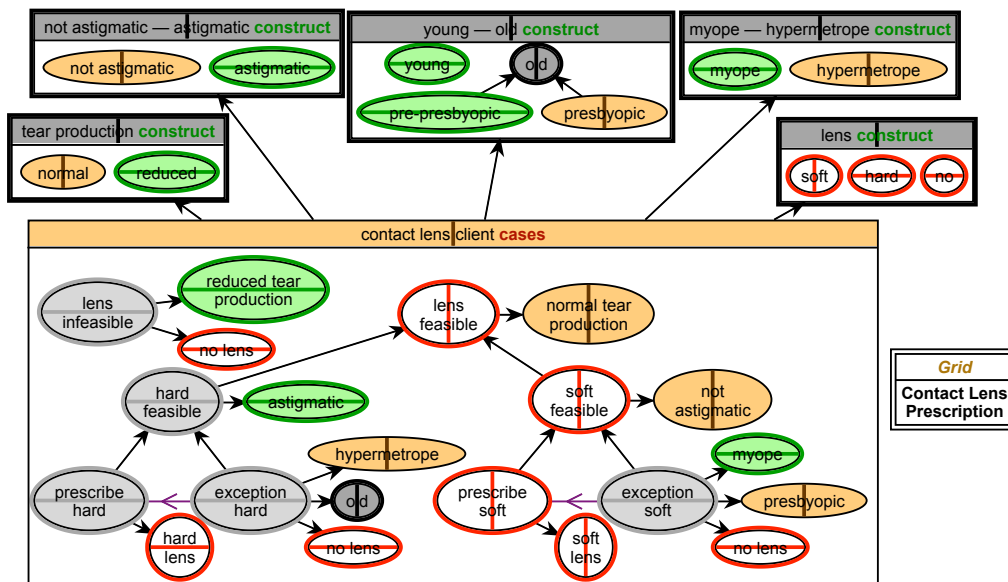


Figure 42: Case-based reasoning net with improved layout—inferences shown

The vertical lines in concepts indicate they are true, horizontal that they are false. The orange background indicates an assertion, green an inference through opposition, grey an inference through irrelevance, and red lines indicate an inference through logical necessity. The *CNet Manual* provides detailed information and further examples.

Figure 43 shows the same class and construct specifications exported as a conceptual net supporting rule-based reasoning. The constructs and grid are not shown as they are the same as Figure 42. CNet may again be used to infer anticipations but based on logical definitions rather than abductive case/prototype-based reasoning that may better model human inference.

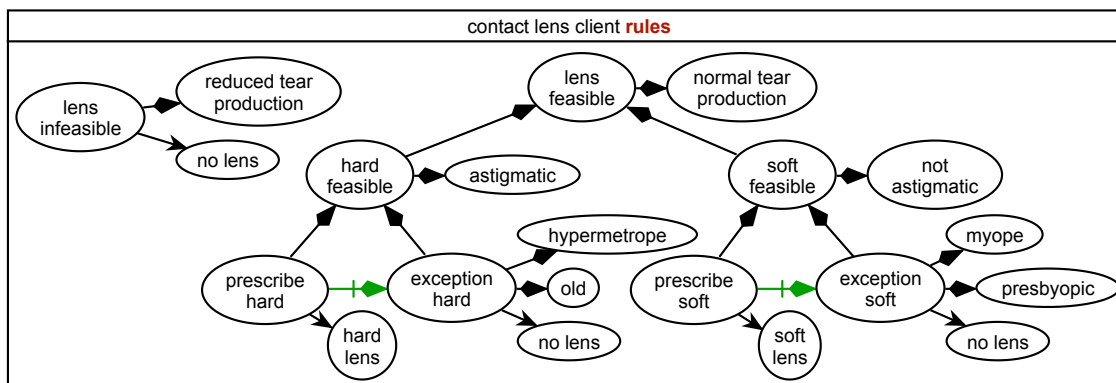


Figure 43: Classes exported as rule-based reasoning net

3.4.6 Using the Classes pane analyses with grids where classes have not been specified

The *Classify* capability of the Classes pane is useful as a way of summarizing a grid even when no classes are defined. Figure 44 shows the classifications by constructs of the grid about learning situations used initially to illustrate editing a grid—the elements falling under each construct pole.

Figure 45 shows the classification by elements of the same grid—the construct pole under which each element falls.

Figure 46 shows the constructs and elements in the *learning situations* grid exported as a conceptual net.

Each element is represented as a state of affairs representing an experience of a learning situation that is expressed in the grid, with the element name as an indicial property indicating the type of the experience.

Running CNet, setting the pole of a construct true, and running abductive inference over the cases allows relations between the constructs to be discovered, for example that *like* always implies *involvement* and that *dislike* always implies *specific content*. These asymmetric relations sometimes, but not always, correspond to the symmetric matches of other analyses, but sometimes provide different insights into the conceptual model implicit in the grid.

Classification by Constructs

- 1.1: **involvement**: seminar, practical, library, programmed text, informal interaction
- 1.2: **remoteness**: lecture, film
- 2.1: **flexible**: practical, library, video tape, informal interaction
- 2.2: **rigid**: lecture, tutorial, film, programmed text
- 3.1: **equipment**: practical, film, programmed text, video tape
- 3.2: **no equipment**: lecture, tutorial, seminar, informal interaction
- 4.1: **self-organised**: library, programmed text, video tape, informal interaction
- 4.2: **staff-organised**: lecture, tutorial, seminar, film
- 5.1: **small group**: tutorial, library, programmed text, video tape
- 5.2: **large group**: lecture, seminar, practical, film
- 6.1: **variable content**: seminar, library, informal interaction
- 6.2: **specific content**: lecture, tutorial, film, programmed text, video tape
- 7.1: **like**: seminar, practical, library, informal interaction
- 7.2: **dislike**: lecture, tutorial, film, programmed text, video tape

Figure 44: Classification of elements by constructs

Classification by Elements

- 1: **lecture**: remoteness, rigid, no equipment, staff-organised, large group, specific content, dislike
- 2: **tutorial**: rigid, no equipment, staff-organised, small group, specific content, dislike
- 3: **seminar**: involvement, no equipment, staff-organised, large group, variable content, like
- 4: **practical**: involvement, flexible, equipment, large group, like
- 5: **film**: remoteness, rigid, equipment, staff-organised, large group, specific content, dislike
- 6: **library**: involvement, flexible, self-organised, small group, variable content, like
- 7: **programmed text**: involvement, rigid, equipment, self-organised, small group, specific content, dislike
- 8: **video tape**: flexible, equipment, self-organised, small group, specific content, dislike
- 9: **informal interaction**: involvement, flexible, no equipment, self-organised, variable content, like

Figure 45: Classification of constructs by elements

3.4.7 Classes as ideal elements or compound constructs in grids

The intersects specifying classes may be also be represented, in part, as *ideal elements* or as *compound constructs*. The subordination relations between classes may be expanded and included in the ideal elements, but the preference relations may not, and are included as notes. Figure 47 shows the contact lens grid displayed with its classes as ideal elements.

The *Analyze* row in the Classes pane may be use to specify that all classes, or selected classes, are including in grids being analyzed. This inclusion is only temporary for the purposes of the analysis. The *Case* and *Rule* checkboxes determine whether the classes are represented by elements or constructs, respectively. The *Add to Grid* button makes it permanent. Further examples are given in the appropriate analysis sections.

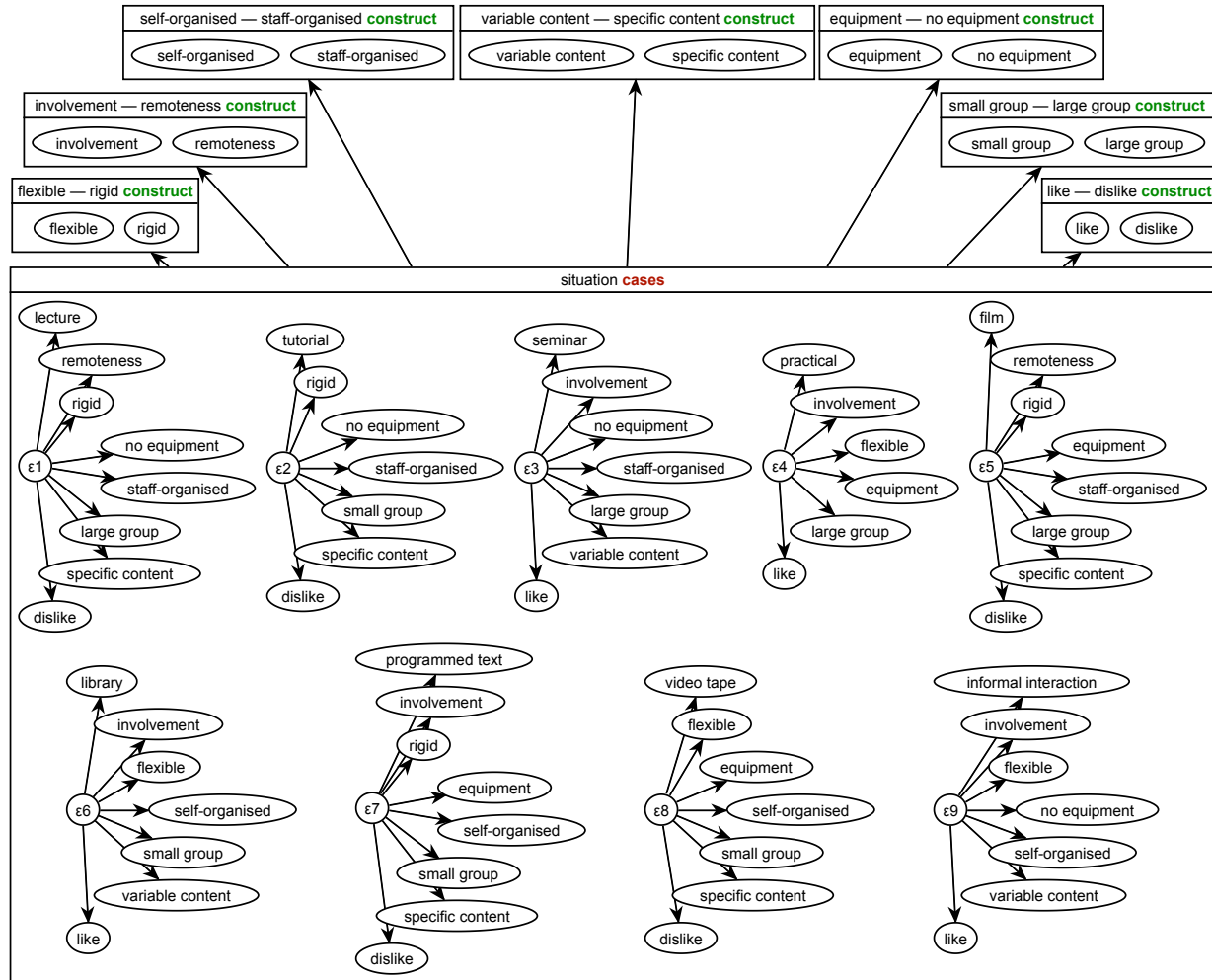


Figure 46: Conceptual net for *learning situations* grid

3.5 Items pane

RepGrid makes provision for additional items of data to be added to a grid, stored with it, and edited. These items may be viewed and edited through the *Items* pane. Clicking on the *Items* tab in the *House Choice* RepGrid window brings up a pane listing the items by name and value (Figure 48).

The first five items shown are automatically generated when a new grid is created: a unique identifier string and the date, time, and location (the IP of the machine on which the grid was created), and the status of the grid indicating whether is new or derived from an existing grid.

The next four items are fields entered in WebGrid to control the styling of the web pages generated in eliciting and analyzing this grid. It is sometimes convenient to edit these fields in RepGrid.

The final three items keep track of the RepGrid window position and size, and the column positions of the tables on the *Elements* and *Constructs* panes.

no	reduced	not astigmatic	myope	young	none case 1 (none)
soft	normal	not astigmatic	myope	young	soft case 1 (soft)
no	reduced	astigmatic	myope	young	none case 2 (none)
hard	normal	astigmatic	myope	young	hard case 1 (hard)
no	reduced	not astigmatic	hypermetrope	young	none case 3 (none)
soft	normal	not astigmatic	hypermetrope	young	soft case 2 (soft)
no	reduced	astigmatic	hypermetrope	young	none case 4 (none)
hard	normal	astigmatic	hypermetrope	young	hard case 2 (hard)
no	reduced	not astigmatic	myope	pre-presbyopic	none case 5 (none)
soft	normal	not astigmatic	myope	pre-presbyopic	soft case 3 (soft)
no	reduced	astigmatic	myope	pre-presbyopic	none case 6 (none)
hard	normal	astigmatic	myope	pre-presbyopic	hard case 3 (hard)
no	reduced	not astigmatic	hypermetrope	pre-presbyopic	none case 7 (none)
soft	normal	not astigmatic	hypermetrope	pre-presbyopic	soft case 4 (soft)
no	reduced	astigmatic	hypermetrope	pre-presbyopic	none case 8 (none)
no	normal	astigmatic	hypermetrope	pre-presbyopic	none case 9* (none)
no	reduced	not astigmatic	myope	presbyopic	none case 10 (none)
no	normal	not astigmatic	myope	presbyopic	none case 11* (none)
no	reduced	astigmatic	myope	presbyopic	none case 12 (none)
hard	normal	astigmatic	myope	presbyopic	hard case 4 (hard)
no	reduced	not astigmatic	hypermetrope	presbyopic	none case 13 (none)
soft	normal	not astigmatic	hypermetrope	presbyopic	soft case 5 (soft)
no	reduced	astigmatic	hypermetrope	presbyopic	none case 14 (none)
no	normal	astigmatic	hypermetrope	presbyopic	none case 15* (none)
no	reduced	*	*	*	lens infeasible
*	normal	*	*	*	lens feasible
*	normal	astigmatic	*	*	hard feasible
*	normal	not astigmatic	*	*	soft feasible
no	normal	astigmatic	hypermetrope	presbyopic	exception hard
no	normal	not astigmatic	myope	presbyopic	exception soft
hard	normal	astigmatic	*	*	prescribe hard (prefer exception hard)
soft	normal	not astigmatic	*	*	prescribe soft (prefer exception soft)

young — old

myope — hypermetrope

not astigmatic — astigmatic

normal tear production — reduced tear production

soft lens — no lens

Figure 47: Grid with classes represented as ideal elements

Items may be deleted by selecting the item row and pressing the *delete* key. Names and values may be edited by clicking in the cells and editing the text. Additional fields may be added by clicking on the *Add* button and entering the item name and value.

Multi-line values may be edited by double-clicking on the item number which brings up a multi-line editor as shown below for item six which defines some styles in the header of a WebGrid page (Figure 49).

3.5.1 User-defined items

You can define your own items to use in storing additional data in a grid file, and it is common for elicitation scripts to collect additional data and store it in this way. RepGrid will not allow such items to have names that correspond to its own reserved names, and to avoid possible clashes user-defined items should commence with an underline, _ character.

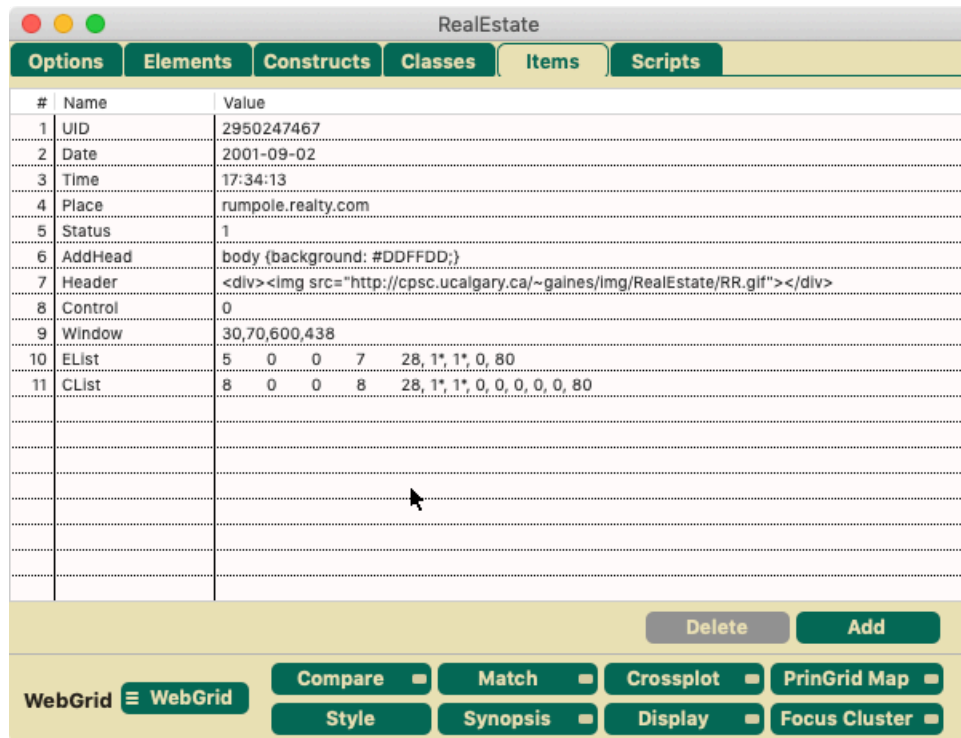


Figure 48: RepGrid *Items* pane showing additional items included with grid data

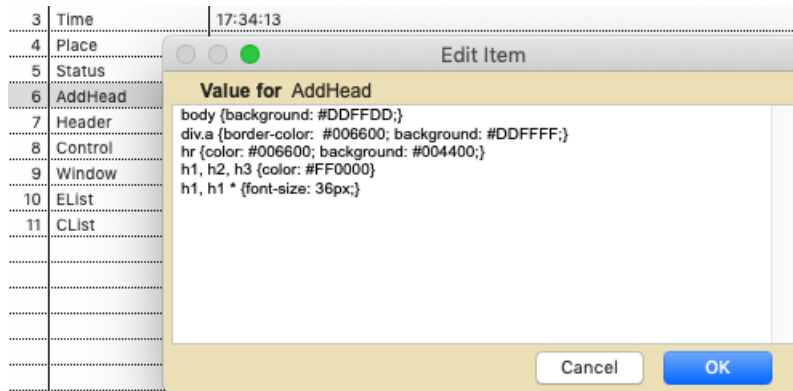


Figure 49: RepGrid *Items* pane editing a multi-line value

3.6 Scripts pane

RepGrid has an open architecture through the capability to run scripts written in *RepScript* that have full access to read and modify the grid data (see RepScript manual). The scripts are text files that users can develop to support their own analyses, such as cognitive complexity measures, or to import and export grid data in other formats.

Clicking on the *Scripts* tab in the RepGrid window brings up the *Scripts* pane that allows such scripts to be run (Figure 50). The script button/popup menu at the top right enables a script to be chosen and run. It is dual-purpose in that clicking in the menu symbol on the left (or right-clicking/CTL-

clicking anywhere in the button) brings up a menu which enables one to select the script named in the button, and clicking in the script name causes the script to run.

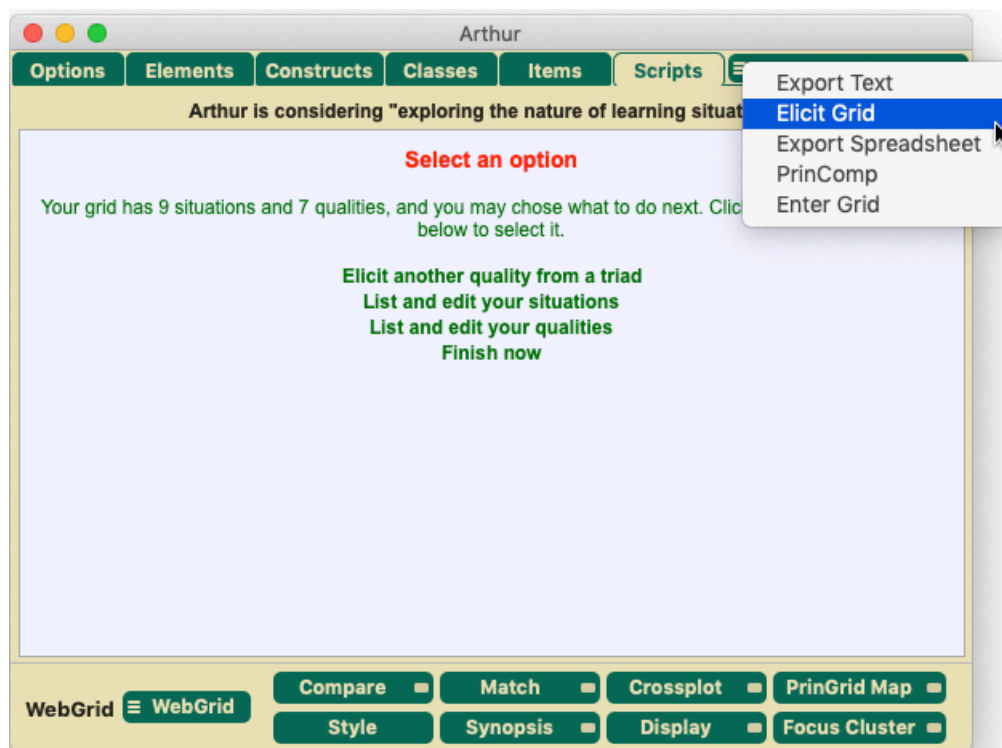


Figure 50: RepGrid *Scripts* pane and scripts menu

Just under the tabs is a line of text that can be set under script control, for example, to indicate the purpose, or status of the script. Under that is an interactive text pane that may be programmed by a script to conduct an interactive dialog, including clickable menus, or used to display status information or results. The pane is shown running the *Elicit Grid* script and offering various elicitation options.

A script may be terminated at any time by pressing the *esc* key.

Scripts have full access to all data in the grid, can access files and interact with users. Hence they may be used to import and export grids in any specified format, enter grids, elicit them interactively, edit them, and so on. Scripts are text files which users may modify and create their own customized scripts. In particular they may translate them into other languages. Rep Plus uses a Unicode text representation so that non-Roman scripts may be used. The display and analysis tools have been designed to use only the text entered into the grid so that they also support graphic output in the language being used.

The initial set of scripts currently supplied with RepGrid include *Enter Grid* for rapid entry of grid data, *Elicit Grid* for conversational elicitation of a grid, and scripts to export grids in various formats. They provide basic facilities for grid entry, elicitation and export, and also serve as exemplars for those developing custom scripts.

The supplied scripts are within the Rep Plus application directory *GridScripts*, and the name of the script is what appears on the popup menu. User scripts may be put in the *GridScripts* directory in the *Rep Plus* directory created automatically in the *Documents* or *MyDocuments* directory. If certain scripts are intended for use only with certain types of grids they may be placed in a *GridScripts* within the directory containing the scripts, and will only appear in the scripts menu of those grids.

4 Grid entry, elicitation and export scripts

The scripts supplied with RepGrid accessible through the *Scripts* pane are designed to support rapid entry of existing grid data, conversational elicitation of new grids, and export of grids in other formats.

4.1 Enter Grid

The *Enter Grid* script allows existing grid data to be entered rapidly. The script is programmed to:-

- Request missing fields in the *Status* window such as the user's name and the purpose of the elicitation
- Ask the user to enter the elements
- Ask the user to enter the constructs and their ratings on the elements
- Offer the user the option to edit and enter more elements or constructs, or to finish the entry process

The figures below illustrate the *Enter Grid* script being used to enter the *Arthur* grid used to illustrate this manual. The first screen shows the name and purpose being entered as text terminated by the *return* key (Figure 51).

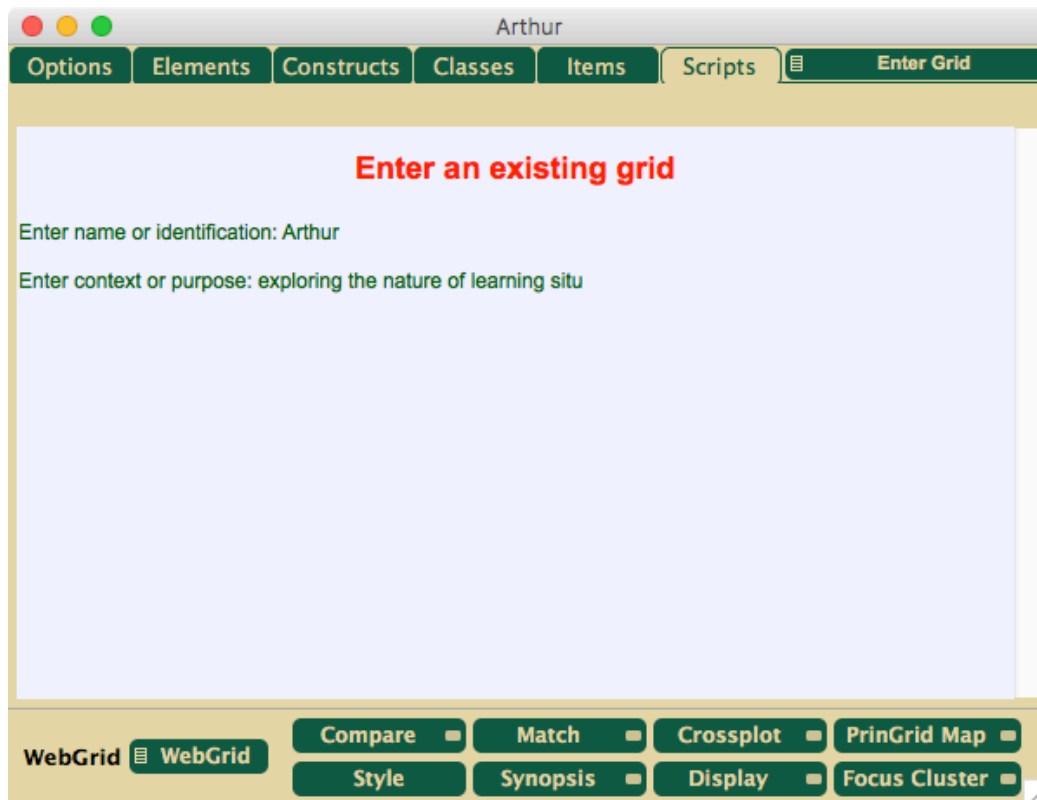


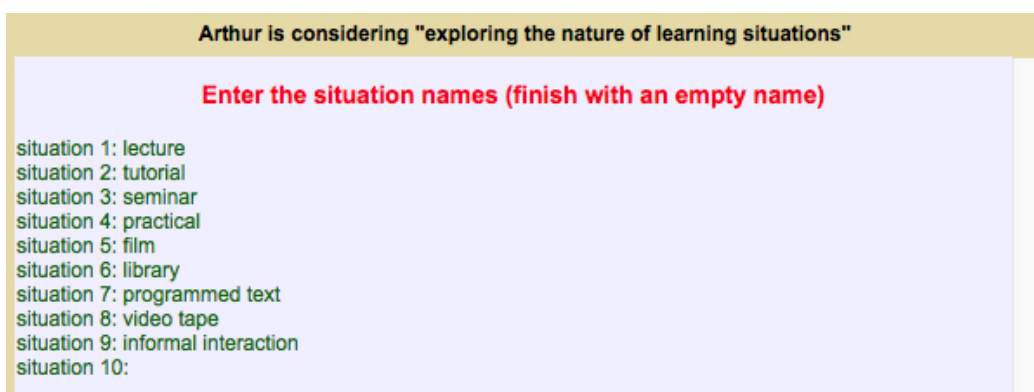
Figure 51: *Enter Grid* script entering name and purpose

The entry process is similar to that of a text editor. The *delete* key may be used to erase text, and text may be selected, copied and pasted.

The *Enter Grid* script may be run immediately with a new grid, or with one where the name and purpose have been entered on the *Status* pane, and/or the elements have been entered on the *Elements* pane. **If the rating scale is other than 1 to 5 or the terms for elements and constructs are different then these should be set up in the *Status* pane before the script is run.** In the example below elements have been termed *situations* and constructs *qualities*.

The script offers the facility to edit elements, constructs and ratings, and this may also be done through the *Elements* and *Constructs* panes. The *Display* button provides a convenient way of checking the data entered against its source to ensure that it is correct.

After the purpose has been entered, the element names are entered (Figure 52). Keying *return* with an empty name terminates the element entry and proceeds to the constructs and ratings entry.



Arthur is considering "exploring the nature of learning situations"

Enter the situation names (finish with an empty name)

situation 1: lecture
situation 2: tutorial
situation 3: seminar
situation 4: practical
situation 5: film
situation 6: library
situation 7: programmed text
situation 8: video tape
situation 9: informal interaction
situation 10:

Figure 52: *Enter Grid* script entering element names

Figure 53 shows the pole names of the first construct being entered. Keying *return* after the right pole name proceeds to the ratings entry for that construct.



Arthur is considering "exploring the nature of learning situations"

Enter the quality poles and ratings (finish with an empty pole name)

Left pole of quality 1 (rated 1): involvement
Right pole of quality 1 (rated 5): remoteness

Figure 53: *Enter Grid* script entering pole names of the first construct

Then the ratings on each element are requested (Figure 54). They can be typed in as numbers followed by *return*, or by clicking the mouse and selecting a rating from a popup menu.

When all the elements have been rated on the first construct, the pole names of the second construct are requested, and so on, until all the constructs and ratings have been entered. Construct entry is terminated when an empty pole name is entered, and the grid editing options are listed (Figure 55).

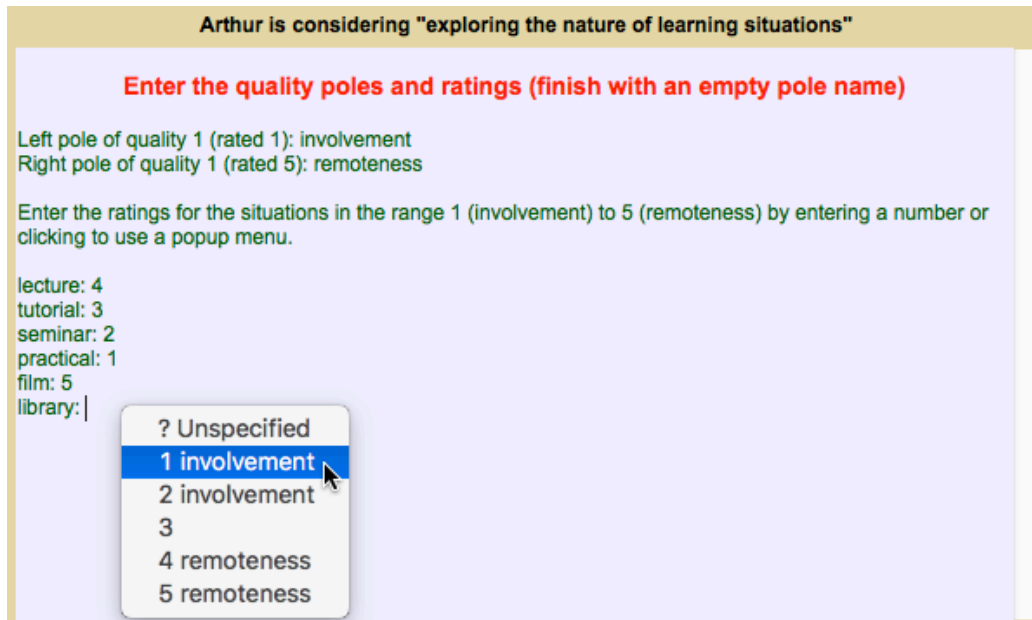


Figure 54: *Enter Grid* script entering pole names of the first construct

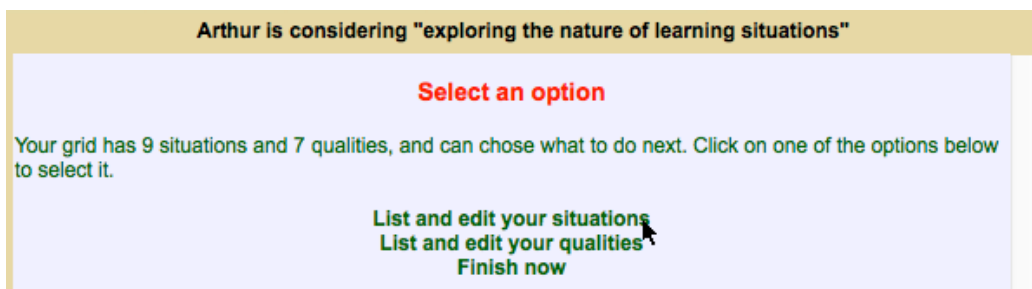


Figure 55: *Enter Grid* script options after grid entry is complete

The *Display* button may be used to display the grid to check that it has been entered correctly. If not, one the list of options displayed may be clicked to list the elements or constructs may be listed and allow the names and ratings to be edited. Clicking on *List and edit your situations* shows the elements and provides options to add more or to edit an element name and ratings (Figure 56).

Clicking on the element *practical* displays its name and its rating on each construct (Figure 57).

Clicking on the option to delete the element displays a warning and options to do so or cancel (Figure 58).

Clicking on the element name makes it available for editing as shown below (Figure 59).

Clicking on a construct makes the rating on the selected element available for editing (Figure 60).

Similarly, the option in Figure 55 to list and edit the constructs may be chosen. They are then listed for editing (Figure 61).

Clicking on a construct displays it for editing (Figure 62). The editing procedures for the pole names and the ratings are essentially the same as those described above for the elements.

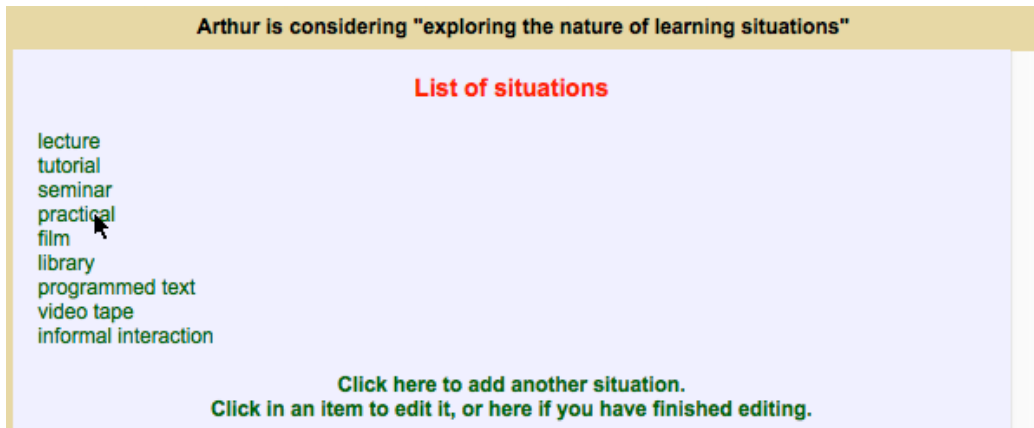


Figure 56: *Enter Grid* script listing elements

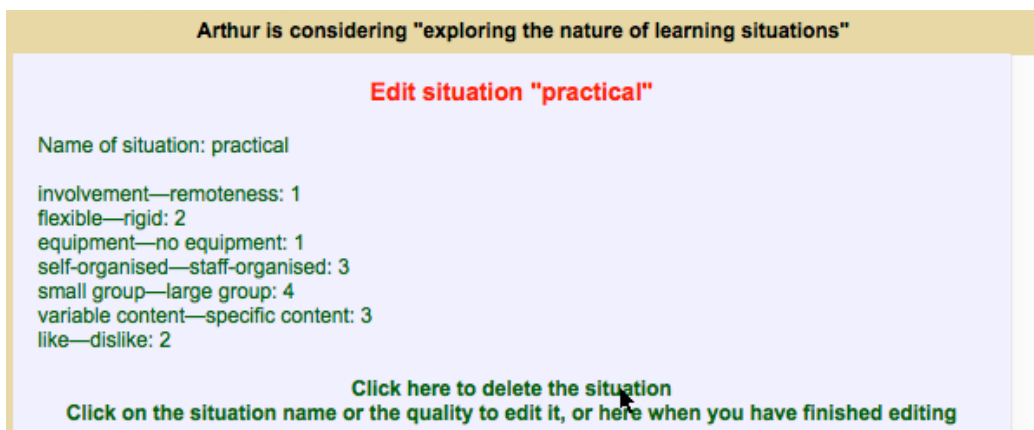


Figure 57: *Enter Grid* script editing a selected element

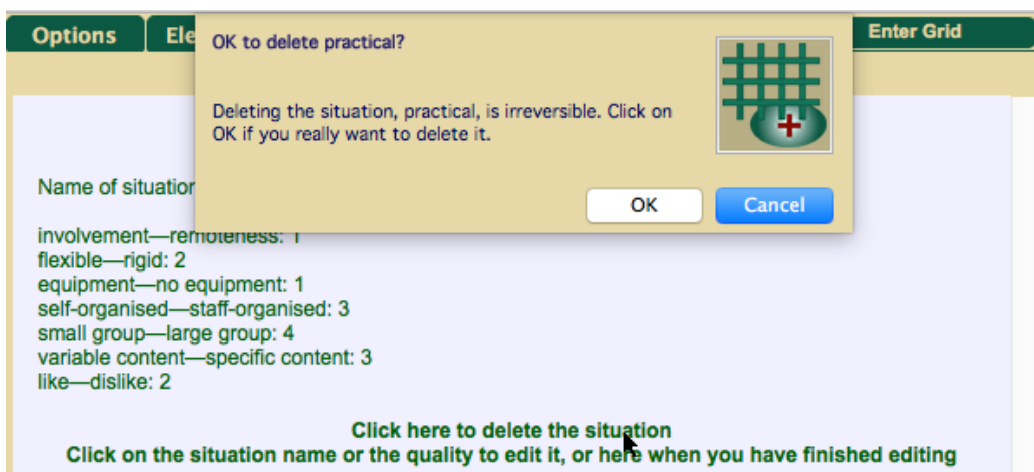


Figure 58: *Enter Grid* script deleting the selected element

The *Enter Grid* script may be run at any time with an existing grid and will open with the editing options of Figure 55 enabling the grid to be edited conversationally as described above as an alterna-

Arthur is considering "exploring the nature of learning situations"

Edit name of situation "practical"

Name of situation: practical laborator|

Figure 59: *Enter Grid* script editing an element name

Arthur is considering "exploring the nature of learning situations"

Edit rating for situation "practical laboratory"

self-organised—staff-organised: 3|

Figure 60: *Enter Grid* script editing a rating on a constructt

Arthur is considering "exploring the nature of learning situations"

List of qualities

involvement—remoteness
flexible—rigid
equipment—no equipment
self-organised—staff-organised
small group—large group
variable content—specific content
like—dislike

Click here to add more qualities.
Click in an item to edit it, or here if you have finished editing.

Figure 61: *Enter Grid* script listing constructs

Arthur is considering "exploring the nature of learning situations"

Edit quality "self-organised—staff-organised"

Left pole rated 1: self-organised

informal interaction: 1
library: 1

programmed text: 2
video tape: 2

practical laboratory: 3

seminar: 4
tutorial: 4

film: 5
lecture: 5

Right pole rated 5: staff-organised

Click here to delete the quality
Click on the situation or the pole name to edit it, or here when you have finished editing

Figure 62: *Enter Grid* script entering pole names of the first construct

tive to editing through the *Elements* and *Constructs* panes. The editing is synchronized between the scripts and the panes so that it is possible to switch back and forth between them if desired.

4.2 Elicit Grid

The *Elicit Grid* script supports conversational elicitation of conceptual grids by emulating the Shaw's (1980) interactive repertory grid elicitation program, "PEGASUS".

The script is programmed to:-

1. Request fields that are empty in the *Status* window such as the user's name and the purpose of the elicitation
2. Ask the user to enter six or more elements
3. Elicit constructs from triads of elements until there are four
4. Check element and construct matches and offer the user the opportunity to enter more constructs or elements to break the matches
5. Offer the user the option to elicit more constructs from triads, edit and enter elements or constructs, or to finish the elicitation
6. Ask the user to rate the elements on any given constructs when the elicitation process is finished
7. Modify the elicitation process appropriately to elicit ratings for *exchange* grids in which the elements and constructs are given but the ratings are open, *elements* grids in which the *elements* are given, and *constructs* grids in which the constructs are given.

Once steps 1 through 3 are complete the scripts loop between steps 4 and 5.

At any time during the elicitation the user can click on the analysis facilities available through the *Display*, *Synopsis*, *Focus*, *PrinGrid*, and so on, buttons to display or analyze the grid, and then continue the elicitation.

The *Elicit Grid* script operates in a similar way to the *Enter Grid* script except that, as shown below, the interaction is more tutorial with and greater explanation feedback of matches to suggest appropriate element and construct elicitation.

Clicking on the script menu option *Elicit Grid* brings up an initial explanatory dialog (Figure 63).

The initial elements are entered in much the same way as for the *Enter Grid* script (Figure 64).

The element elicitation section of the script can be edited to accommodate more specific requests for particular types of element as, for example, might be appropriate to a *role grid* requesting that family members and friends be entered, a core competency grid requesting that employees with different types and levels of skills be entered, or a market research grid requesting that products of certain categories and qualities be entered.

The *Elicit Grid* script differs from the *Enter Grid* script primarily in that it elicits constructs using triadic elicitation in which the user is asked in which way two elements are alike and differ from a third, and in the feedback of element and construct matches to prompt the elicitation of further constructs and elements to reduce the matches (Shaw, 1980).

Three elements are selected by the script and the user is asked to consider in what way two are alike and different from the third, and to click on the one that is different (Figure 65).



Figure 63: Initial *Elicit Grid* script requesting name and purpose

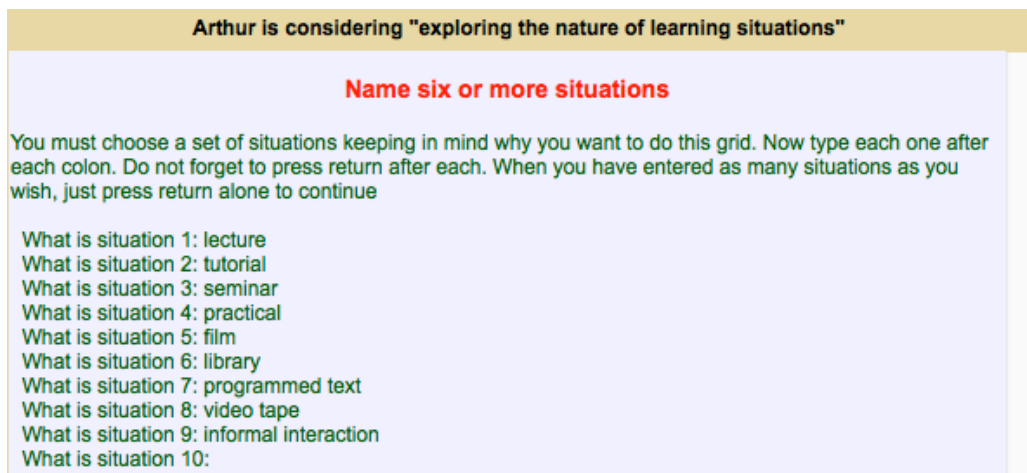


Figure 64: *Elicit Grid* script entering element names

When the user clicks on an element in the triad they are asked to enter the construct pole names construct pole names for the distinction being made (Figure 66).

They are then shown the three elements of the triad rated appropriately and asked to rate the remaining elements on the elicited construct (Figure 67).

Arthur is considering "exploring the nature of learning situations"

Elicit quality from a triad

Can you choose two of this triad of situations which are in some way alike and different from the other one?

tutorial
practical
video tape

Click in the situation which is different, or here if you cannot do this.

Figure 65: *Elicit Grid* script eliciting a construct from a triad of elements

Arthur is considering "exploring the nature of learning situations"

Name the poles of your quality

Now I want you to think what you have in mind when you separate the pair from the other one. Just type one or two words for each pole to remind you what you are thinking or feeling when you use this quality.

Or click here if you cannot do this

Left pole rated 1 {practical, video tape}: equipment
Right pole rated 5 {tutorial}: no equipment

Figure 66: *Elicit Grid* script entering construct pole names

Arthur is considering "exploring the nature of learning situations"

Rate the situations on your quality "equipment—no equipment"

According to how you feel about them, please assign to each of the following situations a rating from 1 (equipment) to 5 (no equipment) by entering a number or clicking to use a popup menu.

practical 1
video tape 1
tutorial 5
lecture:

? Unspecified
1 equipment
2 equipment
3
4 no equipment
5 no equipment

Figure 67: *Elicit Grid* script entering element ratings

When all the elements have been rated the screen in Figure 68 is shown which makes the pole names and ratings available for editing as has been shown for the *Edit Grid* script. Since the process is now one of elicitation rather than data entry it is likely that the user will not be content with the ratings as initially entered and will edit them at this point as illustrated in the previous section.

Construct elicitation continues with further explanation and another triad of elements (Figure 69).

Arthur is considering "exploring the nature of learning situations"

Edit quality "equipment—no equipment"

Left pole rated 1: equipment

video tape: 1
practical: 1

programmed text: 3

library: 4
film: 4
lecture: 4

seminar: 5
tutorial: 5
informal interaction: 5

Right pole rated 5: no equipment

Click on the situation or the pole name to edit it, or here when you have finished editing

Figure 68: *Elicit Grid* script entering element ratings

Arthur is considering "exploring the nature of learning situations"

How to think about qualities

Now you have one quality you know what to do. You may think of qualities as lines along which each of your situations has a place in relation to all the other situations. Please do not use qualities which do not apply to all your situations. An example of this is *redhead--blond*, as it is impossible to rate a person with black hair on this quality. One pole must be in some sense what the other is not, and they must divide your situations into two approximately equal groups, so please try to avoid qualities where nearly all the situations are at one end. An example might be "extremely tall--not extremely tall"

Can you choose two of this triad of situations which are in some way alike and different from the other one?

**practical
informal interaction
programmed text**

Click in the situation which is different, or here if you cannot do this.

Figure 69: *Elicit Grid* script eliciting a construct from a second triad of elements

When four constructs have been elicited through triads the script tests for matches between constructs and between elements and, if it finds any above eighty per cent, asks the user to enter an element or construct to reduce the match (Figure 70).

The user clicks on *self-organized and rigid*, and enters a new element, *programmed text*, that is construed as both and should reduce the construct match (Figure 71).

The new element is then shown already rated appropriately on the matching constructs and the user is asked to rate it on the remaining constructs (Figure 72).

When the new element has been rated the name and ratings can be edited as already shown. The test for matches continues and notes an element match which prompts the elicitation of a new construct (Figure 73).

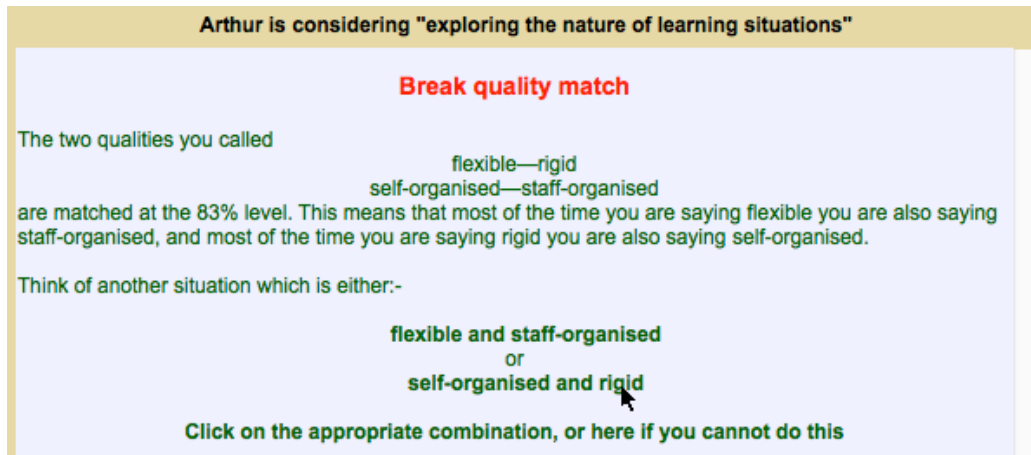


Figure 70: *Elicit Grid* script eliciting a construct from a triad of elements

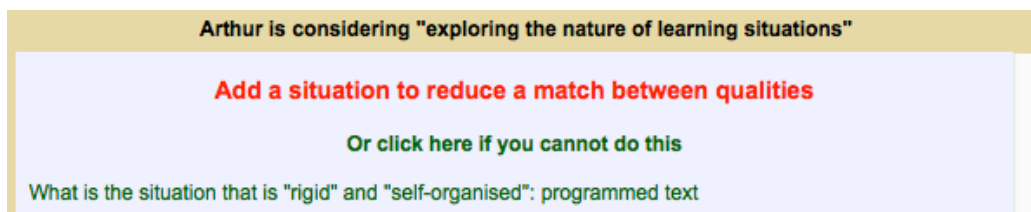


Figure 71: *Elicit Grid* script eliciting a construct from a triad of elements

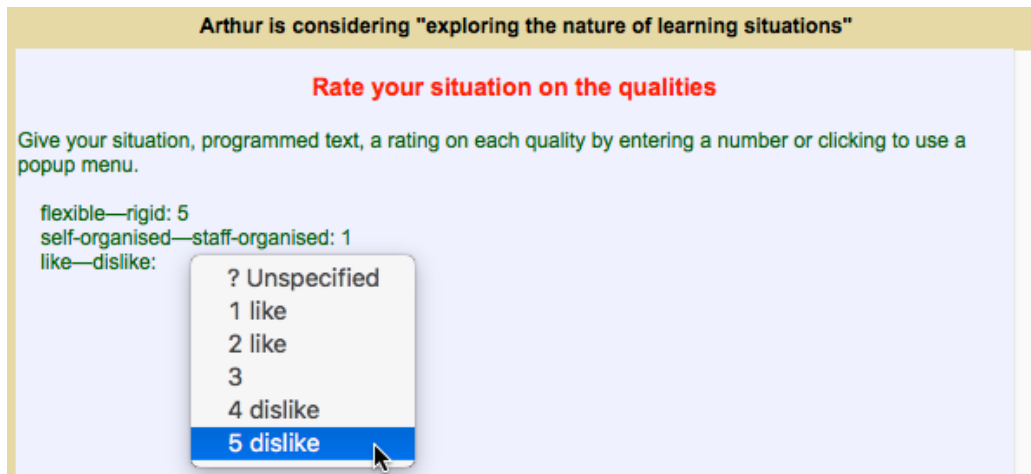


Figure 72: *Elicit Grid* script eliciting a construct from a triad of elements

The new construct is then shown with the matched elements already rated appropriately on the new construct and the user asked to the remaining elements on the new construct (Figure 74).

The process continues until there are no further matches to be displayed and various options for further elicitation are then listed (Figure 75).

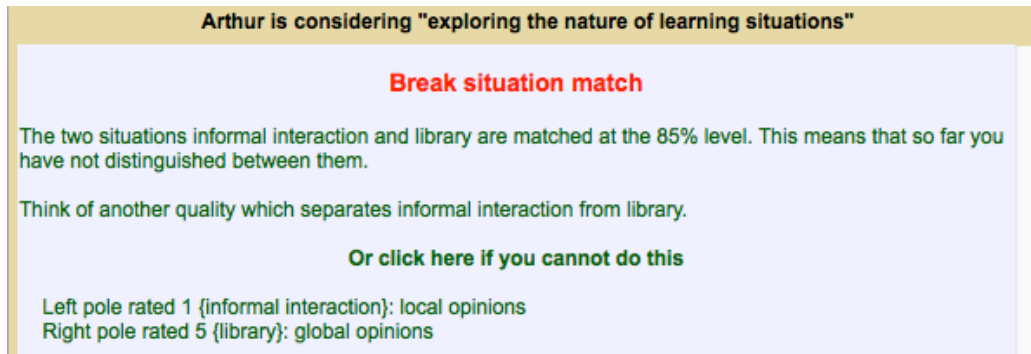


Figure 73: *Elicit Grid* script eliciting a construct from a triad of elements

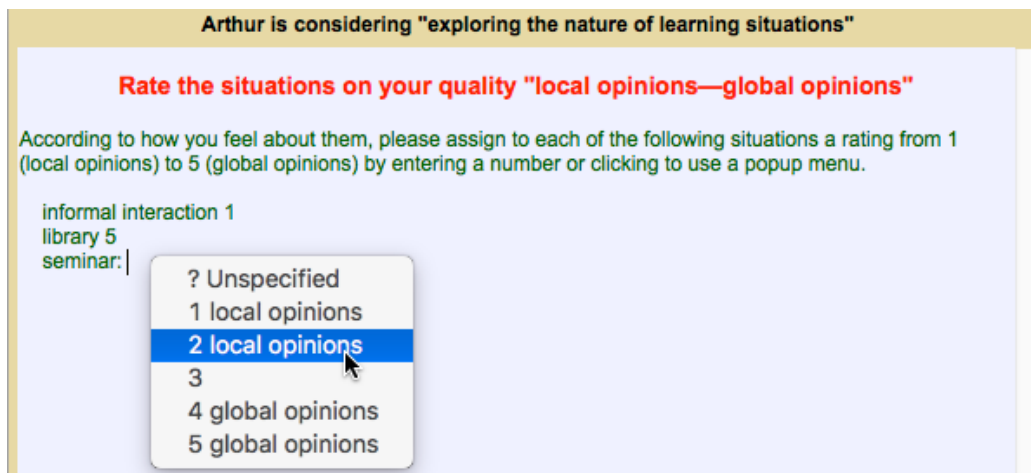


Figure 74: *Elicit Grid* script eliciting a construct from a triad of elements

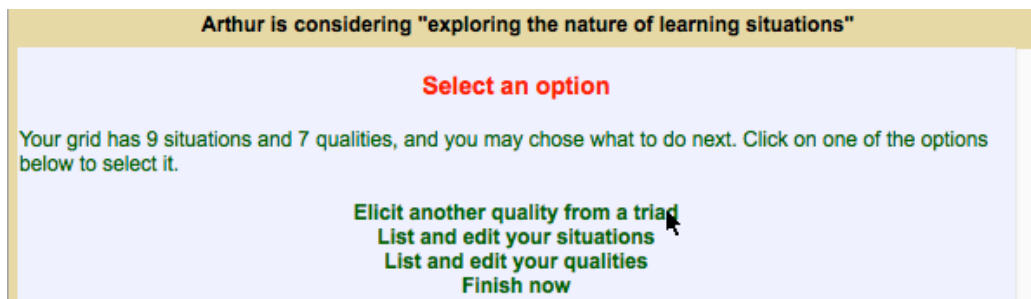


Figure 75: *Elicit Grid* script options

The options are the same as those for the *Enter Grid* script with the addition of one for triadic elicitation. Selecting this now also provides an option for the user to select some or all the elements of the triad (Figure 76).

The user can also use the editing facilities to add and rate elements and constructs directly. The test for matches is applied each time the user enters an item so that further feedback is given if appropriate. This process of elicitation from triads and matches, and entry and editing, proceeds until the user is satisfied that the grid is relatively complete and chooses the option to finish.

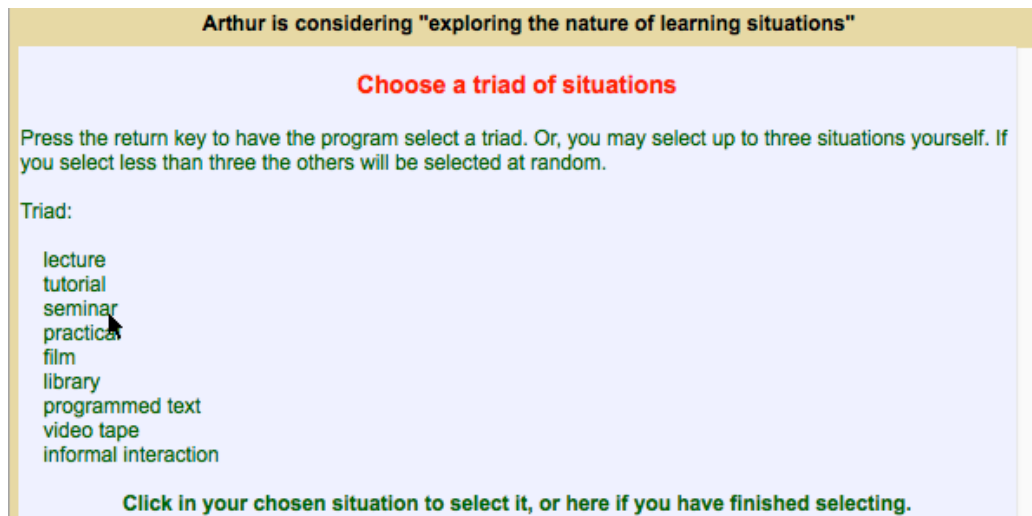


Figure 76: *Elicit Grid* script selecting elements for a triad

Before finishing a check is made for unrated constructs such as *given constructs* provided by the facilitator and the user is asked to rate the elements on these if there are any.

The *Display*, *Synopsis*, *Focus Cluster*, *PrinGrid Map* and *CrossPlot* buttons may be used during the elicitation to supply an interim display or analysis of the grid. Viewing this may prompt the entry of further elements and constructs or the editing of the grid.

The *Elicit Grid* script can be run with an existing a grid at any time to elicit further elements or constructs and will automatically skip steps 1, 2 or 3, if the grid already contains the appropriate data. For example, one can prepare a grid for use by others with the purpose specified as *to understand people I know*, with *person* as the term for an element, *people* for elements, *characteristic* for construct, *characteristics* for constructs, and with initial elements *self* and *ideal self*. Running the *Elicit Grid* script set starting with a copy of this grid will result in the user being asked their name, being asked for additional examples of people, having four initial constructs elicited from triads, and then being taken through an elicitation process for further elements and constructs. One can also enter some given constructs such as *powerful—powerless* that capture an important aspect of the purpose of the elicitation, and, as already noted, the script will ask the user to rate the elements on these as they finish.

To facilitate comparison of grids the *Rep Plus Manager* window allows one to open a copy of an existing grid with the ratings reset to be open (*Exchange*) or with the constructs removed (*Elements*) or the elements removed (*Constructs*). Another user, or the same user at a later time, can fill in the ratings and add constructs and elements to such grids for comparison with the original (§5.6). It first requests the user's name. The *Elicit Grid* script supports elicitation commencing with such partial copies of grids. It first requests the user's name and then: with *Exchange* grids it asks the user to rate all the elements on each construct in turn; with *Elements* grids it proceeds immediately to triadic construct elicitation; with *Constructs* grids it requests elements and then asks the user to rate all the elements on each construct in turn. It then offers the normal options.

4.3 Export grid data

Two scripts are supplied which allow the grid data to be exported in formats suitable for use in other applications.

The *Export Text* script writes the grid data into a text window in the basic grid format described in §9.1.

The *Export Spreadsheet* script writes the grid data into a text window in the tab-delimited grid format described in §9.2.

The data in the text window can then be edited, saved, copied and pasted, or dragged to another application.

4.4 Analyze grid data

Scripts have full access to all the information in the grid and may be used to provide additional analysis capabilities. The *PrinComp* script is provided as an example of the use of the mathematical and graphic libraries available in RepScript to generate a principal components analysis with graphical and textual output.

4.5 Modifying scripts

The scripts supplied are intended as examples that a facilitator can copy and modify to serve specific requirements and user communities.

RepGrid looks for GridScripts directories both in the application directory and in the *Rep Plus* directory where the default files are kept as discussed in §2.2. The *Rep Plus* directory in the *Documents* or *MyDocuments* directories is intended for scripts developed by the facilitator.

For example, one might copy the *Elicit Grid* scripts from the Rep Plus application directory, rename them with an appropriate name to appear in the popup menu, and edit the *Main* script to make the initial element elicitation more specific to the purpose, for example, by requesting the names of the user's mother and father, good friend, teachers, and so on.

One might also develop highly specific elicitation procedures for specific purposes such as market research, knowledge management or system design requirements elicitation.

It is also reasonably straightforward to translate the existing scripts to support conversational elicitation in languages other than English by replacing the English text with the equivalent in another language. It may be necessary to slightly reorder the output to reflect the different literary style of the target language.

5 Grid display and analysis

The buttons at the bottom of any pane of the RepGrid window provide access to various grid display and analysis functions (Figure 77) which will be described in the following sections.

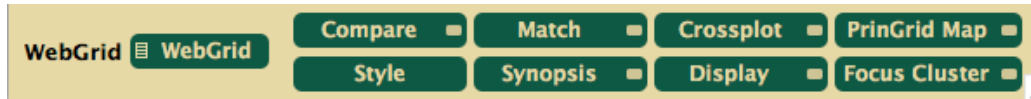


Figure 77: RepGrid analysis buttons

The buttons may have a small icon on the left or right. Clicking in the body of the button generally performs the analysis immediately. When the mouse cursor is over the button icon on the right of a button it changes to a button shape, and clicking brings up a dialog enabling the analysis parameters to be changed. When the mouse is over the menu icon on the left of a button it changes to show a menu symbol, and clicking on it evokes a popup menu enabling the function of the button to be changed.

5.1 Display: Plotting the grid as a matrix of ratings of elements on constructs

Clicking on button icon on the right of the *Display* button brings up a dialog which controls the way in which the content of the grid is displayed as a matrix (Figure 78).

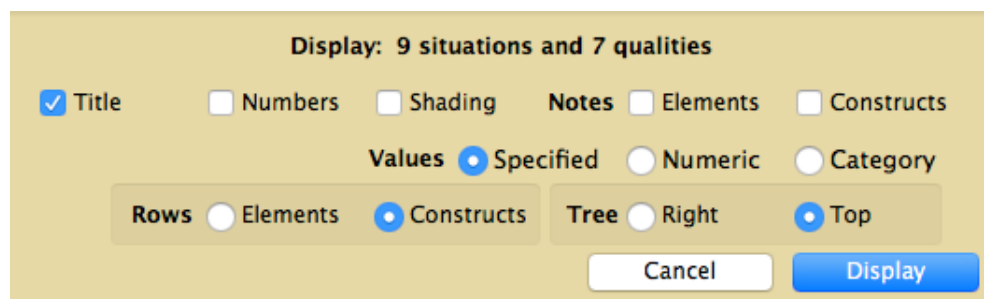


Figure 78: RepGrid *Display* dialog

The row of check boxes at the top determine whether the plot is titled, whether the elements and constructs are numbered, whether the ratings are shaded (to indicate the top third of high values and bottom third of low values), and whether the notes attached to elements and/or constructs are shown.

The *Values* panel determines whether numeric or categorical values should be displayed: as specified by the *Use in plots* checkbox in the construct dialog, or temporarily overriding that setting for all the constructs.

The *Rows* panel determines whether the matrix of grid data is displayed with elements or constructs as rows. For conceptual grids with only rating scale constructs it is conventional that constructs are displayed as rows, but where categorical values are displayed the plot has a more condensed format is the constructs are displayed as columns.

5.1.1 Display plot output

Figure 79 shows the plot produced when one clicks the *Display* button with the settings above. The title, constructs, elements and ratings are shown.

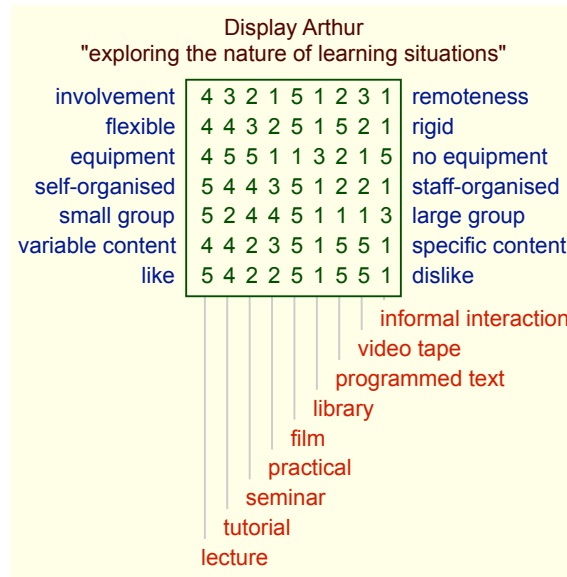


Figure 79: Display of the data in a grid as numbers

Figure 80 shows the plot produced when one specifies the values should be shown as categories and that the rows should be elements. Note that when no category applies to the middle value of the scale then none is shown.

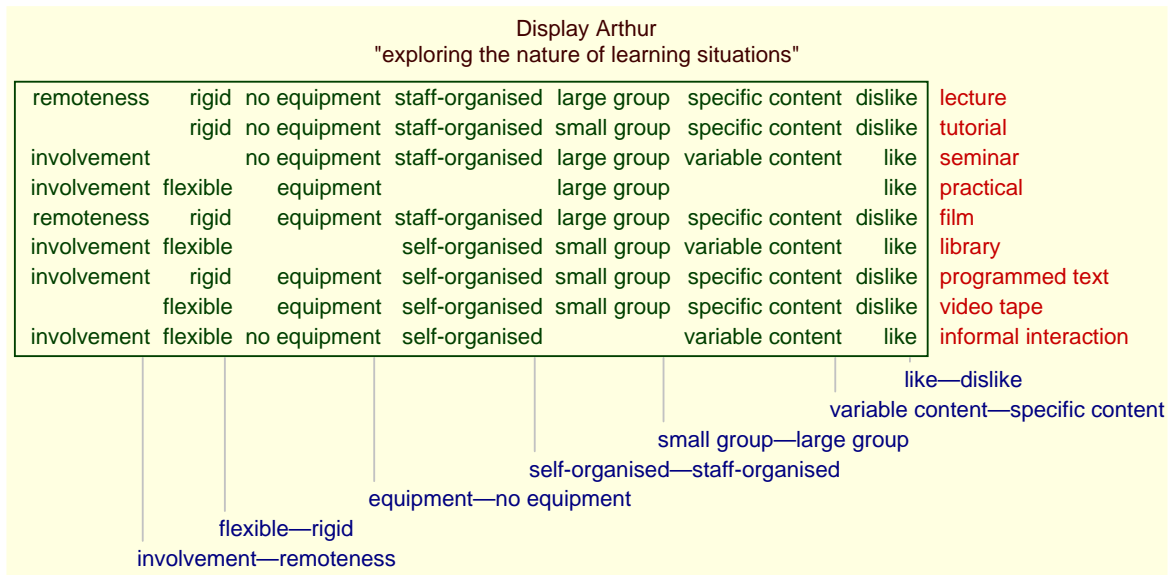


Figure 80: Display of the data in a grid as categories

Figure 81 shows a display of the house choice grid used in earlier examples which uses four types of construct.

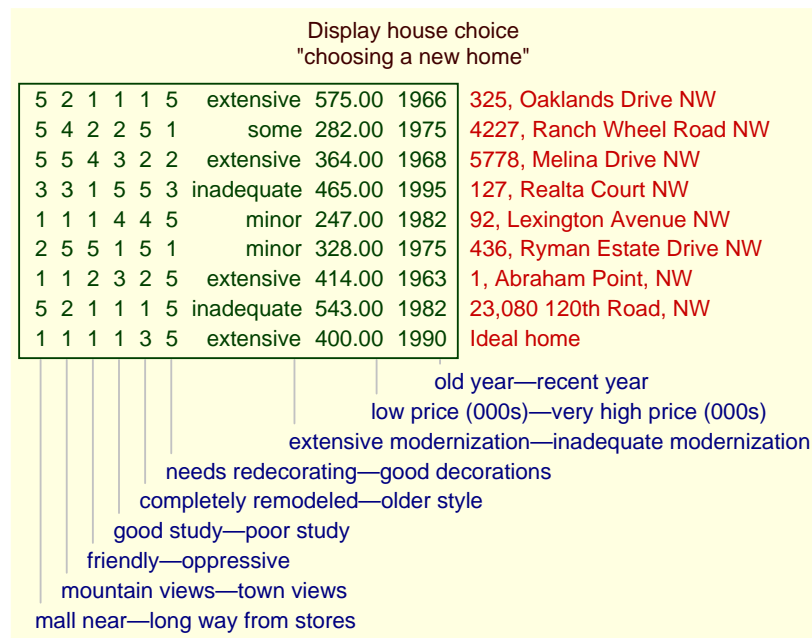


Figure 81: *Display of the data in a mixed-type grid—constructs as columns*

This mixed-type grid is displayed with constructs as columns as this is more compact than a display with constructs as rows shown in Figure 82.

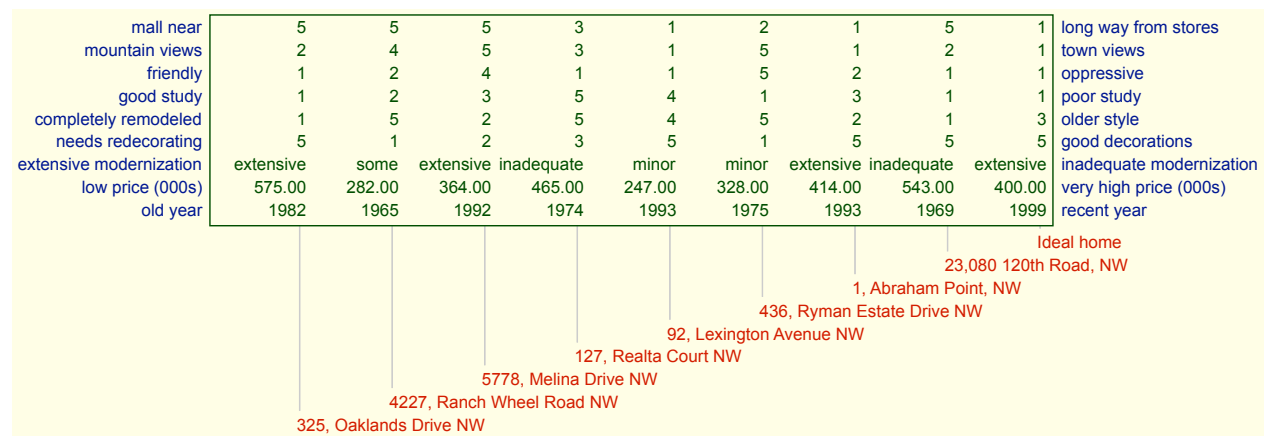


Figure 82: *Display of the data in a mixed-type grid—constructs as rows*

5.1.2 Including classes as elements or constructs in the analyses

As discussed in §3.4.7, classes may be converted to *ideal elements* or *compound constructs* as part of the grid being analyzed, participate in the analysis process and be shown in the plots. Figure 47 in

that section shows how the contact lens grid used to illustrate classes in §3.4.1 may be displayed with the classes represented as ideal elements. Figure 83 shows the same classes displayed as compound constructs. The pole name are labelled with the mathematical symbols \in for *is an element of* and \notin for *is not an element of* to indicate whether an element is, or is not, classified under the class specified.

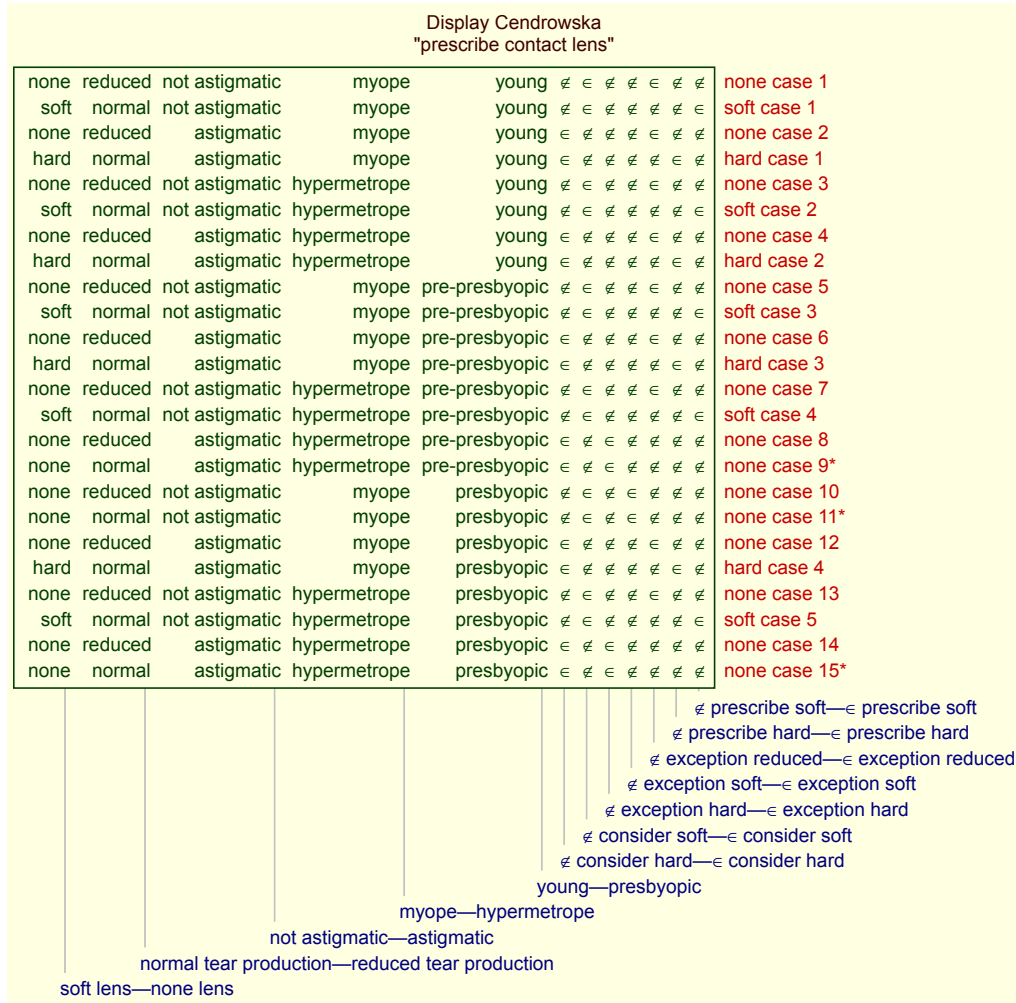


Figure 83: *Display* with classes included as compound constructs

The other analysis programs treat the elements or constructs representing classes as if they were normal parts of the grid and the inclusion of classes for each will not be illustrated except for the *Matches* analysis where it is insightful to examine the relation between ideal elements that could be entered without the availability of classes and those deriving from classes (§5.7.1).

5.2 Synopsis: Histograms and scree plot

Display presents the grid as a matrix of ratings of elements on constructs and is useful for examining the grid data, for example, in checking data entry. There are other assessments of the data in a grid

where an alternative form of display is useful, for example, how the element ratings are distributed across the possible values; are they skewed, have both poles been used, and so on. A histogram of the distribution of the element ratings on a construct supports rapid appraisal of such considerations. One may also be interested in the *complexity* of cognition exhibited; are all the constructs similar in the distinctions they make, how many distinctions are being instantiated, and so on.

The *Synopsis* presentation provides a different presentation of the grid data that addresses these issues by displaying histograms of the distribution of the element ratings on the constructs and a scree plot of the variance in the data accounted for by its principal components.

Clicking on button icon on the right of the *Synopsis* button brings up a dialog which controls the way in which the output is displayed (Figure 84). There are two panels, one managing the output of the histograms, and the other the scree plot of the principal components.

Figure 84: RepGrid *Synopsis* dialog

The check boxes on the histograms panel determine whether: the plot is titled; the constructs are numbered; the notes attached to the constructs are shown; a horizontal rating value scale and/or a vertical count scale are shown. The radio buttons determine whether the element labels should be as specified by *Use in plots* or all numeric or categorical. The text field on the right enables the geometry of the histograms to be adjusted. It can contain up to 5 numbers separated by commas: the number of pixels for each vertical increment in the histogram bars; their widths; horizontal space between bars; vertical space between histograms, and horizontal space at the end of each plot. If fewer than 5 values are specified the remainder are filled from the appropriate position in the default values 3, 3, 8, 4, 4.

The check boxes on the components panel determine whether: the plot is titled; Frontier's (1976) comparative plot of the distribution if the principal components were randomly generated should also be plotted to provide an estimate of the number of significant components. The text field on the right enables the geometry of the scree plot to be adjusted. It can contain up to 4 numbers separated by commas: whether the vertical numeric increments are 1 or 2; vertical separation between scale points; horizontal separation between scale points; horizontal space at each end of plot. If fewer than 4 values are specified the remainder are filled from the appropriate position in the default values 2, 3, 20, 10.

In both cases there is rarely a need to change the detailed geometry but the options are available to fine-tune the output for presentation or publication. Mousing over either the text field brings up help text giving information about available parameters.

5.2.1 Synopsis histogram and scree plots output

Figure 85 shows the plot produced when one clicks the *Synopsis* button with the settings above for the grid displayed in Figure 79.

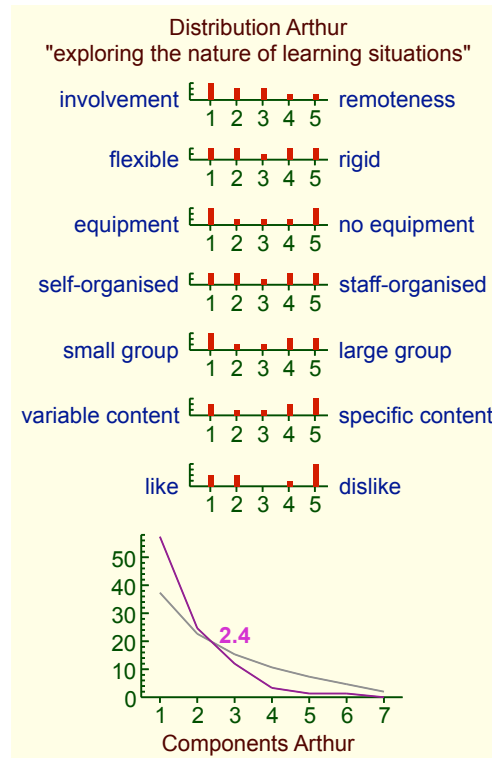


Figure 85: Synopsis of grid data

Figure 86 shows the plot produced when one clicks the *Synopsis* button with the settings above for the grids displayed in Figure 81. The histograms on at the bottom illustrate how construct categories are shown in histograms.

Both grids were elicited using PEGASUS-style elicitation with feedback of matches and the histograms show that the ratings on all constructs are fairly evenly distributed with poles used. Grids elicited in other ways may not have these characteristics, such as those with given constructs that those filling in the ratings do not normally use within their own constructions.

The scree plots suggest that the first grid exhibits some 2 significant underlying dimensions, and the second right 3. This provides a rapid assessment of the cognitive complexity of the grids—a full PrinGrid analysis (§5.5) would enable the nature of the underlying distinctions to be investigated. Note that the estimate of significant dimensions does not necessarily indicate that some constructs

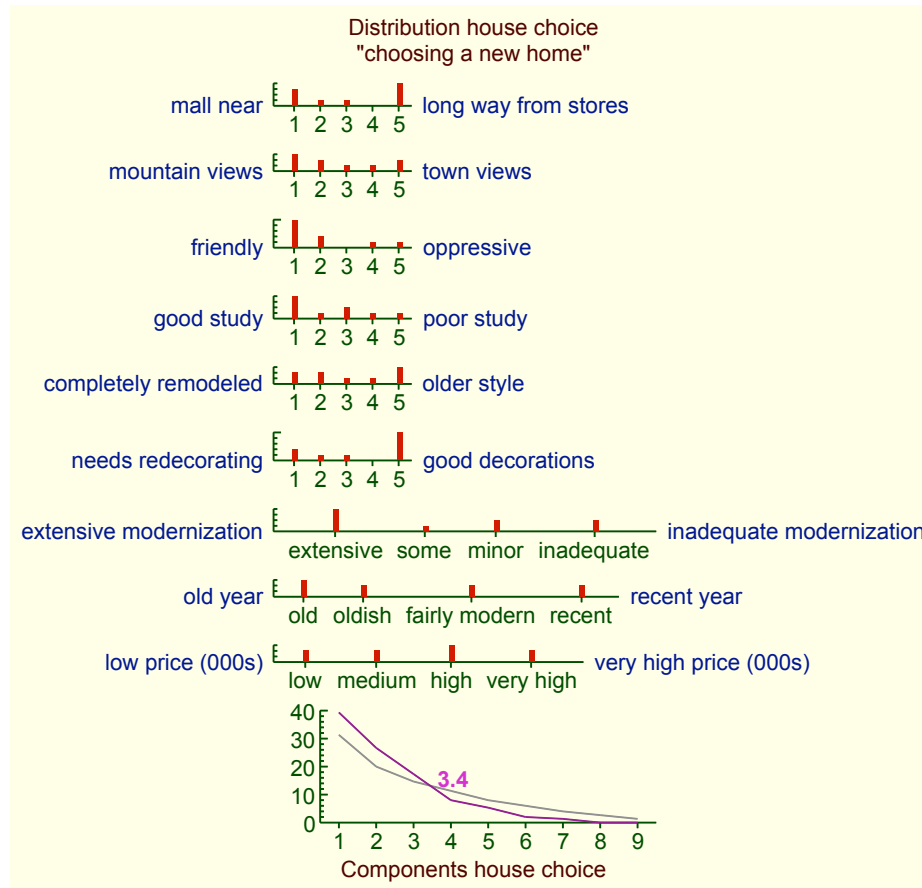


Figure 86: Synopsis of grid data with mixed types

are redundant but more likely that there are insufficient elements to discriminate between all the constructs.

5.3 Focus: Sorting by similarity and hierarchical clustering

Shaw's (1980) Focus algorithm sorts the rows and columns of the grid display to bring similar elements and similar constructs closer together, and also shows the hierarchical structure of similarities that results from sorting the grid in this way.

Clicking on button icon on the right of the *Focus* button brings up a dialog that controls the Focus analysis which can be presented as a graphic plot or as textual data (Figure 87).

The upper pane controls the presentation of the graphic plot. The row of check boxes at the top determine whether: a plot is produced; it is titled; the elements and constructs are numbered; the ratings are shaded (to indicate the top third of high values and bottom third of low values); the notes attached to elements and/or constructs are shown.

The dialog box is titled "Focus Cluster 9 situations and 7 qualities". It contains several sections of controls:

- Top Row:** Checkboxes for ☒ Plot, ☒ Title, ☐ Numbers, ☒ Shading, ☐ Notes, ☐ Elements, and ☐ Constructs.
- Values Section:** A label "Values" followed by three radio buttons: ☒ Specified, ☐ Numeric, and ☐ Category.
- Rows and Tree Section:** Two groups of radio buttons. The first group has "Rows" (radio), ☐ Elements, and ☒ Constructs. The second group has "Tree" (radio), ☒ Right, and ☐ Top.
- Interior and Metrics:** A checked checkbox for ☒ Interior. To its right are three input fields: "Power" with value 1.0, "Cut off" with value 25, and "Elements" with value 25. Further right are the labels "Constructs" and "Scale" with value 100.
- Bottom Row:** A row of checkboxes: ☐ Data, ☒ Elements, ☒ Constructs, ☒ Matches, ☒ Links, and ☒ Sorts.
- Buttons:** "Cancel" and "Focus" buttons at the bottom right.

Figure 87: RepGrid *Focus Cluster* dialog

The *Values* panel determines whether numeric or categorical values should be displayed: as specified by the *Use in plots* checkbox in the construct dialog, or temporarily overriding that setting for all the constructs.

The *Rows* panel determines whether the matrix of grid data is displayed with elements or constructs as rows. The *Tree* panel determines whether Focus cluster tree for the columns is shown at the top of the grid or at the lower right.

The *Interior* check box controls the Focus matching strategy. Leaving it unchecked specifies the standard Focus algorithm in which items are matched only against the items at the edges of existing clusters. This sometimes leads to items with a high match to interior items being shown as having a lower match to an edge item. Checking the *Interior* check box allows Focus to match against interior items in an existing cluster; it then displays the interior match and places the item at the edge of that cluster that has highest match to the item.

The *Power* value determines the exponent used in the Minkowski metric used to compute matching scores (Shaw, 1980, p.160). The default (and generally recommended) power of 1.0 defines the standard city block metric normally used in the Focus algorithm. A power of 2.0 defines a Euclidean metric. Fractional powers in the range 0.1 to 10.0 may be used—a higher power weights larger differences more than smaller ones, and *vice versa*.

The *Cut off* values determine the level of match below which an element or construct cluster will not be shown. The *Scale* value determines how much space will be allocated to the trees showing the cluster hierarchies.

The row of check boxes near the bottom determine whether: textual data from the analysis is displayed; element and construct data are output; match matrices, cluster links, and sorts are output.

5.3.1 *Focus cluster plot output*

Figures 88 and 89 shows the Focus cluster plots produced for the grids displayed in Figures 79 and 81 when *Focus* button is clicked with the settings above. The grids have been sorted to bring closely matching elements together, and closely matching constructs together.

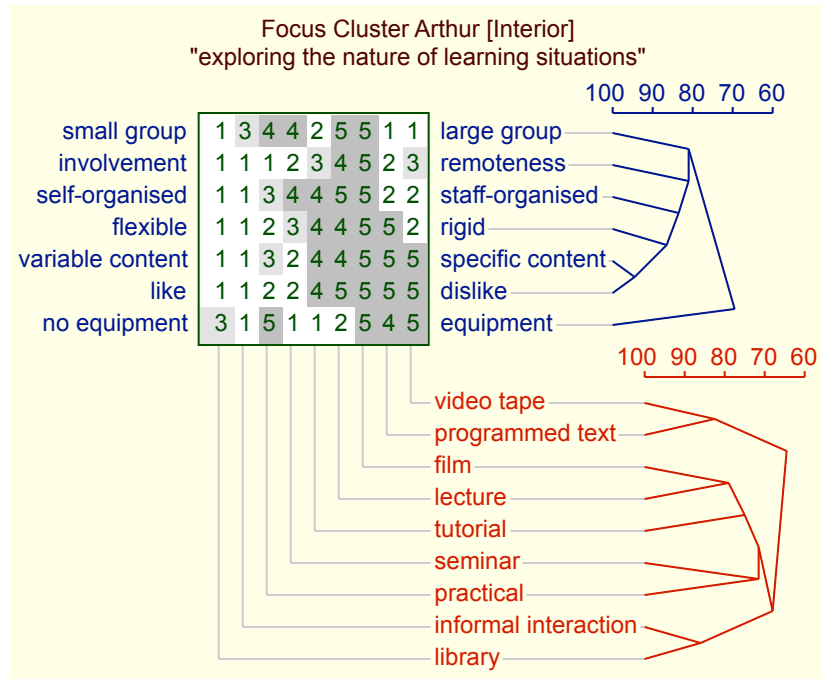


Figure 88: Focus Cluster analysis

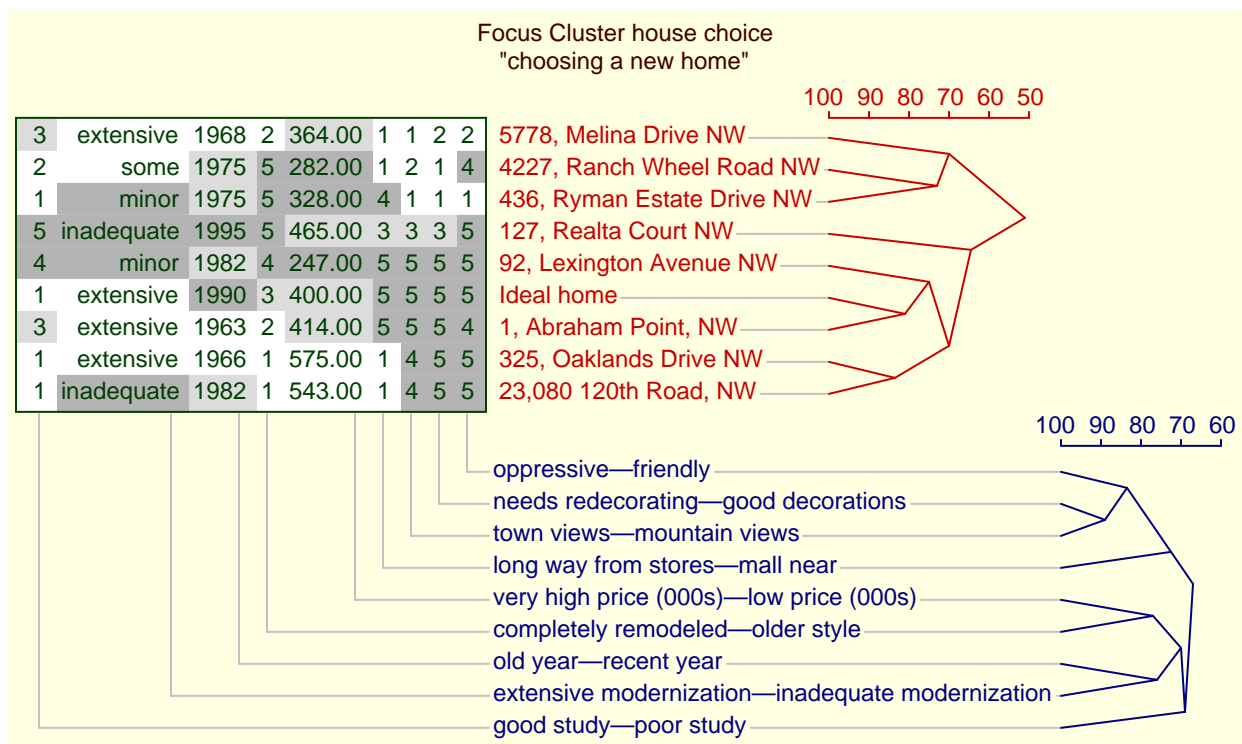


Figure 89: Focus Cluster of a mixed-type grid—constructs as columns

5.3.2 Focus data output

Figure 90 shows the data underlying the Focus analysis of Figure 88 when the *Data* checkbox is set in the Focus dialog.

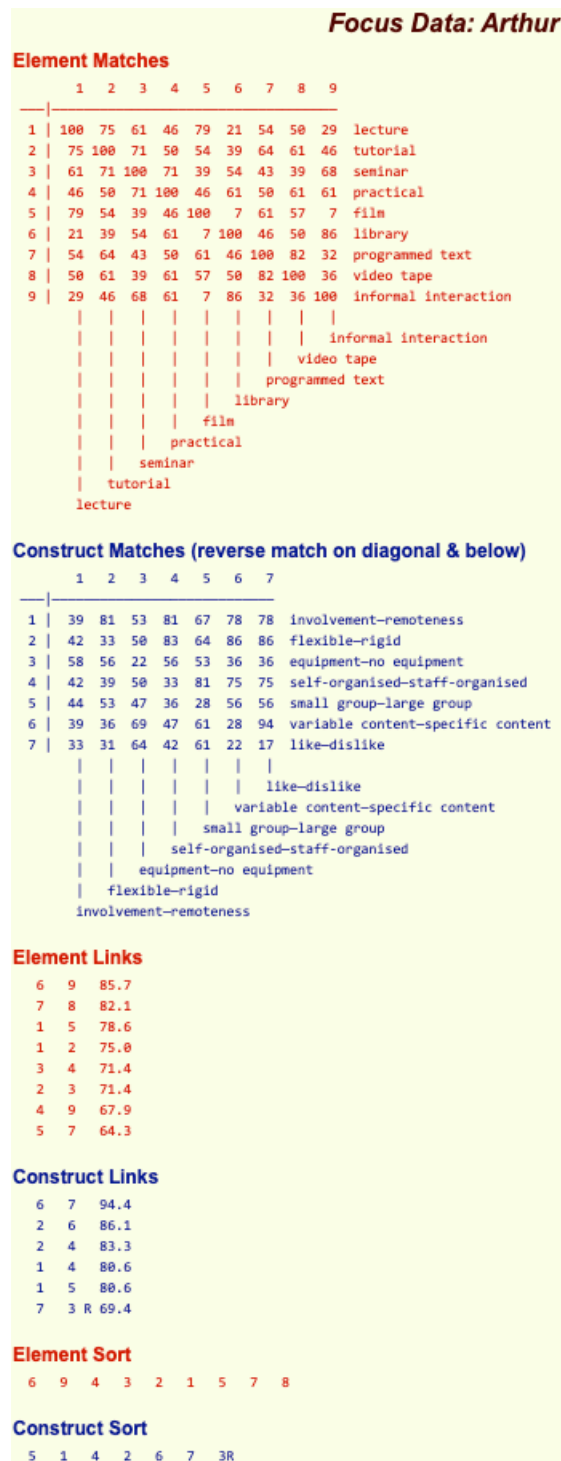


Figure 90: Data underlying a *Focus Cluster* analysis

The element and construct *Matches* are shown as a percentage of the maximum possible match, and it should be noted that the values on the diagonal and below it in the construct match matrix correspond to the match when one of the constructs is reversed in its values. The *Links* data corresponds to the clusters in the plot. The *R* indicates that the Focus algorithm has reversed the construct in determining the highest matches. The *Sort* data indicates how the algorithm has sorted the grid data to produce the Focus plot.

5.3.3 Status of the Focus hierarchical clusters

This manual does not cover the detailed interpretation of grid analyses but there several books that do so (Shaw, 1980, 1981; Shaw and McKnight, 1981; Pope and Denicolo, 2001; Jankowicz, 2004).

One issue that may arise is the meaningfulness of the cluster structure. Most grid datasets are too small for meaningful statistical analysis, but Heckmann and Bell (2016) have shown how *bootstrap* techniques (Efron and Tibshirani, 1993; Davison and Hinkley, 1997) may be used to investigate the robustness of the clusters against perturbation of the grid data. Bootstrap techniques are non-parametric in making no assumptions about the distribution from which the data is drawn, but treat the actual dataset as a sample fully representing that distribution and run the analysis many times against samples of that dataset to estimate the sensitivity of the full analysis to partial datasets.

Users may make a similar analysis in RepGrid by selecting different subsets of elements and constructs and comparing the resulting plots to provide some insight into the dependency of the results on the data provided, perhaps leading to further elicitation if there are concerns about the robustness of what seem to be interesting structures.

Note also that the elicitation techniques used may have a major impact on the representativeness of the grid data and the results of analysis. Shaw's (1980) computer-based elicitation techniques continually analyze the data, feed back high matches between elements and between constructs, and suggest the type of constructs or elements that the elicitee might add to reduce them (§4.2). This is designed to ensure that the clusters shown in analysis are not artefacts of inadequate coverage of the domain.

It is informative to test the cluster analysis on grids with a known hierarchical structure to determine whether it reproduces that structure. Shaw and Gaines (1998) did this for artificial and natural datasets where hierarchical structures were represented in grids. Figure 91 shows the simple hierarchy they used as an initial test.

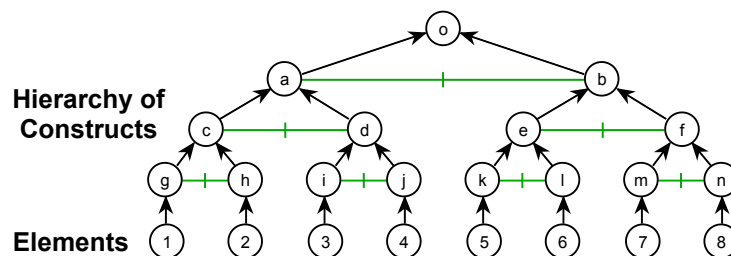


Figure 91: Simple hierarchy

Figure 92 shows this simple hierarchy represented as grid and then clustered using the Focus algorithm. It can be seen that Focus has reconstructed the original hierarchy from the grid representation. Indeed it can be shown from a mathematical analysis that it will necessarily do so for such hierarchical structures.

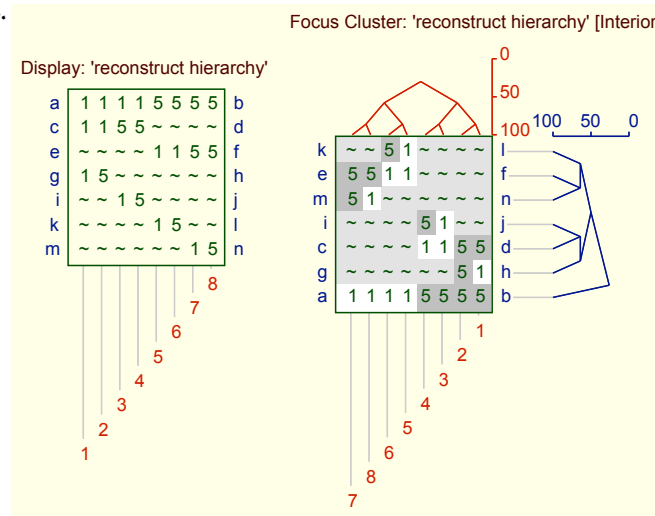


Figure 92: Simple hierarchy represented in a grid, *Display* and *Focus Cluster*

The metavalue *Inapplicable* has been used in representing the ordinal relationships in the hierarchy as discussed in §3.3.6 on ordinal relations in grids. Yorke (1978; 1983) has noted that the middle value of a rating scale is ambiguous in being used for multiple purposes such as inapplicable, and it is interesting to see the impact of doing so with this dataset.

Figure 93 shows the grid with the midpoint value 3 replacing the metavalue ~ together with the resulting Focus analysis. Again it can be seen that Focus has reconstructed the original hierarchy suggesting that the overloading of the midpoint of a rating scale may not be a significant problem in practice.

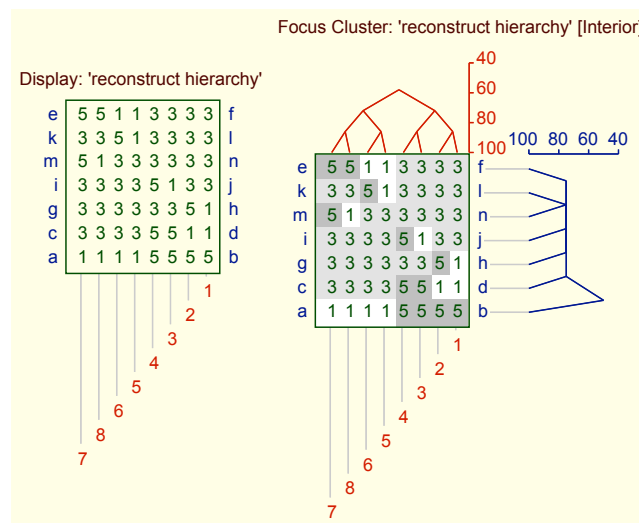


Figure 93: Simple hierarchy represented in a grid with no metavalues

The simple hierarchy is artificial and it is interesting to take a less well-structured example from the literature and test it in the same way. Figure 94 is extracted from a paper analyzing the relations between the early Internet services shortly after the Internet was commercialized in 1995 (Gaines et al., 1997). It may be seen as an example of Plato's *collection and division* (*diareisis*) technique (Kaldis, 2008; Gill, 2010), applied to modern technologies, but also exhibits some of the complications that Aristotle (Balme, 1987) notes in the application of the technique, such as the use of the same *differentia* on different branches, and exemplars that cut across the categorization.

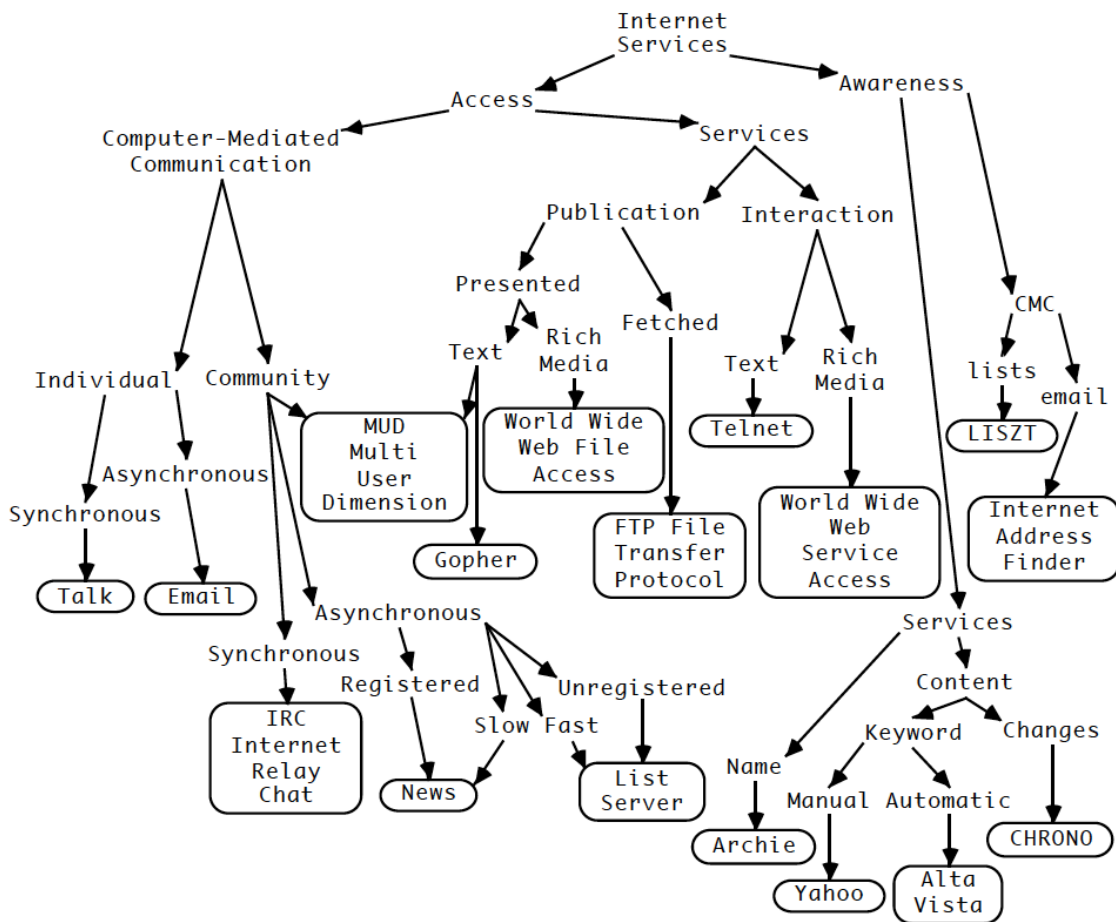


Figure 94: An analysis of relations between Internet services (Gaines et al., 1997)

Figure 95 shows the Internet services represented as a grid using the midpoint of the scale to represent inapplicable, and then clustered in Focus. Again it can be seen that the Focus algorithm has reconstructed the hierarchy, this time developed for an exposition of the services based on the collection and division method. That is, grid elicitation represents an alternative method to Plato's *diareisis* that develops the same structural model of the domain.

The construct matches in Figure 95 illustrate another feature of the Focus analysis. High matches are of obvious interest because they indicate similar constructs, but the low matches of other other constructs are also significant because they indicate major dimensions of construing that structure the domain. For example, the lowest matched constructs in Figure 95 are *access—awareness* and

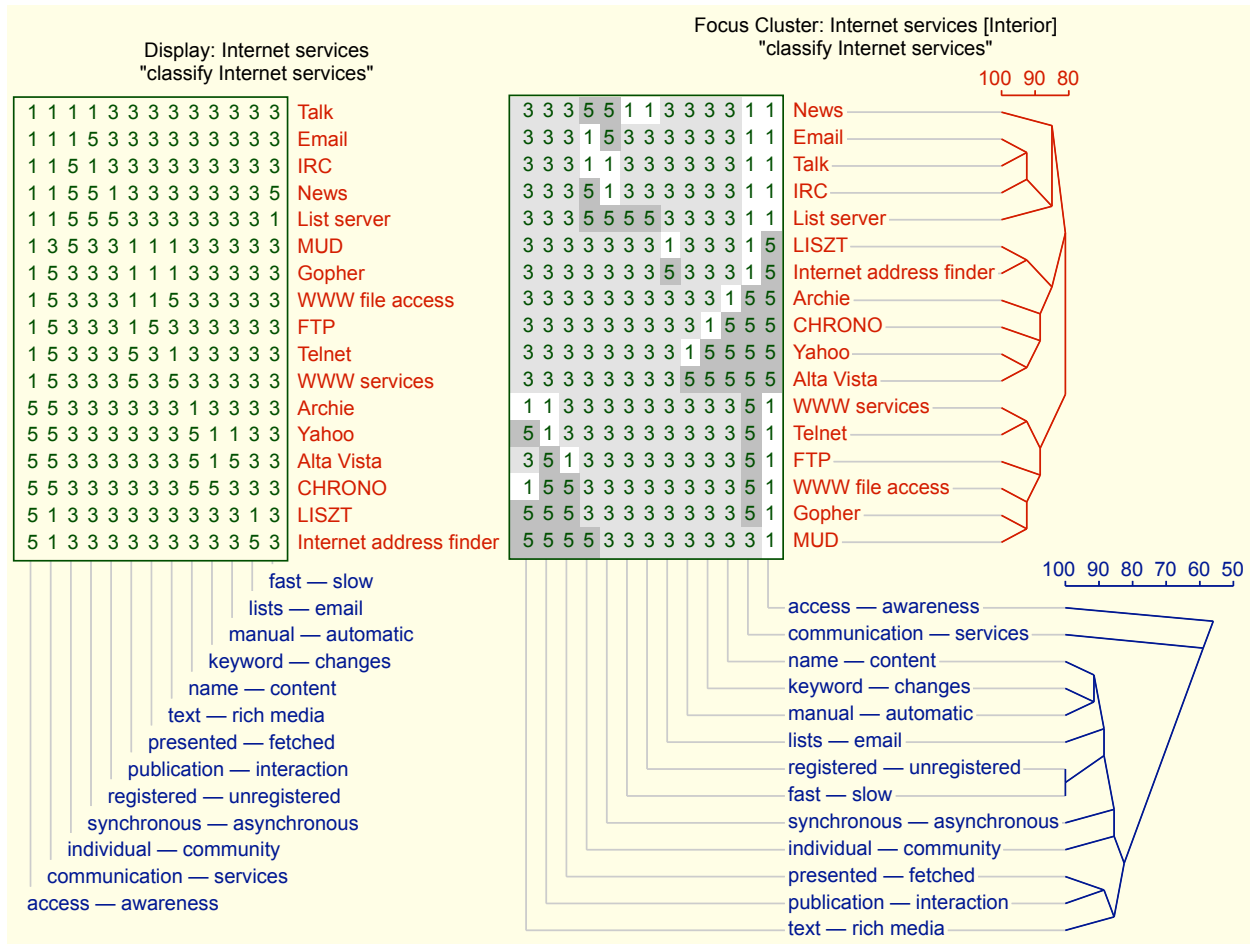


Figure 95: Internet services represented in a grid, *Display* and *Focus Cluster*

communication—services that are the top two distinctions made in the preliminary division of the domain in Figure 94. In a principal components analysis, as discussed in the following section, these constructs will appear as those dominating the structure of the domain and having the highest loading on the first two components. Thus, it is important to study the entire Focus cluster, not just the part representing the high matches.

5.4 PrinGrid Map: Spatial rotation and scree plot

Kelly (1955, ch.6) presents constructs as defining the dimensions of a *psychological space* (Kelly, 1969; Shaw and Gaines, 1992) in which the elements are located. The PrinGrid analysis is consistent with this and treats a grid as defining a geometric configuration in which the constructs form the axes of an n-dimensional space and the elements are represented by points located in that space determined by their ratings on the constructs. It rotates the configuration to lower its dimensionality as much as possible so that it may be plotted with principal components as axes in 2 or 3 dimensions.

Slater (1976; 1977) first applied this method of *principal components analysis* to conceptual grid data using a non-statistical, distance-based, geometric model and algorithms developed by Gower (1966) and his presentation of the results as *biplots* (Gower and Hand, 1995; Gower et al., 2011).

Clicking on button icon on the right of the *PrinGrid* button brings up a dialog which allows a principal component analysis of the grid to be displayed (Figure 96).

The dialog box is titled "PrinGrid Map 9 situations and 7 qualities". It contains several rows of checkboxes and input fields. The first row has checkboxes for Plot, Title, Numbers, Variance, Axes, and Labels. The second row has checkboxes for Elements, Notes, Wrap, Vertical, and Voronoi. The third row has checkboxes for Constructs, Notes, Wrap, Vertical, Means, and Dimensions. Below these are input fields for Scale (100), Cut off (1.0), C/E (1.0), Margins (20 V), and Spread (checked). The next row shows Axes X (1: 57.3%) and Y (2: 24.8%) with Reverse checkboxes. The following row has checkboxes for 3D, Rotate (45 °X, 30 °Y, 0 °Z), and Z (3: 12.0%) with Reverse checkboxes. The eighth row has checkboxes for Scree, Title, Compare, and Scales (2). The bottom row has checkboxes for Data, Variance, Loadings, Elements, and Constructs. At the bottom are buttons for Covariances, Cancel, and PrinGrid.

Figure 96: RepGrid *PrinGrid Map* dialog

The check boxes in the first row determine whether: a graphic plot is produced; it is titled; the elements and constructs are numbered; the percentage variance accounted for by each component is shown at the bottom of the plot; the component axes are shown; the axes, if shown, are labelled by component number and percentage variance.

The check boxes in the second row determine whether: the constructs and the notes attached to them are shown; the construct names are allowed to wrap if there are multiple words; they are placed vertically below or above their location in the plot, or horizontally to its right or left; a Voronoi diagram is plotted to indicate the points closest to the element location.

The check boxes in the third row determine whether: the elements and the notes attached to them are shown; the element names are allowed to wrap if there are multiple words; they are placed vertically below or above their location in the plot, or horizontally to its right or left; the constructs are plotted with the mean at the origin (so that asymmetric distributions of elements on constructs are visible); the construct dimensions are shown as a line between the poles.

Note that the *Means* check box determines whether the construct pole positions are symmetrical about the origin or whether they are placed such that the mean of the element values on the construct is at the origin. This option is useful to allow the plot to convey information about the asymmetrical use of a construct. It maximizes the information conveyed in that the angles between dimensions convey the correlations between them, the length of each conveys the combined loadings on the components plotted, and the position of the centre conveys the mean element value.

The check boxes in the fourth row determine: the overall scale of the plot (a useful option for larger grids to make more space for the labels and prevent them being spread too far from their orig-

inal positions); the *cut off* variance percentage below which components will not be shown; the ratio between the construct and element plot position multipliers (adjusting the length of the construct axes); the margins, if any, that will be added to the rectangle around the plotted points (specified as two numbers with a space between for horizontal or vertical components, e.g. 5 15, or one specifying both, followed by an optional “V” if the margins only apply when a Voronoi diagram is plotted); whether the element and construct pole names are spread out automatically to avoid overlap (one can also drag them to different positions in the plot itself).

Note that the C/E adjustment is also a useful option for grids with large numbers of constructs as the geometry of the analysis tends to project the element outside the hypercube formed by the constructs. Since only the relative direction of the construct dimensions is meaningful, not the absolute position of the poles, this scaling has no effect on the interpretation of the analysis.

The X and Y axis menus in the fourth row show the principal components with the percentage variance for which each accounts, and are used to select which components, if any, are plotted on the horizontal and vertical axes. The relative placement of the four quadrants is arbitrary, and the *Reverse* check boxes allow the plot to be reversed horizontally and vertically for greater perspicuity. These are useful options if multiple PrinGrid analyses are to be compared as they enable one to show the elements in similar quadrants in successive plots to the extent that this is possible.

The Z axis check box in the fifth row specifies that a three-dimensional plot should be produced, with the menu and *Reverse* checkbox specifying what component should be plotted on the Z axis and whether it should be reversed. The three values on the right specify the rotations in degrees of the X, Y and Z axes, respectively. The rotations are computed in the order Y, X, Z to keep the Y plots vertical if only X and Y rotations are used.

The check boxes in the sixth row determine whether: a scree plot should be produced; is titled; Frontier’s (1976) comparative plot of the distribution if the principal components were randomly generated is also plotted to provide an estimate of the number of significant components.

The text field on the right enables the geometry of the scree plot to be adjusted. It can contain up to 4 numbers separated by commas: whether the vertical numeric increments are 1 or 2; vertical separation between scale points; horizontal separation between scale points; horizontal space at each end of plot. If fewer than 4 values are specified the remainder are filled from the appropriate position in the default values 2, 3, 20, 10. In practice there is rarely a need to change the detailed geometry of the scree plot but the options are available to fine-tune the output for presentation or publication. Mousing over either the text field brings up help text giving information about available parameters.

The row of check boxes near the bottom determine whether a textual analysis is produced, and, if so, whether the percentage variances for the components are output, and whether the element and construct loadings on the components are output.

The menu at the bottom left specifies the metric to be used in calculating the data matrix for principal components analysis, either construct covariances or element distances (Minkowski—any power).



5.4.1 PrinGrid Map plot output

Figure 97 shows the plot produced for the grid of Figure 79 when one clicks the *PrinGrid* button with the settings above.

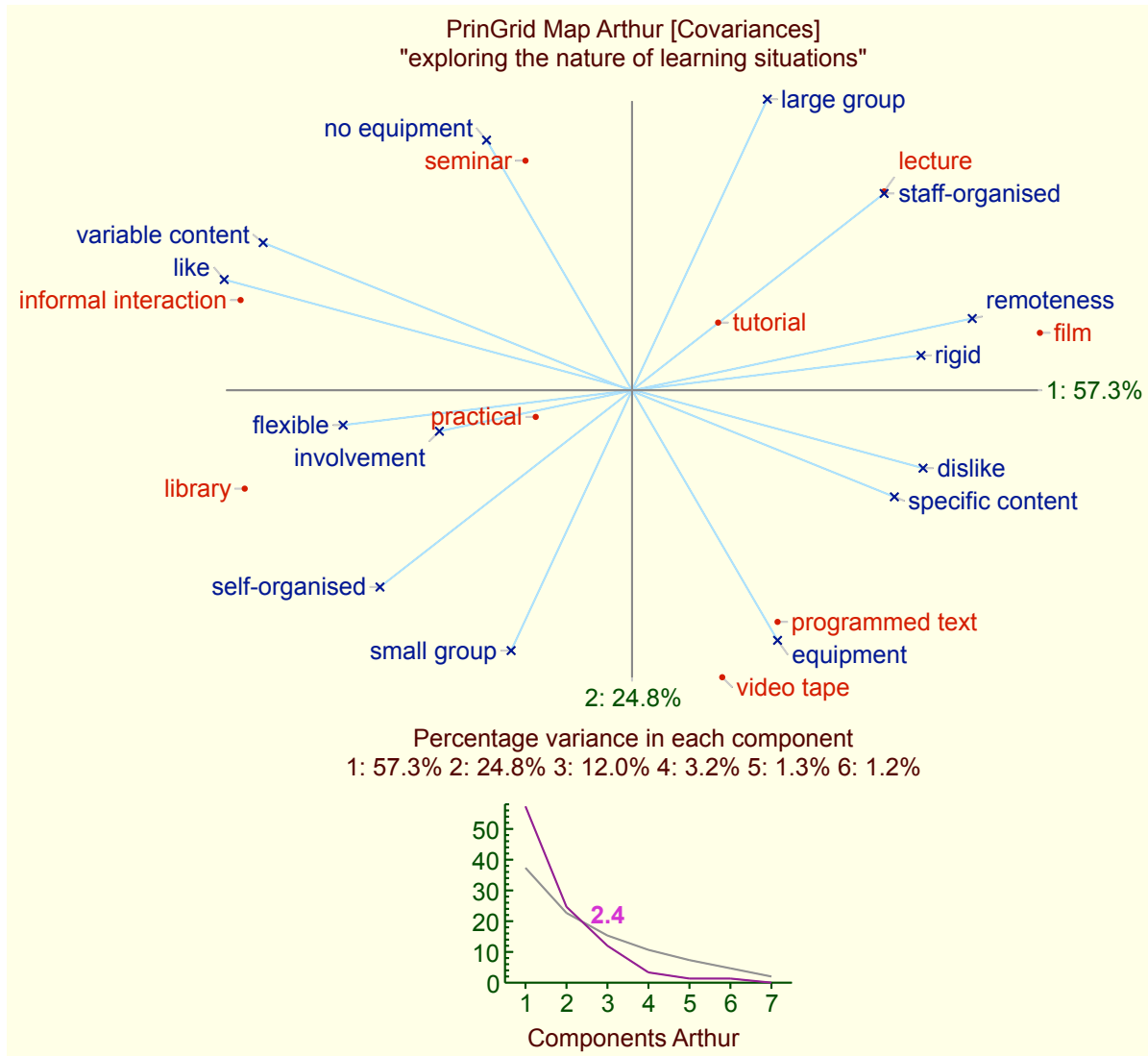


Figure 97: *PrinGrid Map* analysis

The plot opens in RepNet, and may be edited, annotated, and saved as a net and in graphic interchange formats. In particular, while the *spread* algorithm makes a reasonable attempt to place the element and pole names in a way that avoids overlap, the human eye may see more perspicuous locations. There are connecting lines between the labels and the points they label so that the linkage is apparent regardless of the label location.

The *scree* graph at the bottom plots the percentage variance accounted for by each component. The superimposed graph plots Frontier's (1976) estimated distribution if the principal components

were randomly generated which has been found to provide a good estimate of the number of significant components (Jackson, 1993). The number indicates this in terms of where the plots cross—when there is fractional value of significant components indicated the recommendation in the literature is to round down. However, if the estimate is used as a complexity indication the continuous scale of the fractional values may be useful.

Note that the comparative plot may cross the scree plot more than once indicating that there is more than one possible estimate of the number of significant components. There are continuing studies of such estimation techniques (Peres-Neto et al., 2005)—the literature is growing and the articles noted here provide useful search terms for such later literature as they are generally cited.

5.4.2 *PrinGrid Map with Voronoi diagram*

The interpretation of the *PrinGrid Map* plot generally involves noting: construct dimensions that intersect at a small angle and hence are similar in the context of the set of elements that have been construed (Kelly, 1969, p.105); elements that cluster together and hence are similar in the context of the constructs used to construe them; and the positioning of those clusters on the construct dimensions to gain understanding of the basis of the similarity.

This information is apparent visually in the conventional principal components analysis *biplot* (Gower et al., 2011) but there is no graphical plot supporting the visualization of element clusters as there is for *Focus*. Cluster analysis has long been a major research topic and there are a large number of approaches and algorithms to support it (Hennig et al., 2016), but most automated techniques are not simply explicable to those reflecting on their conceptual structures through grid analysis.

However, one readily-explained technique that supports visual understanding of a principal components biplot is to superimpose a *Voronoi diagram* (Okabe et al., 1992) that partitions the space to show locations that are nearer to one element location than to any other element location. This is a simple notion that dates back to Descartes and Dirichlet, and is easy to explain. However, computer algorithms to generate the diagram for any set of points proved difficult to design and prone to round-off errors for a variety of ill-conditioned configurations, and automatic generation of the diagrams was not implemented in the early grid analysis programs. Later research has resulted in a number of sound algorithms (Fortune, 1987) and computer implementations (Okabe et al., 1992).

Figure 98 shows the Voronoi partitions that PrinGrid superimposes on the plot when the *Voronoi* option is selected at the top right of the dialog (Figure 96). The options to wrap and centre the labels vertically above or below the locations were also selected to create a more compact appearance making it easier to place the element labels within their associated partitions. The *Focus* element tree from Figure 88 is shown at the right for comparison.

The way in which the elements cluster is now more readily perceived through the adjacency of their Voronoi partitions. For example, the highly related pairs on the left of the *Focus* tree are in adjacent partitions, and the linear sort of the tree appears as a two-dimensional path through adjacent Voronoi partitions. The relation between the partitions and the construct poles that characterize them is also apparent.

Gärdenfors (2000) has developed a theory of *conceptual spaces* that parallels Kelly's of *psychological space* in its psychological, philosophical and empirical foundations and techniques, and has

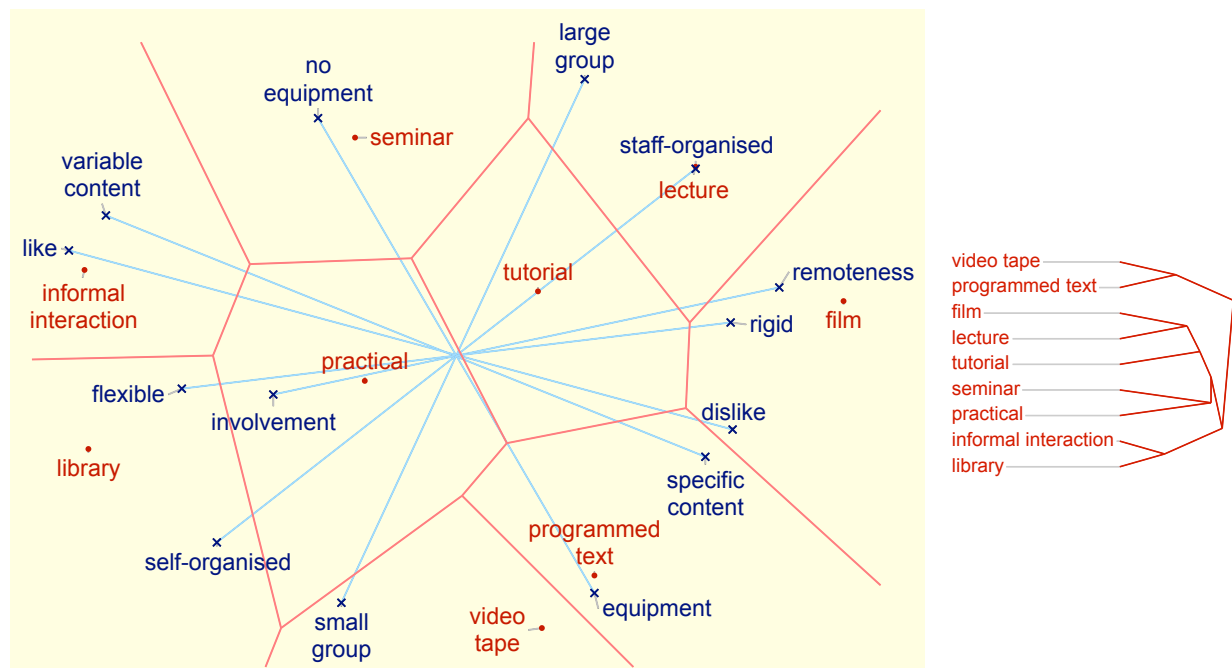


Figure 98: *PrinGrid Map* with Voronoi diagram (left) compared with Focus cluster (right)

argued that the Voronoi partition associated with an entity models its role as a conceptual *prototype* in Rosch's (1978; 1978) theory of categorization. There is significant ongoing research on conceptual spaces and their applications (Zenker and Gärdenfors, 2015; Kaipainen et al., 2019), much of which is relevant to personal construct psychology studies, even though the two areas have, so far, had little interaction.

5.4.3 *PrinGrid Map with alternative metrics*

The analysis of conceptual grids is largely based on distance measures calculated from the differences in ratings, but the major algorithms in common use are based on different measures, for example: construct covariances or correlations in Slater's (1976; 1977) *INGRID*; sum of absolute differences in Shaw's (1980) *FOCUS*; and chi-squared distances deriving from *correspondence analysis* (Greenacre, 2007, pp.177-181) in Feixas' *RECORD/GRIDCOR* (Feixas and Cornejo, 1996). Gower (1966) shows that distances matrices based on any of these, or other metrics complying the axioms for distance measures, may be embedded in Euclidean space and analyzed using principal components analysis.

There are interesting relationships between these measures: covariances are equivalent to a Minkowski distance of power 2.0; sum of absolute differences is equivalent to a Minkowski distance of power 1.0; the two may be seen as related by regarding the power of 2.0 as weighting a difference by itself making larger differences more significant; correlations may be regarded as covariances normalized by standard deviations making the spread of usage of a rating scale less relevant; and so on.

The different distance measures underlying the mainstream techniques for developing conceptual models from grid data may raise questions about why one measure is used rather than another,

what difference does it make, which is best, and so on. Part of the answer is in the design objectives of the developers: Slater and Shaw were concerned to develop presentations of the grid data that would be understandable to those from whom the grids were elicited and could be simply explained to them. Slater took Kelly's notion of elements in the space defined by constructs and rotated that space to create a map of the elements in the plan that captured as much as possible of the spatial relationships between them. Shaw sorted the grid to bring similar items together, and presented this as hierarchical trees around the grid itself. From Slater's and his clients' perspective the Euclidean metric was natural to the space being rotated. From Shaw's and her clients' perspective the simple sum of absolute differences was easy to understand in the sorted grid that is part of the Focus presentation.

In RepGrid alternative metrics are made available in the *Focus Cluster* (and associated analyses) and *PrinGrid Map*, with the default for Focus being Minkowski distances power 1.0, and that for PrinGrid being covariances (equivalent to Minkowski distances power 2.0 (Gower, 1966)). However, both analyses may be run for any Minkowski power, not because any nonstandard value is recommended, but so that researchers may experiment with different metrics if they wish, for example, to run a sensitivity analysis to find what features of a conceptual model, if any, are subject to significant variation as the metric is changed.

For example, Figure 99 shows *PrinGrid Map* analyses for Minkowski powers of 1.0, 2.0, 0.5 and 4.0. The first two are commonly used values and it can be seen that the component variances for 2.0 are the same as for covariances as in Figure 97 (as they must necessarily be). The unusual values of 0.5 and 4.0 show what happens as the power moves towards 0.0 where the elements will be equidistant, and towards infinity where the distance will be the maximum absolute difference over all constructs.

What is apparent, for this particular grid, is that the topology of the elements and their relations to the constructs changes little with this wide variation of the metric—the interpretation of the conceptual map is insensitive to the choices of metric shown. The same construct dimensions intersect at a small angle; the connectivity of the element Voronoi partitions is the same; and the positioning of those partitions on the construct dimensions is much the same.

It is possible to construct artificial grids where more substantial changes occur, and there are changes in the metric, for example those resulting from weighting constructs that can result in meaningful changes (§5.9). However, we have found for natural grids elicited to investigate anticipatory relations in a significant domain well-known to the elicitee, the analyses of Focus and PrinGrid closely track one another, and that this seems to continue to apply to an wider spread of metrics.

There is scope for research on the impact of metrics on the analysis and interpretation of conceptual grids, and Rep Plus provides technical support for this. In theoretical terms, one might precisify the interpretations guidelines noted above and analyze the impact of varying the metric on the angle of intersection of the constructs, connectivity of the Voronoi partitions, and their relation to the construct dimensions, or similar interpretive guidelines. Empirically, one might analyze the impact of different metrics on the actually interpretation of a large corpus of grid data.

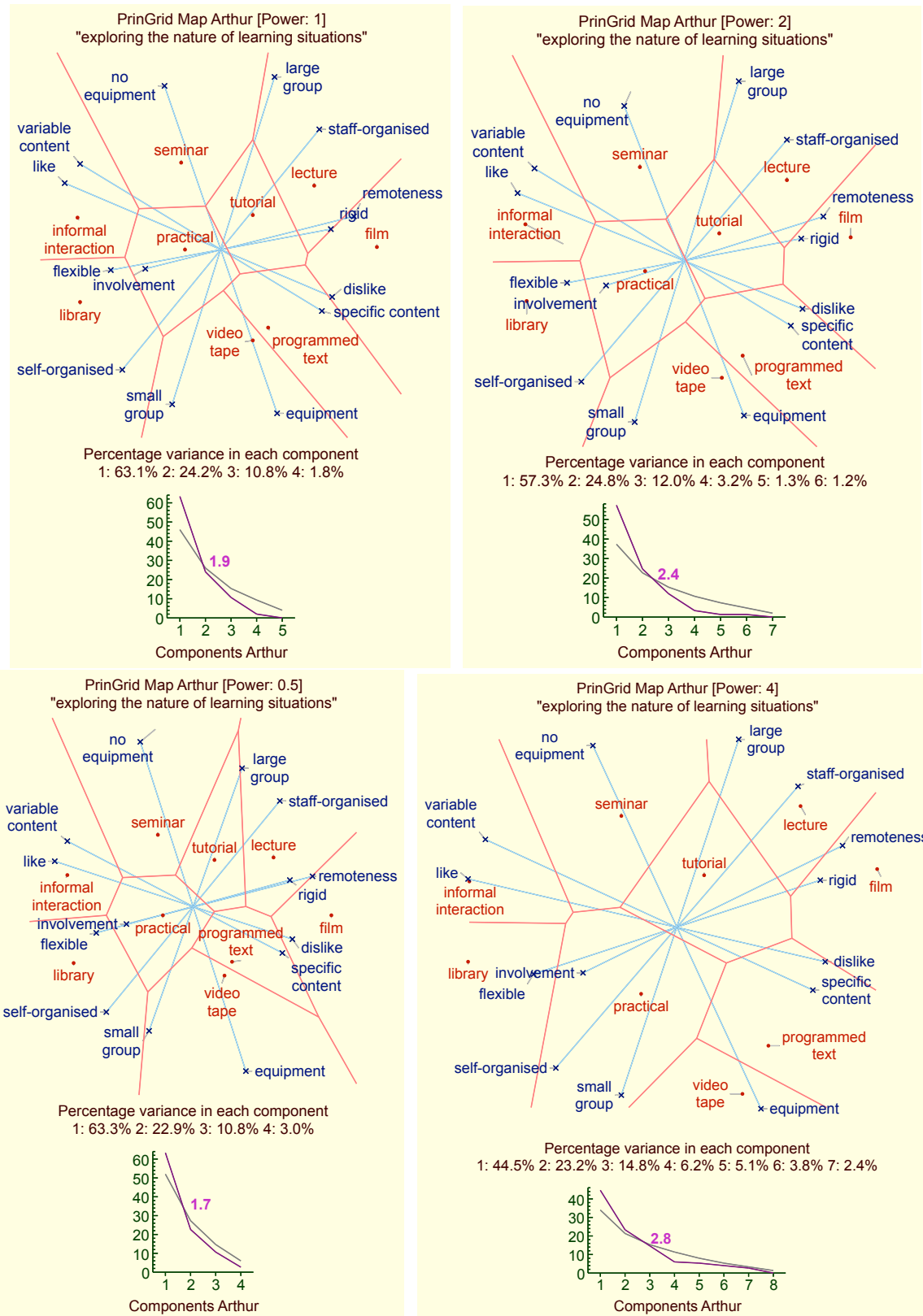


Figure 99: *PrinGrid Map* analyses with different Minkowski distance powers

Any such research needs to take into account both the ethos of personal construct psychology and the massive body of literature on the metrics of psychological space, and some of the interactions between them in Kelly's early studies preceding his 1955 book and his later commentaries on the nature of psychological space. Good starting points are Kelly's (1938) paper that analyzes Spencer's (1862, p.216) definition of *evolution* as "change from an indefinite, incoherent homogeneity to a definite, coherent heterogeneity" in mathematical and psychological terms, linking it to his discussion with Cyril Burt of *selection effects* in correlations and to Thurstone's (1935) use of factor analysis to study the "vectors of mind".

Kelly's student, Emmons (1939), extended his analysis with a critical study of factor analysis in psychology, and Kelly links it to his later research in an address to the *Moscow Psychological Society* in 1961 where he emphasizes that constructs are independent dimensions of psychological space that are brought into relationships as a selection effect of the elements chosen to populate that space in a particular context (Kelly, 1969, p.105)—construct systems create a *coherent heterogeneity* in Spencer's *incoherent homogeneity* or James' (1890, p.488) *buzzing confusion* of experience. Similarly, he emphasizes that there are no intrinsic *distances* in psychological space (Kelly, 1969, p.105), but these may be introduced to measure *similarities* between elements as counts of their *incidences* and *voids* with construct poles (Kelly, 1955, p.) (which can be extended to rating scales as structures constituted by multiple constructs (Gaines and Shaw, 2012, §3.4)).

The most substantial literature on the appropriate distance measures for psychological data is in studies of *psychophysics* from the 1930s to our era. The road map commences with Attneave's (1950) survey of *dimensions of similarity*, and proceeds through Torgerson's (1958) *theory and methods of scaling*, Shepard's (1964) *metric structure of stimulus space*, Tversky's (1977) *features of similarity*, Nosofsky's (1985) *identification of separable-dimension stimuli*, to Algorn and Fitousi's (2016) *half a century of research on Garner interference and the separability–integrality distinction*.

The psychophysics literature is useful in presenting the issues in the operationalization of the notion of *psychological distance* but is largely based on perceptual similarities of well-defined physical stimuli and it not clear to what extent any conclusions apply to the complex experiences represented as *elements* in personal construct psychology. That in itself is a major research topic.

There are also issues with the notion of similarity in the psychological literature that have interesting constructivist interpretations. James (1890, p.579) noted that an entity could be similar to a second entity in some respect and similar to a third in another respect, but the second and third entities may have no similarity—the constructs on which a psychological metric is based may tacitly change. Nosofsky (1985, p.427-430) explains finding a different metric for similarities than that which Shepard (1964) derived as Shepard's subjects using different perceptual information from that which he varied in designing the experiment—the experimenter-as-scientist and the subject-as-scientist may employ different construct systems.

5.4.4 PrinGrid Map with mixed construct types

The *PrinGrid Map* analysis may be applied to grids with any mixture of the rating scale types available in *Rep Plus*. For example, Figure 100 shows a plot produced for the grid of Figure 81. The multiple types of the constructs are not apparent, as they are in the Focus plot, because the actual ratings are not shown in a PrinGrid map as they are in a Focus clustering.

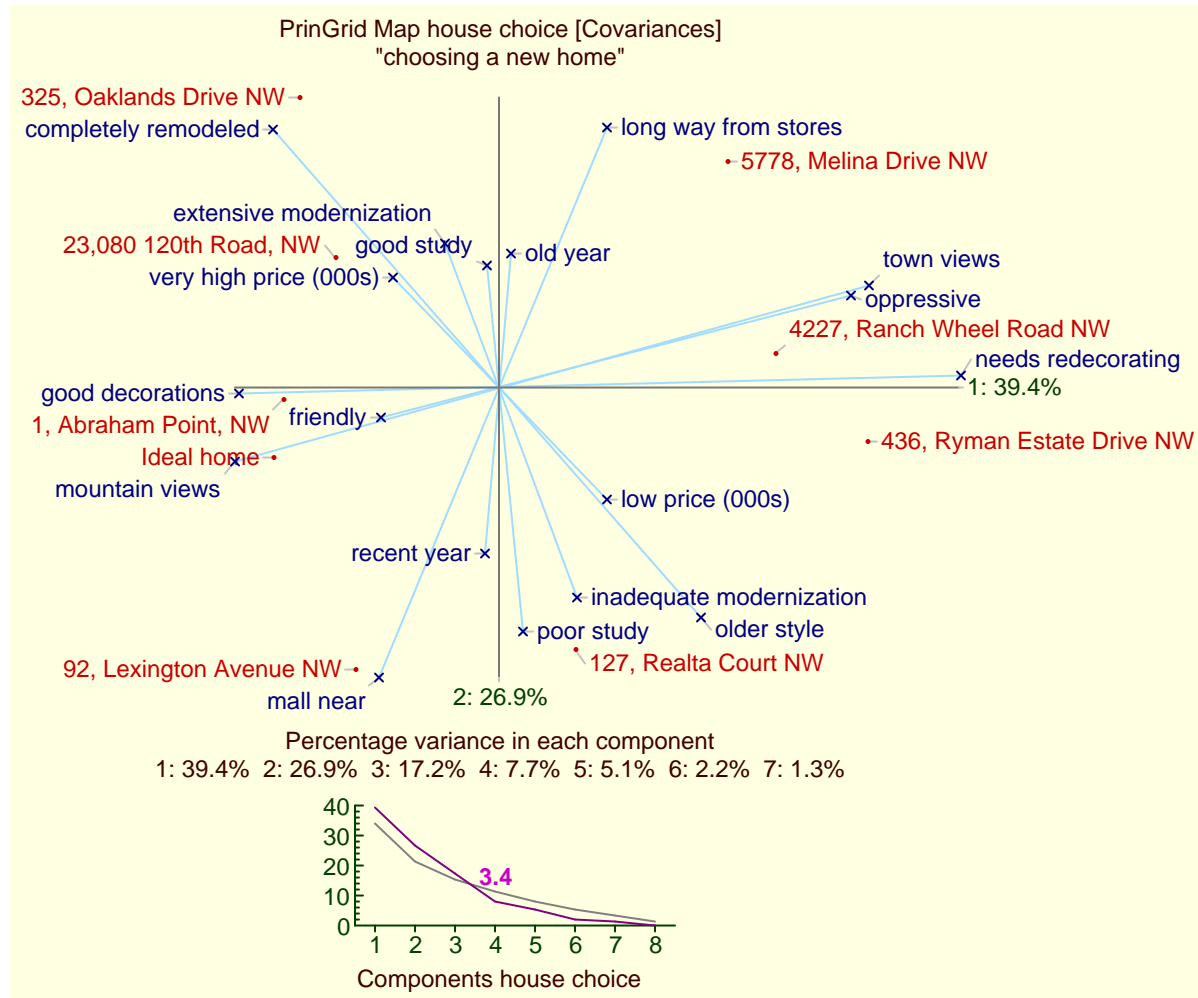


Figure 100: *PrinGrid Map* analysis of a mixed-type grid

5.4.5 PrinGrid 3D plot

It is sometimes of interest to show three components in what is essentially a two-dimensional section of a three-dimensional plot. Figure 101 shows the three-dimensional output generated when the check box to the left of the Z axis specification in Figure 96 is clicked. The X-Z plane is shown by the orange rectangle, and lines have been dropped from the element and construct pole positions to their coordinates in this plane.

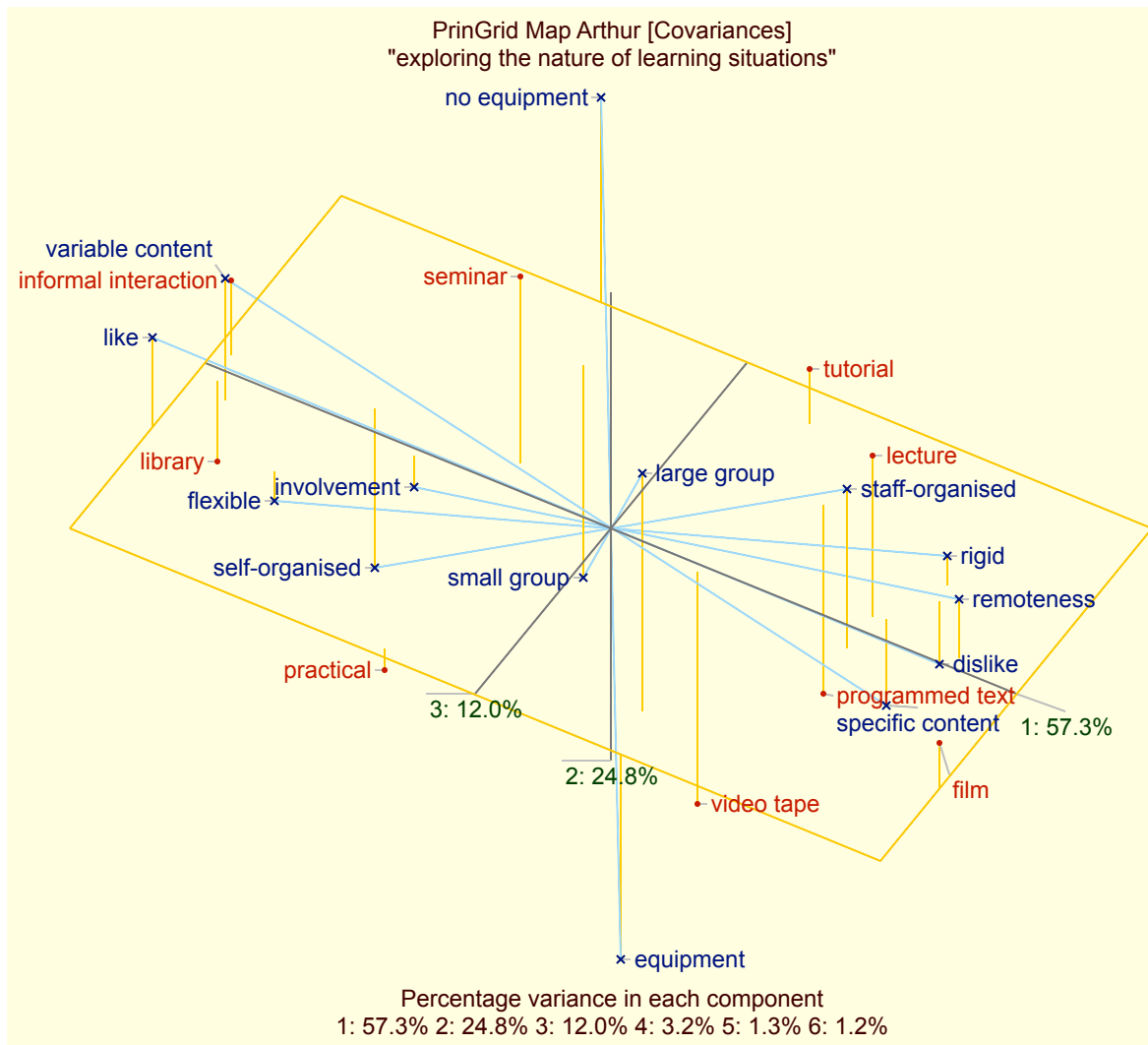


Figure 101: Three dimensional *PrinGrid Map* display

What component is shown on what is axis is arbitrary and may be varied for perspicuity, as may the angles of rotation of the three-dimensional plot before it is projected into two dimensions for display.

5.4.6 *PrinGrid text output*

Figure 102 shows the textual output of the data underlying the plot of Figures 97 produced when the *Data* checkbox at the bottom left of Figure 96 is checked, and below it the data underlying the plot of Figure 100. Note that the variance and Frontier estimate are shown for all the components, not just those above the specified cut-off, and also that, if comparing the data with other principal components analyses, the absolute values of the loadings produced are arbitrary and depend on how the grid data has been scaled—only the relative values are meaningful.

Percentage Variance in Each Component

	1	2	3	4	5	6	7	
57.28	24.83	12.03	3.22	1.28	1.18	0.18		Variance %
37.04	22.76	15.61	10.85	7.28	4.42	2.04		Frontier estimate %
57.28	82.11	94.14	97.36	98.64	99.82	100.00		Cumulative variance %

Element Loadings on Each Component

	1	2	3	4	5	6	
1	1.537	1.216	-0.115	0.304	0.227	-0.056	lecture
2	0.525	0.410	-1.201	0.004	-0.108	0.297	tutorial
3	-0.655	1.401	-0.178	-0.206	-0.058	0.164	seminar
4	-0.587	-0.159	1.385	-0.374	0.218	0.280	practical
5	2.485	0.349	0.815	0.019	-0.331	-0.244	film
6	-2.359	-0.603	0.087	0.083	-0.426	-0.013	library
7	0.883	-1.418	-0.716	-0.685	0.121	-0.198	programmed text
8	0.556	-1.750	0.046	0.670	0.123	0.137	video tape
9	-2.385	0.554	-0.121	0.185	0.234	-0.368	informal interaction

Construct Loadings on Each Component

	1	2	3	4	5	6	
1	1.809	0.380	-0.068	0.650	-0.394	-0.230	involvement – remoteness
2	1.956	0.242	-0.578	-0.835	-0.185	-0.223	flexible – rigid
3	-0.991	1.700	-1.605	0.147	0.159	-0.010	equipment – no equipment
4	1.704	1.331	0.177	-0.035	-0.168	0.510	self-organised – staff-organised
5	0.867	1.866	1.159	-0.034	0.303	-0.228	small group – large group
6	2.141	-0.863	-0.182	-0.007	0.275	0.169	variable content – specific content
7	2.368	-0.640	-0.531	0.299	0.282	-0.081	like – dislike

Percentage Variance in Each Component

	1	2	3	4	5	6	7	8	9	
40.86	25.04	18.63	7.71	5.08	1.83	0.77	0.08	0.00		Variance %
31.43	20.32	14.77	11.06	8.28	6.06	4.21	2.62	1.23		Frontier estimate %
40.86	65.90	84.53	92.24	97.33	99.16	99.92	100.00	100.00		Cumulative variance %

Element Loadings on Each Component

	1	2	3	4	5	6	
1	-1.159	1.794	-0.122	-0.105	-0.275	-0.259	325, Oaklands Drive NW
2	1.907	0.252	0.194	-0.161	-1.116	0.187	4227, Ranch Wheel Road NW
3	1.230	0.935	-1.208	-0.990	0.574	0.200	5778, Melina Drive NW
4	0.642	-1.033	1.591	-0.650	0.141	-0.506	127, Realta Court NW
5	-1.046	-1.740	0.391	-0.232	0.056	0.543	92, Lexington Avenue NW
6	2.272	-0.624	-0.622	1.142	0.422	-0.135	436, Ryman Estate Drive NW
7	-1.543	-0.558	-0.836	-0.177	0.247	-0.193	1, Abraham Point, NW
8	-0.667	1.489	1.539	0.647	0.427	0.272	23,080 120th Road, NW
9	-1.635	-0.515	-0.926	0.526	-0.476	-0.110	Ideal home

Construct Loadings on Each Component

	1	2	3	4	5	6	
1	1.018	2.246	0.714	-0.691	-0.280	0.249	mall near – long way from stores
2	2.221	0.476	-0.367	-0.169	0.351	-0.097	mountain views – town views
3	1.521	0.000	-1.158	0.335	0.834	0.029	friendly – oppressive
4	0.082	-1.234	0.624	-1.542	0.412	-0.105	good study – poor study
5	1.454	-1.699	0.330	0.137	-0.714	-0.217	completely remodeled – older style
6	-2.510	0.111	0.370	0.092	0.253	0.111	needs redecorating – good decorations
7	0.661	-0.612	2.085	0.586	0.676	0.222	extensive modernization – inadequate modernization
8	-0.638	1.035	0.508	0.100	0.321	-0.792	low price (\$00s) – very high price (\$00s)
9	-1.054	-0.542	-1.132	-0.381	0.347	0.126	old year – recent year

Figure 102: Data from the *PrinGrid Map* principal components analyses of the grids

5.4.7 PrinGrid analysis of hierarchical data

It is interesting to compare Focus and PrinGrid analyses of the hierarchical data structures analyzed in §5.3.3 because, in both cases, the number of independent constructs used is known in advance. Figure 103 shows PrinGrid plots in 2 and 3D for the simple artificial hierarchy with seven constructs of Figure 92. The second and third components are equal, and can be seen on the left that a plot of the first two components does not adequately discriminate between the elements, but on the right that a plot of all 3 components clusters the elements appropriately but not as clearly as in the Focus plot.

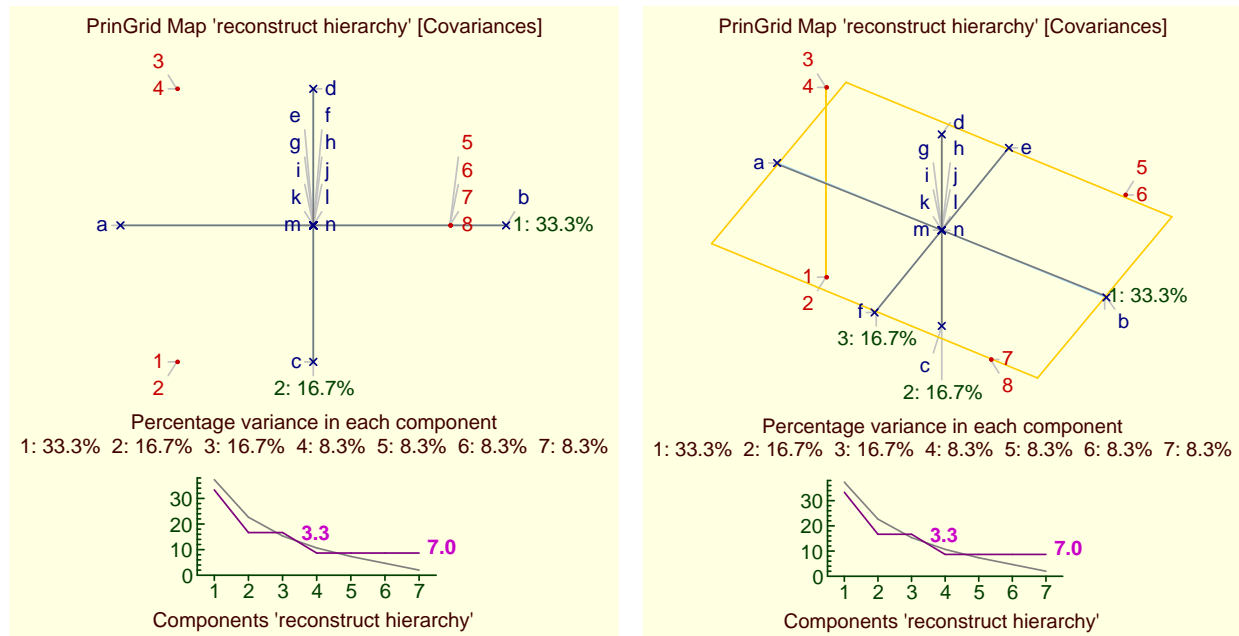


Figure 103: *PrinGrid* analysis of an artificial hierarchy in 2 and 3 dimensions

Frontier's (1976) comparative plot of the distribution two intersections indication that 3 components may reasonably account for the data but 7 are better. In the 3D plot, the correspondence of the axes to the *a—b*, *c—d* and *e—f* constructs shows that the plot accurately represents the higher levels of the hierarchy, but the lack of representation of the four lowest level constructs indicates that a 3D plot cannot capture all of them, as would be expected with the artificial 7-dimensional data.

Figure 104 shows a comparison of PrinGrid and Focus analysis of the natural hierarchy of Internet services in the 1990s analyzed in §5.3.3. The Frontier plot indicates that 2D and 12D analyses are significant, picking up the 2 major constructs differentiating the Internet services, *access—awareness* and *computer-mediation communication—services*, as well as the 12 constructs fully differentiating them. The element clusters of the Voronoi diagram correspond to those in the Focus element tree. Both forms of presentation make apparent the relations between the elements but through different visualizations—they complement one another.

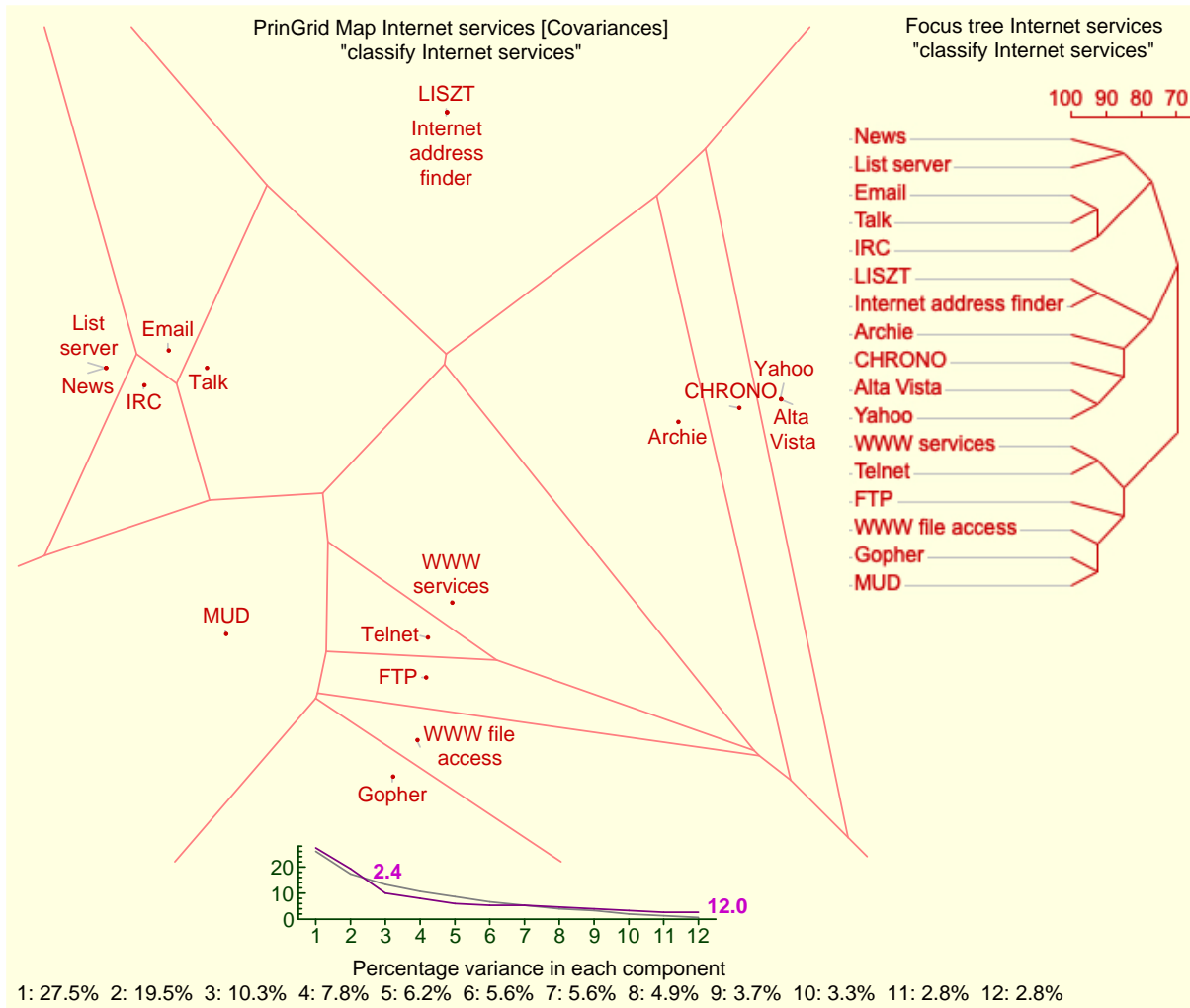


Figure 104: *PrinGrid* analysis of a natural hierarchy compared with *Focus* analysis

5.5 Crossplot: Plotting elements on constructs as orthogonal axes

Quadrant diagrams that show two orthogonal constructs as axes together with a set of elements plotted in the four quadrants thus created are commonly used in many literatures. The **Crossplot** tool in RepGrid allows two or three constructs to be selected as axes and the elements to be plotted in 2D or 3D respectively according to their ratings on the selected constructs,

Clicking on button icon on the right of the **Crossplot** button brings up a dialog which allows the constructs for a crossplot to be selected (Figure 105). The options are similar to those for PrinGrid except that the axes are constructs not components.

The row of check boxes at the top determine whether: the plot is titled; whether the elements and constructs are numbered; whether the element labels are spread to prevent overlap; what margins will be added for a Voronoi diagram; the scale of the plot.

Crossplot 9 situations

☒ Title ☐ Numbers ☒ Spread Margins Scale

Elements ☐ Notes ☐ Wrap ☐ Vertical ☐ Voronoi

Constructs ☐ Notes ☐ Wrap

X ☐ Reverse

Y ☐ Reverse

Z ☐ Reverse

☐ 3D Rotate °X °Y °Z

Figure 105: RepGrid *Crossplot* dialog

The row of check boxes on the second row determine whether: element notes should be shown; element labels should wrap; labels should be vertically above or below their location; whether a Voronoi diagram for the element locations should be plot.

The row of check boxes on the third row determine whether: construct notes should be shown; construct labels should wrap.

The fourth row allows the construct for the X axis to be selected, followed by a check box determining whether it is reversed.

The fifth row allows the construct for the Y axis to be selected, followed by a check box determining whether it is reversed.

The sixth row allows the construct for the Z axis to be selected, followed by a check box determining whether it is reversed.

On the bottom row, the 3D check box specifies whether a three-dimensional plot will be produced and numbers in the three text boxes following determine the rotations of the 3D plot.

Figure 106 shows the plot produced when one clicks the *Crossplot* button with the settings above.

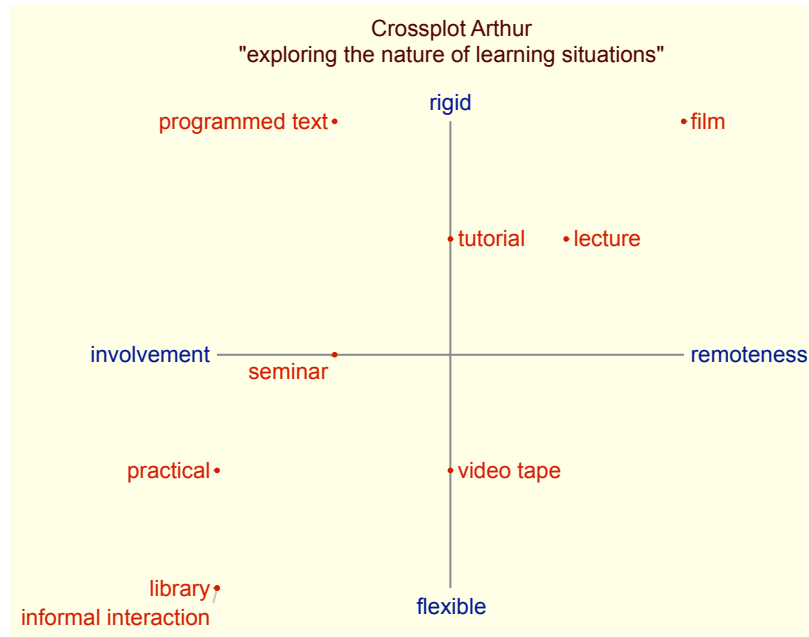


Figure 106: *Crossplot* of the elements on two selected constructs

Figure 107 shows the three-dimensional output generated when the *3D* check box is also set.

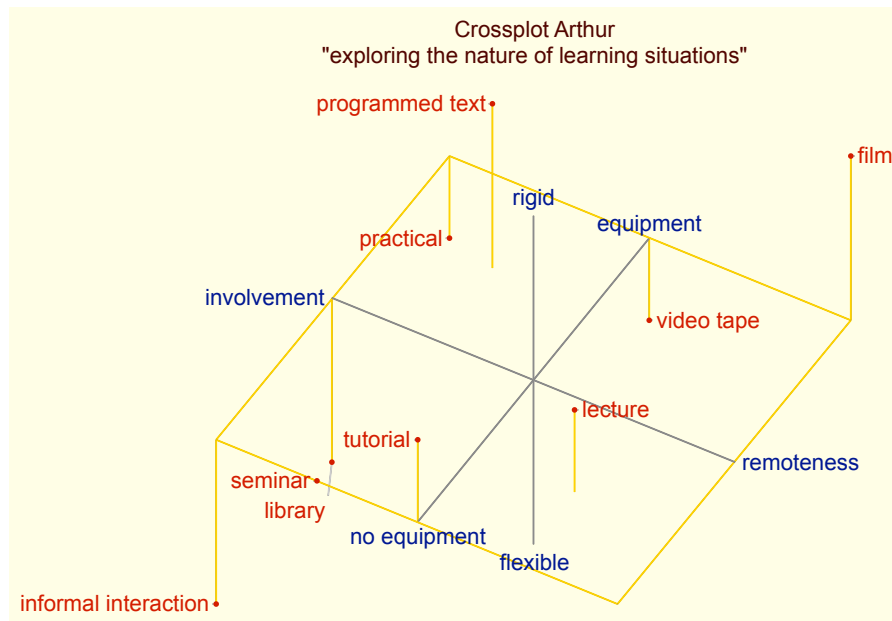


Figure 107: *Crossplot* of the elements on three selected constructs

Crossplots are useful visual presentations of grid data in their own right, and also useful in having users come to understand the PrinGrid plots as rotations of multi-dimensional crossplots.

5.6 Compare: Comparison of grids with some common elements and/or constructs

When two grids were elicited from the same person or from two members of the same community who are expected to use similar terminology to identify events and shared concepts, if they have have elements and/or constructs in common it is possible to be able to compare them for similarities and differences in the use of constructs and the construing of elements. The *Compare* tool in Rep-Grid provides a graphic comparison of such grids based on the *MINUS* algorithm of Shaw (1980) extended to grids having a diversity of rating scales including multiple types.

Note that determining common elements across grids is based on lexical equality of the element names, and common constructs on the lexical equality of the pole names and construct names (if any). Hence a grid being compared should not have two or more elements with the same name or two or more constructs that are equal on the above criterion. **The supposition that lexically equivalent elements and/or constructs are intended by the elicitee(s) to have the same meaning needs careful consideration and justification if the analysis is to be meaningful.** This can be addressed by discussions with the elicitees or a focus group representing them or managing the study.

5.6.1 Methodology of grid comparison

The methodological basis of grid comparison is illustrated in Figure 108 where two grids overlap in a central region of common elements/constructs and three pairs of peripheral regions having: common elements but different constructs; common constructs but different elements; and different elements and constructs.

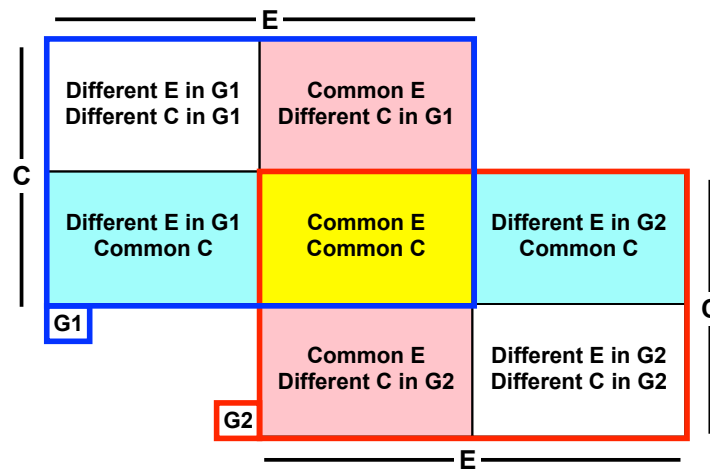


Figure 108: Pairwise comparison of grids with some common elements and/or constructs

The differences in ratings between the two grids in the common subgrid in the centre may be used to determine the degree of *consensus* and *conflict* (Shaw and Gaines, 1989) of the elicitees for each element and each construct represented in the subgrid. In essence, the common constructs are used to match the common elements, and the common elements are used to match the common constructs.

In the regions having some commonality this can be used to enable the items that are lexically different in one grid to be matched against all the items of that type in the other grids and the best

matches can be displayed. This is achieved by analyzing the subgrids comprising the central subgrid combined with each of the four peripheral grids to its left and right and above and below it. In essence: the common constructs in the central grid are used determine the best match for any element in G1 of an element in G2, and *vice versa*; the common elements in the central grid are used determine the best match for any construct in G1 of an construct in G2, and *vice versa*.

Figure 109 shows three common instances of the general schema shown in 108. On the left, grids having both elements and constructs in common arise in many different ways, as a temporal sequence from one individual, developing a grid at one time and then rerating after a learning experience, or through an exchange of grids where two or more people develop grids supplying their own elements and constructs and then exchange them with one another so that each person rates the other's elements on the other's constructs, and so on. In the centre, grids having common elements but different constructs arise when two or more individuals develop grids using a commonly agreed set of elements and wish to see the similarities and differences in their individual constructs. On the right, grids having common constructs but different elements arise when two or more individuals develop grids using a commonly understood set of public constructs and wish to see the similarities and differences in their individual experiences in terms of these constructs.

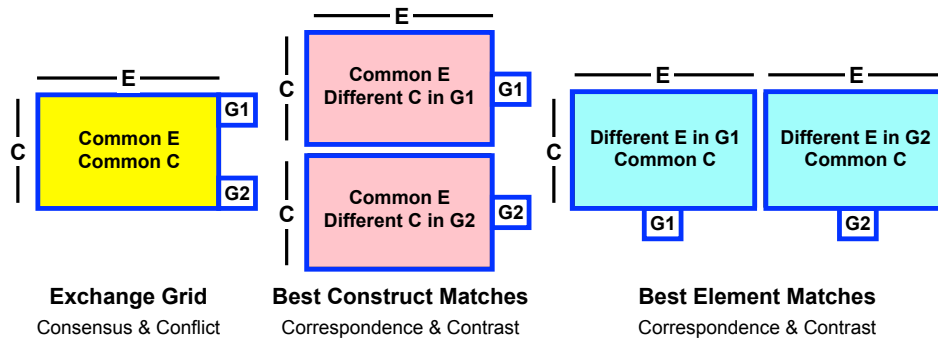


Figure 109: Some special cases of pairwise comparison of grids

The following subsections illustrate how the Compare tool analyses these three types of data, including grids instantiating the more situation shown in Figure 108 where all three forms of analysis may be possible. There are additional analyses possible for research designs where large numbers of grids have been elicited with the various forms of overlap discussed above, or even when there is no overlap but only a common domain of interest, and these are covered in the *RepGrids* manual.

5.6.2 The compare dialogue

Clicking on button icon on the right of the *Compare* button brings up a dialog which allows the current grid to be compared with a another selected one (Figure 110). The *Open Grid* button at the bottom left is highlighted, and the *Compare* button on the left is disabled, because one needs to open a second grid for comparison before proceeding with the analysis. Clicking on the *Open Grid* button brings up a standard file open dialog where one can open a secondary grid for comparison. One may also drag a grid file to the dialogue to open it for comparison.

Note that the *Select* check boxes in the *Element* and *Construct* panes may be used to restrict the items in the primary grid that are used in the comparisons. This enables a subgrid to be used in a

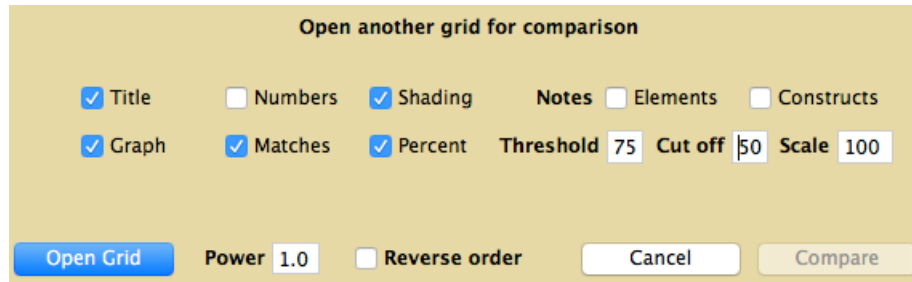


Figure 110: RepGrid initial *Compare* dialog

comparison and dynamically adjusted as a result of the comparison. This capability may be used, for example, to implement Shaw's (Shaw, 1980, p.77) *CORE* algorithm to identify core constructs in a set of grids. In particular, the WebGrid port of the Compare functionality provides an interactive user interface that makes this a very simple activity.

5.6.3 Comparing grids with substantial numbers of elements and constructs in common

Figure 111 shows the dialog when a grid having both elements and constructs in common is opened. The text at the top identifies the grids being compared and the number of elements and constructs they have in common.

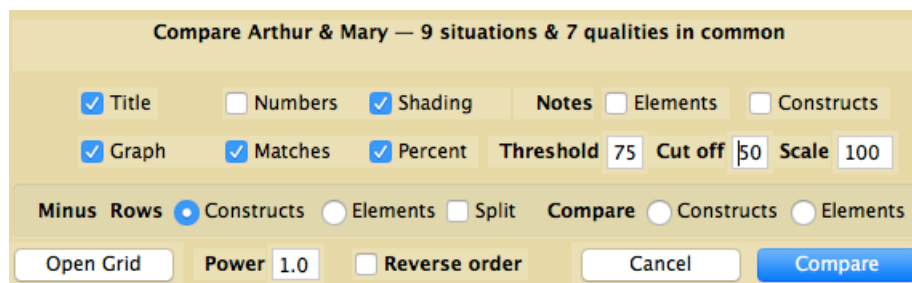
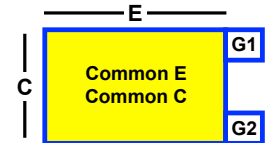


Figure 111: *Compare* dialog—grid with same elements and constructs opened for comparison. The first row of check boxes determine whether: the plot is titled; elements and constructs are numbered; high and low differences are shaded; notes attached to elements and constructs are shown.

The second row determines whether matches will be graphed; match values will be shown; the cumulative percentage of matches above or equal to the match value will be shown; and specifies the threshold, cut off, and scale. The *Threshold* value determines where in the plot matches will be shown as below threshold. The *Cut off* value specifies the lowest match that will be plotted. The *Scale* value determines how much space will be allocated to the graph of matches.

The panel below is only visible if the grids being compared have both common elements and common constructs. The four radio buttons select whether a *Minus* plot of the difference grid is required and, if so, whether the rows should be constructs or elements, or whether a *Compare* plot is required where the best matching constructs or elements are shown. The *Split* check box determines whether a *Minus* plot shows the difference in ratings or the actual ratings on separate lines (*Compare* plots when only elements or only constructs are in common always use separate lines).

In the bottom row, the *Power* value determines the exponent used in the Minkowski metric used to compute matching scores as discussed in §5.3. The normal mode of comparison is that the primary grid is compared with the secondary grid, and the *Reverse order* check box reverses this.

Figure 112 shows the plot produced when one clicks the *Compare* button with the settings above.

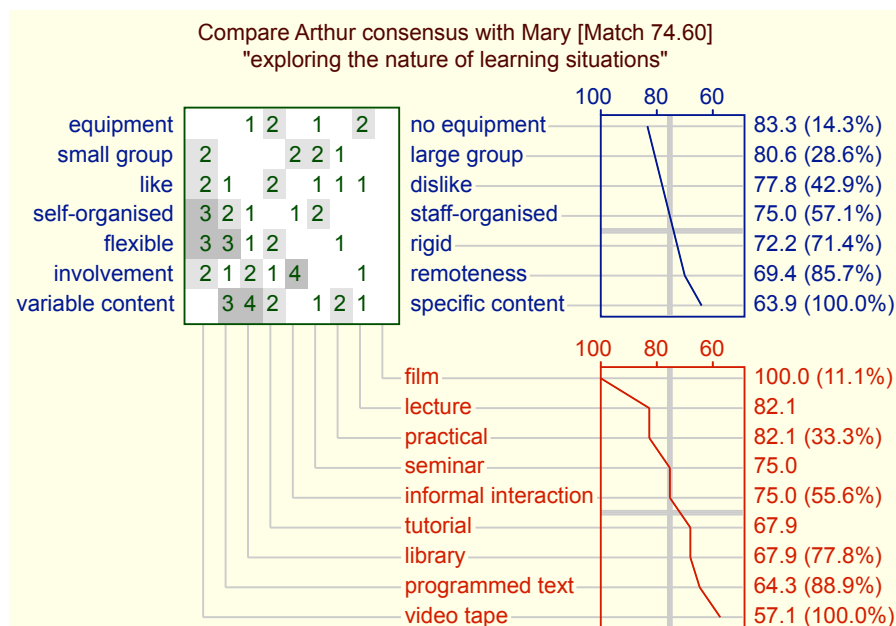


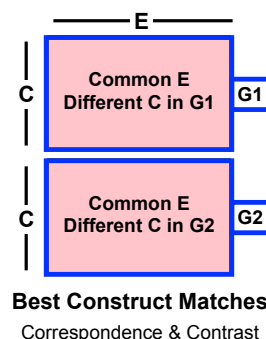
Figure 112: Comparing two grids with the same elements and constructs

On the left is shown the absolute difference between the ratings in the two grids with the constructs and elements sorted so that those that are most similar in the two grids are the top and on the left, respectively. The number on the right of the title is the overall match between the two grids. If the power is set to 1.0 it corresponds to both the mean construct match and the mean element match.

The graphs on the right provide a plot of the individual construct and element matches between the grids being compared. The numbers on the right show the numeric match value and, in parentheses, the cumulative percentage of items matching at that value or greater.

5.6.4 Comparing grids with a substantial number of elements in common

Figure 113 shows the *Compare* dialog when a grid having the same elements but different constructs is opened for comparison. Figure 114 shows the plot produced when clicks on the *Compare* button with the settings above. Each of the constructs in the primary grid is shown with the best matching construct in the secondary grid on the line below it. Element differences have also been computed based on the two grids of matching constructs, and constructs and elements have been sorted so that those that are most similar in the two grids are the top and on the right, respectively.



Compare Arthur & Mary — 9 situations & 0 qualities in common

☒ Title ☐ Numbers ☒ Shading Notes ☐ Elements ☐ Constructs
☒ Graph ☒ Matches ☒ Percent Threshold 75 Cut off 50 Scale 100

Open Grid Power 1.0 ☐ Reverse order Cancel Compare

Figure 113: *Compare* dialog—grids with same elements but different constructs

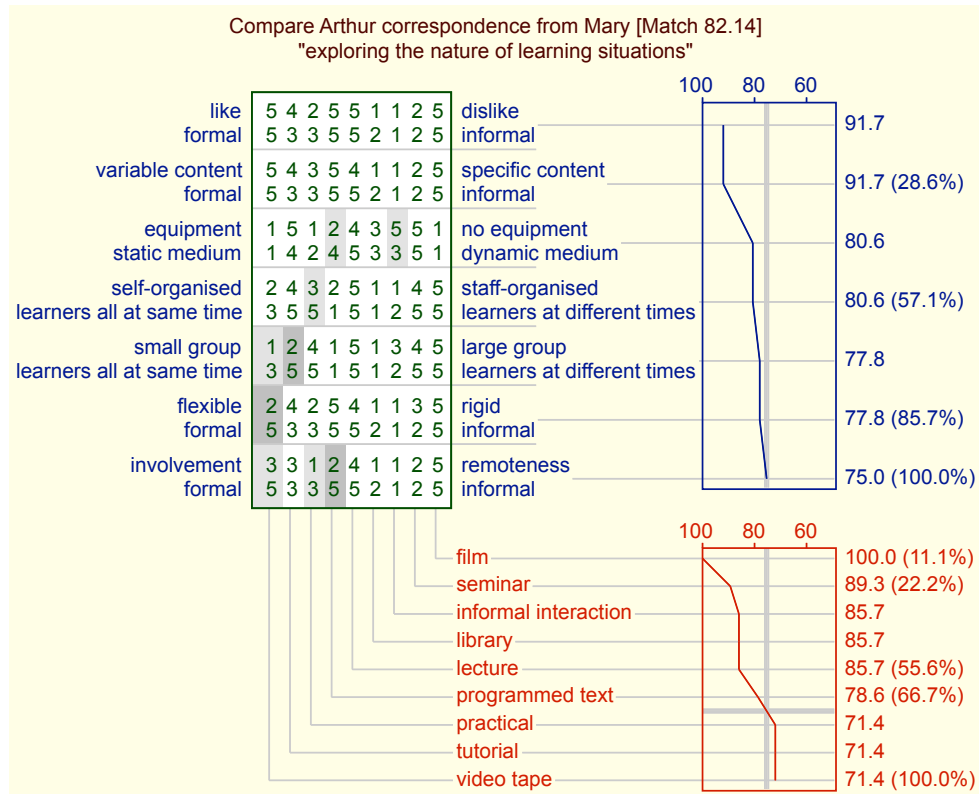


Figure 114: Comparing two grids with the same elements

The plot shows the best match in Mary's grid for each construct in Arthur's grid. If the *Reverse order* check box is set then a similar plot showing the best match in Arthur's grid for each construct in Mary's grid will be produced.

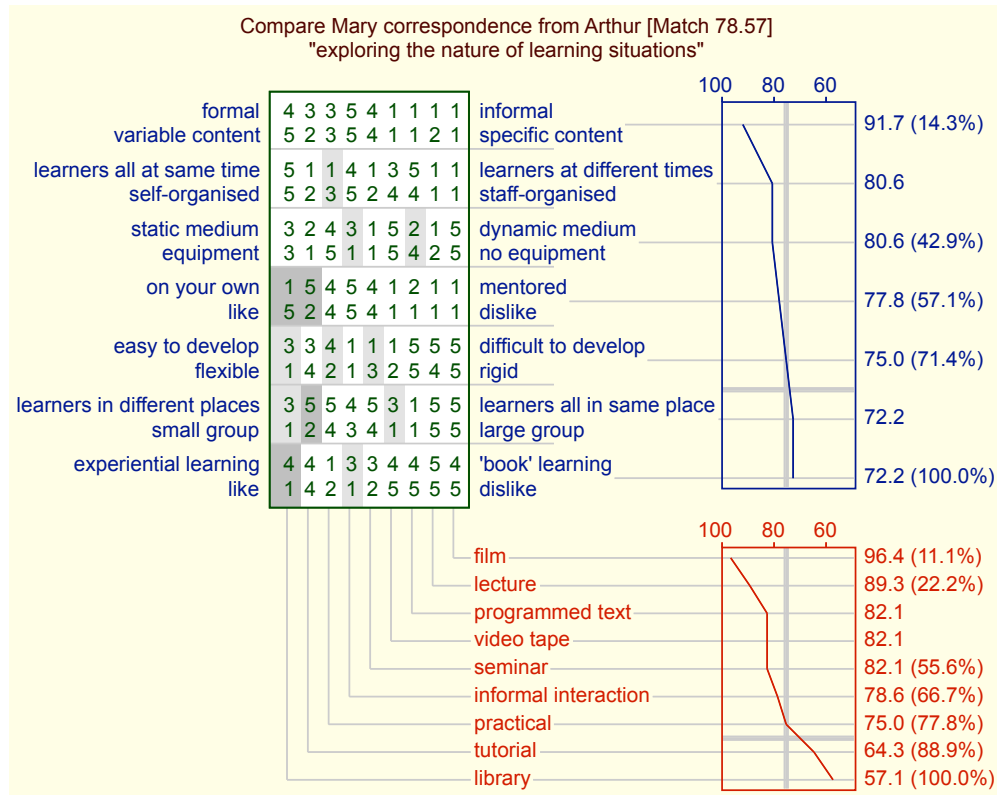


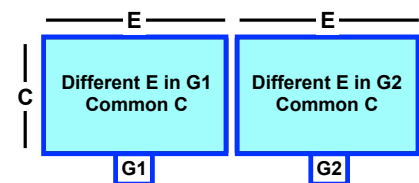
Figure 115: Comparing two grids with the same elements in reverse order

The reduced overall match value indicates that Mary's constructs enable her to better understand Arthur's than *vice versa*, and the effect of this is apparent in the construct and element match plots.

5.6.5 Comparing grids with a substantial number of constructs in common

Figure 116 shows the *Compare* dialog when a grid having the same constructs but different elements is opened for comparison. Figure 117 shows the plot produced when one clicks on *Compare*. Each of the elements in the primary grid is shown with the best matching element in the secondary grid on the line below it. Construct differences have also been computed based on the two grids of matching elements, and elements and constructs have been sorted so that those that are most similar in the two grids are the top and on the right, respectively.

The *Reverse order* check box may be used to determine the best matches for Paul's elements in Arthur's grid.



Compare Arthur & Paul — 0 situations & 7 qualities in common

☒ Title
 ☐ Numbers
 ☒ Shading
 Notes
 ☐ Elements
 ☐ Constructs

☒ Graph
 ☒ Matches
 ☒ Percent
 Threshold 75
 Cut off 50
 Scale 100

Open Grid
 Power 1.0
 ☐ Reverse order
 Cancel
 Compare

Figure 116: *Compare* dialog—grids with same constructs but different elements

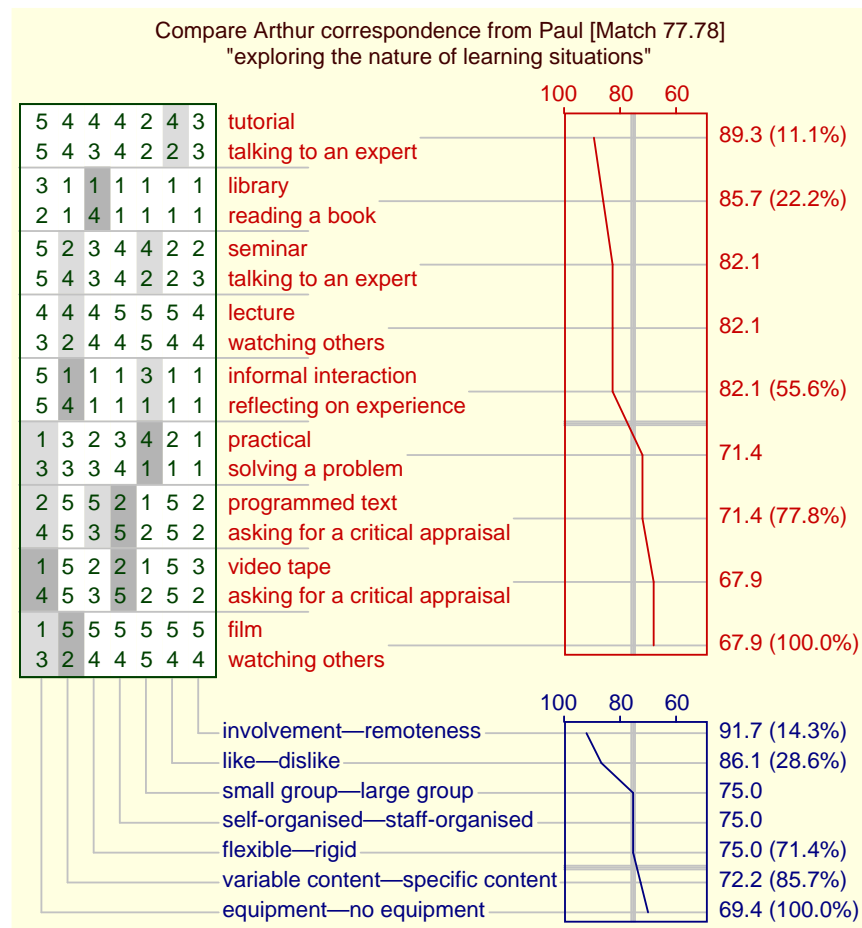


Figure 117: Comparing two grids with the same constructs

When a grid has a substantial number of elements and constructs the same, as discussed in §5.6.3, all three forms of match discussed above are possible. As shown in Figure 111, a panel appears in the *Compare* dialog box that allows the grid to be compared in terms of its common elements only (as in §5.6.4), its common constructs only (as in §5.6.5), or its common elements and common constructs (as the default in §5.6.3).

5.7 Match analysis: Display matches between elements and between constructs

It is often useful to be able to view the matches between selected elements or constructs in reverse order, for example, to see all the matches with an *ideal element* or a particularly significant construct. Clicking on button icon on the right of the *Matches* button brings up a dialog which allows matching elements and matching constructs to be displayed (Figure 118).

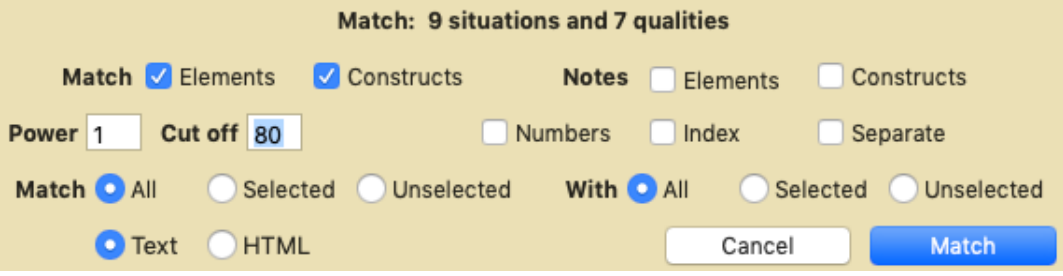
The image shows a 'Match' dialog box with a title bar that reads 'Match: 9 situations and 7 qualities'. The dialog is organized into several rows of controls. The first row contains two checked checkboxes for 'Match' (Elements and Constructs) and two unchecked checkboxes for 'Notes' (Elements and Constructs). The second row features input fields for 'Power' (set to 1) and 'Cut off' (set to 80), followed by three unchecked checkboxes: 'Numbers', 'Index', and 'Separate'. The third row contains two sets of three radio buttons; the first set is for 'Match' (All, Selected, Unselected) and the second is for 'With' (All, Selected, Unselected), both with 'All' selected. The bottom row has two radio buttons for 'Text' (selected) and 'HTML', and two buttons labeled 'Cancel' and 'Match'.

Figure 118: RepGrid *Match* dialog

The first row of check boxes determine whether: matches are output for elements, constructs, or both; element and construct notes are shown.

The second row specifies: the *Power* of the exponent used in the Minkowski metric used to compute matching scores (§5.3); the *Cut off* match value below which matches will not be shown; whether element and construct numbers will be shown; whether the Honey index will be shown (Honey, 1979; Jankowicz, 2004); and whether the matches are separated by element or construct, or all sorted together.

The two sets of three radio buttons in the third row select whether: all, selected or unselected items are matched with all, selected or unselected items. The selection of elements and constructs is set up in the *Elements* and *Constructs* panes, respectively (and is independent of the state of the *TextAnalyze Select* check boxes).

The two radio buttons at the beginning of the bottom row select whether the matches are shown in a text window or as an HTML table in your browser window.

Figure 119 shows the output produced when one clicks on the *Matches* button with *Text* or *HTML* selected.

Match: Arthur

Matches between situations, at least 80%

Situations	M%
library	85.7
informal interaction	
programmed text	82.1
video tape	

Matches between situations (at least 80%)

library informal interaction	85.7%
programmed text video tape	82.1%

Matches between qualities (at least 80%)

Matches between qualities, at least 80%

Qualities	M%
variable content — specific content	94.4
like — dislike	
flexible — rigid	86.1
like — dislike	
flexible — rigid	83.3
variable content — specific content	
flexible — rigid	80.6
self-organised — staff-organised	
involvement — remoteness	80.6
flexible — rigid	
self-organised — staff-organised	
small group — large group	
involvement — remoteness	
self-organised — staff-organised	

variable content—specific content like — dislike	94.4%
flexible — rigid like — dislike	86.1%
flexible — rigid variable content—specific content	
flexible — rigid self-organised — staff-organised	83.3%
self-organised — staff-organised small group — large group	80.6%
involvement — remoteness self-organised — staff-organised	
involvement — remoteness flexible — rigid	

Figure 119: Element and construct matches—left text, right HTML

Figure 120 shows the *Match* dialog set up to show the matches Honey index between the construct *like—dislike* which has been selected and all the other constructs.

Match: 9 situations and 7 qualities

Match ☐ Elements ☒ Constructs **Notes** ☐ Elements ☐ Constructs

Power 1 **Cut off** 0 ☐ Numbers ☒ Index ☒ Separate

Match ☐ All ☒ Selected ☐ Unselected **With** ☒ All ☐ Selected ☐ Unselected

☒ Text ☐ HTML

Figure 120: Selecting construct matches with Honey indices

Figure 121 shows the output produced when one clicks on the *Matches* button.

Match Arthur		Match: Arthur			
Matches between qualities (at least 0%)		Matches between qualities			
like — dislike		Quality	M%	T	R
variable content — specific content	94.4% H 1	like — dislike			
flexible — rigid	86.1% H 2	variable content — specific content	94.4	H	1
involvement — remoteness	77.8% I 3	flexible — rigid	86.1	H	2
self-organised — staff-organised	75.0% I 4	involvement — remoteness	77.8	I	3
no equipment — equipment	63.9% L 5	self-organised — staff-organised	75.0	I	4
large group — small group	61.1% L 6	equipment — no equipment	63.9	L	5
		small group — large group	61.1	L	6

Figure 121: Construct matches with Honey indices—left text, right HTML

Honey (1979) defined his index for purposes of content analysis and Jankovicz (2004) proves a detailed exposition of his method. The *RepGrids* tool for analyzing multiple grids in Rep Plus provides computational support for carrying out such content analyses including the use of indicators such as matches, mode scores, and Honey indices.

5.7.1 Using ideal elements derived from classes in a Match analysis

The *Classes* pane provides the option to include ideal elements derived from the classes in the grid supplied to any analysis program. Doing so with the *Matches* analysis provides useful insights into how the use of ideal elements entered directly may be used to solve a decision problem, and what further contribution is made through the extended properties available to specify intersects as classes.

If the *Include* checkbox in the *Classes* pane is checked for the contact lens prescription grid as discussed in §5.1.2 and the *Match* dialog is brought up it shows 32 elements, the 24 elements in the grid plus the 7 ideal elements generated from the classes. The ideal elements are automatically selected when they are added so that requesting the 100% matches of all the separated unselected elements with the selected ones will show what classes match each of the 24 cases. In addition the *Constructs* pane has been set to include all the constructs other than the lens prescription in the matches.

Match: 33 contact lens clients and 4 significant features (from 5)

Match ☒ Elements ☐ Constructs Notes ☒ Elements ☐ Constructs

Power Cut off ☐ Numbers ☐ Index ☒ Separate

Match ☐ All ☐ Selected ☒ Unselected With ☐ All ☒ Selected ☐ Unselected

☒ Text ☐ HTML

Figure 122: Match dialog to match each contact lens case against ideal elements from classes

Figure 123 shows the matches in two columns. The classes *Consider soft* and *Prescribe soft* generate the same ideal element with the only difference being in the comment field of the latter, *prefer*

exception soft or exception reduced, and similarly for consider and prescribe hard. Consequently every case is matched against a consider/prescribe pair but, as the comments indicate, the prescribe match should not be taken into account if there is an exception match.

Matches between contact lens clients (at least 100%, based on selected significant features)		
no lens 1		
lens infeasible	100.0%	
soft lens 1		
lens feasible soft feasible prescribe soft (prefer exception soft)	100.0%	
no lens 2		
lens infeasible	100.0%	
hard lens 1		
lens feasible hard feasible prescribe hard (prefer exception hard)	100.0%	
no lens 3		
lens infeasible	100.0%	
soft lens 2		
lens feasible soft feasible prescribe soft (prefer exception soft)	100.0%	
no lens 4		
lens infeasible	100.0%	
hard lens 2		
lens feasible hard feasible prescribe hard (prefer exception hard)	100.0%	
no lens 5		
lens infeasible	100.0%	
soft lens 3		
lens feasible soft feasible prescribe soft (prefer exception soft)	100.0%	
no lens 6		
lens infeasible	100.0%	
hard lens 3		
lens feasible hard feasible prescribe hard (prefer exception hard)	100.0%	
no lens 7		
lens infeasible	100.0%	
soft lens 4		
lens feasible soft feasible prescribe soft (prefer exception soft)	100.0%	
no lens 8		
lens infeasible	100.0%	
no lens 9*		
lens feasible hard feasible exception hard [1] prescribe hard (prefer exception hard)	100.0%	
no lens 10		
lens infeasible	100.0%	
no lens 11*		
lens feasible soft feasible exception soft prescribe soft (prefer exception soft)	100.0%	
no lens 12		
lens infeasible	100.0%	
hard lens 4		
lens feasible hard feasible prescribe hard (prefer exception hard)	100.0%	
no lens 13		
lens infeasible	100.0%	
soft lens 5		
lens feasible soft feasible prescribe soft (prefer exception soft)	100.0%	
no lens 14		
lens infeasible	100.0%	
no lens 15*		
lens feasible hard feasible exception hard [2] prescribe hard (prefer exception hard)	100.0%	

Figure 123: Matches of contact lens cases against ideal elements from classes

Thus the match analysis based on the ideal elements provides a solution to the lens prescription problem but leaves the inference of preferring, or taking into account, exceptions to the user.

It might be regarded as a *decision support system* designed to aid people in making better decision rather than an *expert system* that carries out the full inferential process. For some types of problem, the long-standing technique of matching ideal elements in conceptual grids may be seen as as a viable alternative to the logical inference techniques of artificial intelligence, or more generally, these techniques may be seen as similar approaches to supporting and emulating aspects of human intelligence.

If one is interested in modelling human decision-making then both the grid and logical inference models might be over-structured. For example, it might be appropriate to factor the grid of ideal elements in a way that makes each stage of the decision process dependent on only one construct. The initial step might be based on the construct of astigmatism and suggest prescribing a hard lens if the client is astigmatic and a soft lens otherwise, subject no exceptions applying. The next step might be based on the constructs of myopia and presbyopia and the possible exceptions based on intersects of them. The final step on either branch might be based on the construct of tear production and the exception if it is reduced. This interpretation satisfies Cendrowska's requirements and is a simple and natural representation of a readily learnt anticipatory process that can be derived from the classes or the ideal elements based on them.

Such considerations suggest that that the personal construct framework represented in conceptual grids and associated classes or intersects might be useful in the study of human rationality. Cendrowska's contact lens problem is rather more complex than the simple logical tasks that have been used in empirical studies of human rationality but analyzing those tasks in terms of matching ideal elements might provide insights into the difficulties that people have in solving what are apparently very simple logical tasks.

5.7.2 Match analysis of Wason's card selection task

For example, consider Wason's original card selection task: "*The subjects were presented with the following sentence, 'if there is a vowel on one side of the card, then there is an even number on the other side,' together with four cards each of which had a letter on one side and a number on the other side. The task was to select all those cards, but only those cards, which would have to be turned over in order to discover whether the experimenter was lying in making the conditional sentence.*" (Wason, 1968, p.273). There are four possible card types and two possible visible sides, so there are eight possible cases that may be represented in a grid (Figure 124).

The first two constructs in the grid represent the factual situation, the next two the subject's knowledge of it, and the last whether the situation is consistent with the experimenter's statement or shows it to be a lie. A situation where both sides of the card are visible and provide evidence of a lie can be defined as a class *visible is vowel and visible is odd* (Figure 125).

If this card is included as an ideal element and the matches are computed based on the two *visible* constructs representing the subject's knowledge then, because only one side of the card is visible, the maximum match is 50% (Figure 126). However, the four situations that matched are the ones where a vowel or an odd number is visible which are the correct selections. What requires explanation with Wason's task is that the majority of subjects did not arrive at this solution and made incorrect selections.

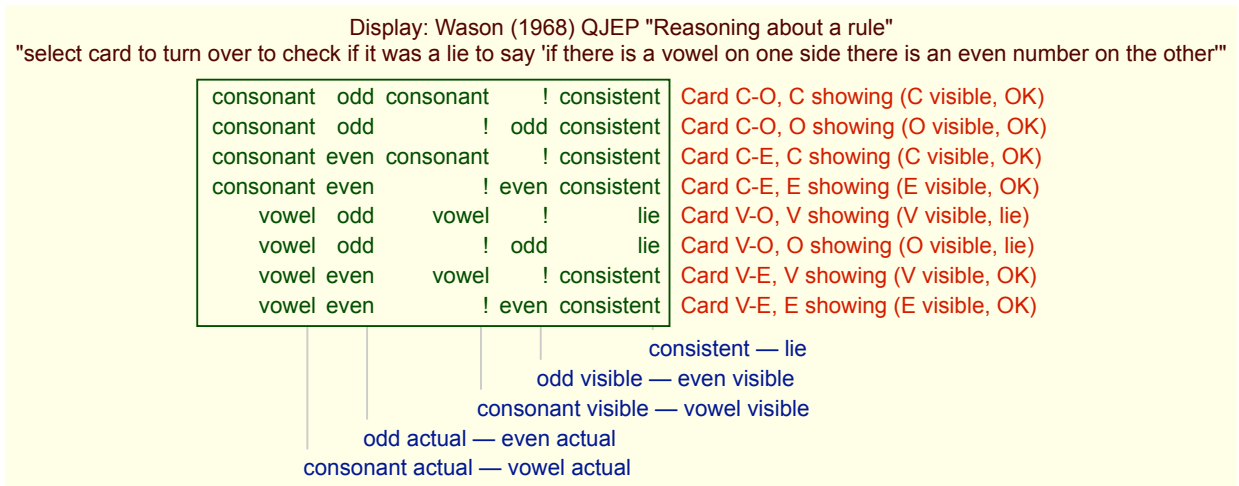


Figure 124: Grid representing the possible situations of Wason's card selection task

Classes			Interpretation <input checked="" type="radio"/> Case <input type="radio"/> Rule	
#	Name	Meaning		
1	evidence of lie	visible is vowel and visible is odd and lie		

Classes			Interpretation <input type="radio"/> Case <input checked="" type="radio"/> Rule	
#	Name	Meaning		
1	evidence of lie	visible is vowel and visible is odd entails lie		

Figure 125: Class specifying the state of card that provides evidence of a lie—as case or rule

Match Wason (1968) QJEP "Reasoning about a rule"

Matches between elements (at least 50%, based on selected constructs)

Card C-O, O showing (O visible, OK)	
evidence of lie	50.0%
Card V-O, V showing (V visible, lie)	
evidence of lie	50.0%
Card V-O, O showing (O visible, lie)	
evidence of lie	50.0%
Card V-E, V showing (V visible, OK)	
evidence of lie	50.0%

Figure 126: Matches of the class as an ideal element to the situations of Wason's card selection task

One explanation might be that the inference is difficult for people that the statement *if there is a vowel on one side of the card, then there is an even number on the other side* is a lie if a card has a vowel on one side and an odd number on the other. If the problem was stated in terms of a violation, that *a card with a vowel on one side and an odd number on the other is not allowed* then the two conditions

where a card should be checked are both contained in the problem statement and the selection task might be easier.

Such considerations have triggered wide-ranging research into variants of Wason's task and other reasoning problems, and there is now a massive literature of empirical data requiring explanation together with many theories suggesting further investigations (Evans et al., 1993; Hardman and Macchi, 2003; Stenning and Lambalgen, 2008; Adler and Rips, 2008). The conceptual modelling tools in Rep Plus provide a personal construct psychology framework within which to represent and compare such results and theories.

5.8 Analysis of selected elements and constructs

RepGrid makes it possible to display or analyze only part of a grid. At the bottom left of the *Elements* (§3.2) and *Constructs* (§3.3) panes are check boxes specifying that only selected items should be analyzed. In essence the grid to be analyzed is reduced to contain only the selected items, and hence all the analyses may be used on a partial grid without actually deleting elements or constructs.

In *Think Again* Shaw and McKnight (1981) present an example of the use of conceptual grids in decision support that illustrates the application of selected and weighted constructs in grid applications. As elements the user enters seven cars as potential choices together with an *ideal car* that will serve to elicit his preferences, and has rated the cars on relevant constructs. Figure 127 shows the grid developed to help in the choice of a car, and Figure 128 shows a Focus analysis where the cars clustered with *ideal car* suggest a basis for making an appropriate choice.

Display Jim
"choosing a car"

high fuel consumption	5	1	3	3	4	1	2	5	low fuel consumption
high running cost	3	2	4	3	5	4	1	5	low running cost
low engine reliability	5	5	4	3	3	1	2	5	high engine reliability
low brake reliability	3	3	5	5	1	3	2	5	high brake reliability
dull and boring	4	1	3	5	5	3	4	5	stylish
comfortable ride	2	3	5	1	4	4	1	1	bumpy
noisy	4	4	3	5	2	1	3	5	quiet
lots of color choice	1	2	2	3	5	1	4	1	not much color choice

	Ideal car
	Hyundai Excel
	Nissan Sentra
	Toyota Tercel
	Subaru Justy
	Volkswagen Golf
	Honda Civic
	Ford Festiva

Figure 127: Car choice decision-support grid

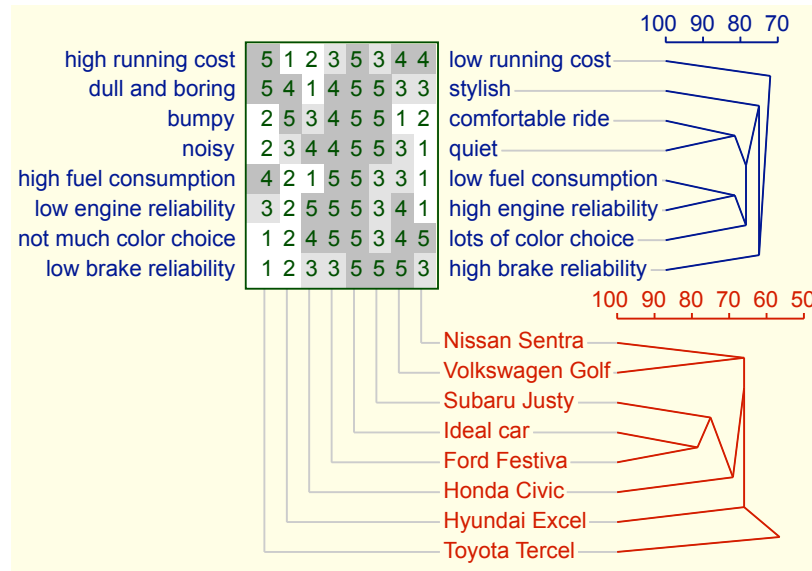


Figure 128: *Focus Cluster* analysis of car choice decision-support grid

After seeing this analysis he may be interested in clustering based only on what he sees as the most significant constructs, and can restrict the analysis by selecting these in the *Constructs* pane and clicking on *Analyze Select* as shown in Figure 129.

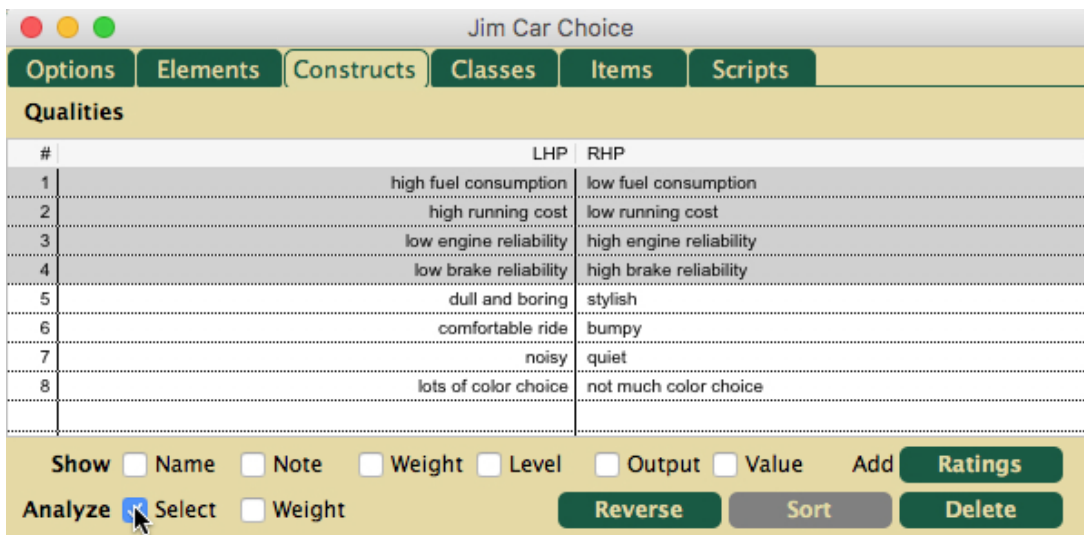


Figure 129: Car choice decision-support grid with analysis based on selected constructs

Clicking on the *Focus* button now results in an analysis based only on the selected constructs (Figure 130). The elements to be used can be selected on the *Elements* pane in the same way. Using *Analyze Select* for both elements and constructs provides the facility to display or analyze partial grids without editing the grid data.

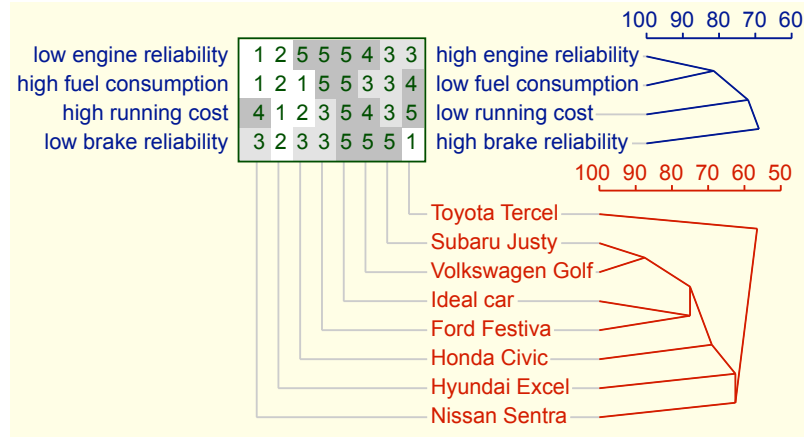


Figure 130: *Focus Cluster* analysis of car choice decision-support grid based on selected constructs

5.9 Analysis of weighted elements and constructs

Grid analysis can be refined in a more nuanced fashion by *weighting* elements and constructs. At the bottom left of the *Elements* and *Constructs* panes are check boxes specifying that the weight values should be used in analyses, notably *Display*, *Synopsis*, *Focus Cluster*, *PrinGrid Map*, and *Match*. The use of weights is indicated in the title of the output and the weight is shown in square parentheses by the elements and constructs in the plot.

Figure 131 shows the *Constructs* pane of the car choice grid where the user has weighted the constructs to indicate their relative importance to him in deciding which car best satisfies his needs. The *Show Weight* and *Analyze Weight* check boxes have been selected to show the weights and use them in analysis.

Qualities			
#	LHP	RHP	Wt
1	high fuel consumption	low fuel consumption	8
2	high running cost	low running cost	9
3	low engine reliability	high engine reliability	6
4	low brake reliability	high brake reliability	10
5	dull and boring	stylish	5
6	comfortable ride	bumpy	6
7	noisy	quiet	4
8	lots of color choice	not much color choice	2

Show ☐ Name ☐ Note ☒ Weight ☐ Level ☐ Output ☐ Value Add Ratings

Analyze ☐ Select ☒ Weight Reverse Sort Delete

Figure 131: Car choice decision-support grid with weighted constructs

Clicking on the *Display* button displays the grid with the fact that weights are in use noted in the title, and the weight values displayed on the right of each construct (Figure 132).

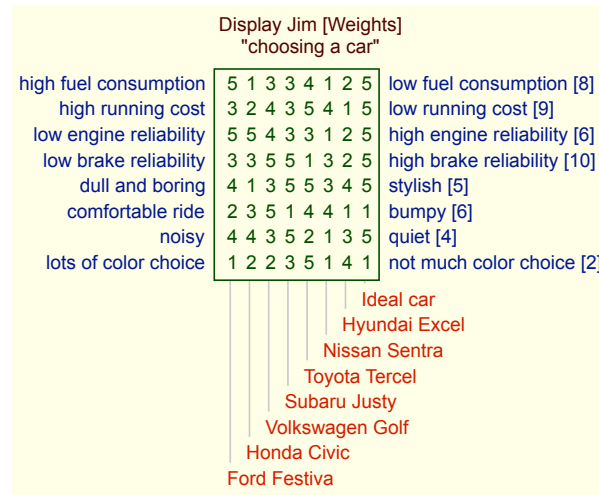


Figure 132: Display of car choice decision-support grid with construct weights

Only the relative values of weights are significant, and they may take any value from zero upwards. It is common to use 0 through 10 or 0 through 100. In the calculation of match scores for the *Focus* and *Matches* analyses, construct weights are used to weight differences in values on constructs in computing element matches, and element weights are used in computing construct matches. The effect on the analysis is the same as if the item with weight n had been entered in the grid n times.

Figure 133 shows the weighted *Focus* analysis for the car choice grid.

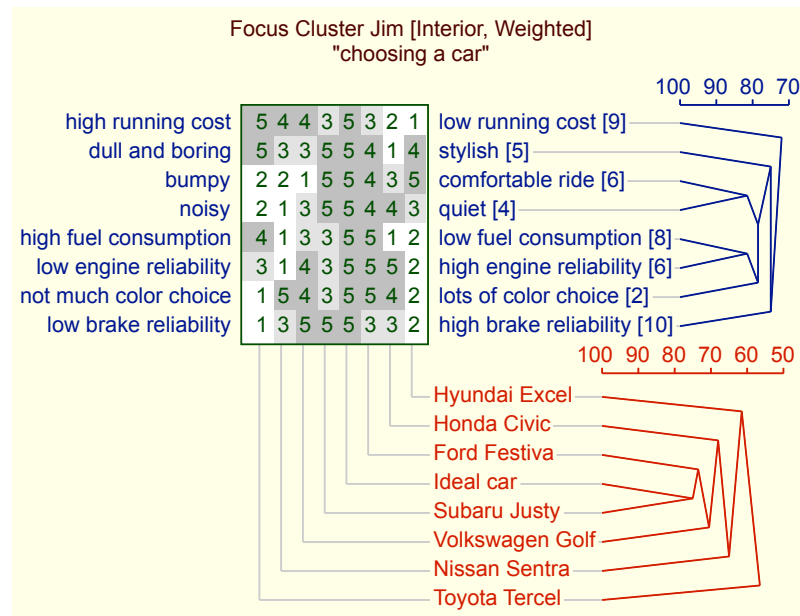


Figure 133: Focus Cluster analysis of car choice decision-support grid based on weighted constructs

In the unweighted analysis of Figure 128 *Ideal car* clusters most closely with *Ford Festiva*. However, in the weighted analysis of Figure 133, *Ideal car* clusters most closely with *Subaru Justy*.

If the user wishes to see the rank ordering of the matches of the *Ideal car* against all the others then *Ideal car* may be selected in the *Elements* pane, and *Matches* selected in the *Analysis* menu with the parameters selected as shown below to show the element matches of the selected element with all the others (Figure 134). This results in the list of matches shown in Figure 135.

Match: 1 cars (from 8) and 8 qualities

Match

☒ Elements
 ☐ Constructs

Notes

☐ Elements
 ☐ Constructs

Power

1

Cut off

0

☐ Numbers
 ☐ Index
 ☒ Separate

Match

☐ All
 ☒ Selected
 ☐ Unselected

With

☐ All
 ☐ Selected
 ☒ Unselected

☒ Text
 ☐ HTML

Cancel

Match

Figure 134: Match analysis to show how *Ideal car* compares the actual cars

Matches between cars (at least 0%, based on weighted qualities)

Ideal car	
Subaru Justy	75.0%
Ford Festiva	73.5%
Volkswagen Golf	62.5%
Toyota Tercel	51.0%
Honda Civic	41.5%
Hyundai Excel	36.5%
Nissan Sentra	35.5%

Figure 135: Weighted Match analysis of decision-support grid

Weights may also be used in other analyses. Figure 136 shows the *PrinGrid* output with the construct weights specified. *Ideal car* appears to be discriminated by the first component plotted on the horizontal axis, with *Ford Fiesta* being the nearest to it.

PrinGrid Map Jim [Covariances, Construct Weights]
"choosing a car"

low running cost x

low fuel consumption x

stylish x

Volkswagen Golf •

Ideal car •

Ford Festiva •

high engine reliability x

lots of color choice x

high brake reliability x

Subaru Justy •

quiet x

comfortable ride x

Toyota Tercel •

x bumpy [6]

x noisy [4]

x low brake reliability [10]

x not much color choice [2]

x low engine reliability [6]

Nissan Sentra •

1: 39.1%

Hyundai Excel •

x high fuel consumption [8]

x dull and boring [5]

Honda Civic •

x high running cost [9]

2: 24.2%

Figure 136: Weighted PrinGrid analysis of decision-support grid

98

The element and construct weights, if selected, are used multiplicatively in the calculation of the distance matrix for a principal components analysis. The effect of the weighting of the principal components analysis is equivalent to that of putting each element and each construct in the grid several times according to the weight allocated to it.

The results are similar to, but not the same as, those of the *Focus* and *Match* analyses, and this is because those analyses are based on a *boxcar* metric which sums absolute distances whereas the principal components analysis uses a *Euclidean* metric which sum the square of the distances and hence gives a greater weight to larger differences.

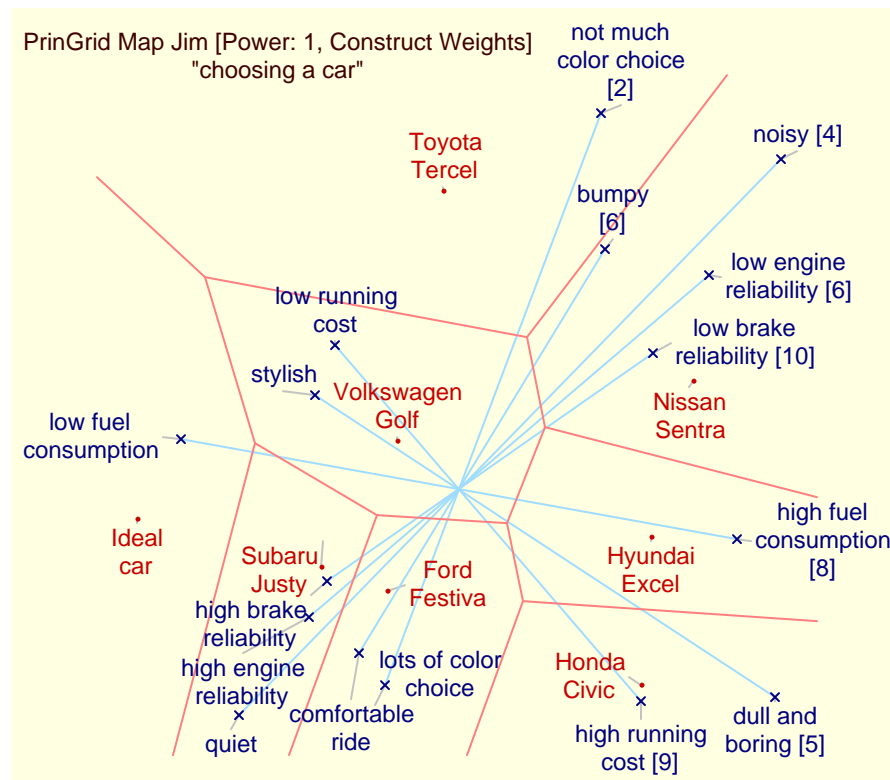


Figure 137: Weighted *PrinGrid* analysis of decision-support grid—Power 1.0, Voronoi diagram

Issues of alternative metrics and related research in the literature have been discussed in §5.4.3. The psychometrics literature suggests the *boxcar* metric corresponding to a Minkowski power of 1.0 is appropriate to human decision-making, which indicates that Figure 137 is more appropriate than Figure 136. However, the role of the grid analysis is to aid the client(s) to better understand the basis of the decision, not make it for them, and the grid analyses are primarily a basis for discussion, a *conversation with self* or a group discussion with those who will be effected by the decision. Study of the effect of different weightings and metrics can be used to ensure that the analyses reflect the intrinsic indeterminacy of multivariable decision-making. Some possibilities may be clearly unattractive, but others involve trade-offs between different desiderata that may be clarified by the analyses.

6 Style—managing the font and colour scheme of analyses

Clicking on the *Style* button brings up the dialog controls the colour schemes and grid identifier used in graphic output from all the analyses (Figure 138).

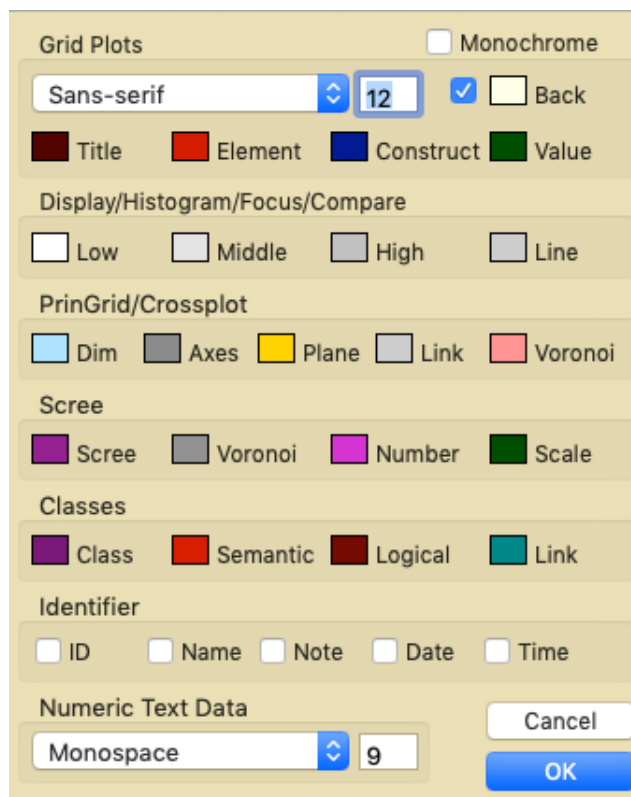


Figure 138: RepGrid *Style* dialog

All graphic output in Rep Plus is in the form of *nets*, graphic data structures that are detailed in the the RepNet Manual, can be edited and annotated in RepNet, and can be converted to a variety of graphic formats suitable for high-quality publication in documents, presentations and the web (§7).

Grid Plots The *Grid Plots* panel is common to all analyses. If the *Monochrome* checkbox is checked then all the analyses are in shades of grey determined only by the intensity of the colour selected, not its hue. The popup menu and associated text box specify the font and size of text output. The *Back* checkbox and associated swatch specifies whether the analyses have a coloured background and, if so, what colour is to be displayed. Colours may be specified for: the title; element names and trees; construct names and trees; and values of ratings and component percentages.

Display/Histogram/Focus/Compare The *Display/Histogram/Focus/Compare* panel is specific to those analyses. Colours may be specified for: the background tints to low, medium and high rating values; and lines joining element and construct names to the grid.

PrinGrid/Crossplot The *PrinGrid/Crossplot* panel is specific to those analyses. Colours may be specified for: the construct dimension lines; plot axes; X-Z plane and lines dropping to it in 3D output; link lines connecting element and construct names to their coordinates; and Voronoi diagrams.

Scree The *Scree* panel is specific to that analysis. Colours may be specified for: the scree plot; comparison plot; estimated number of underlying dimensions; and scales.

Classes The *Classes* panel is specific to the listing of the class meanings. Colours may be specified for: the class name; semantic logical terms; logical connectives; and values.

Identifier The *Identifier* panel is common to all analyses. It specifies what fields will be used to construct a phrase that identifies the grid. The *ID* is an item named *ID* which the user can enter as a customized identifier for the grid. If its use is specified and no *ID* item has been entered then the *UID* item that RepGrid automatically creates to provide the grid with a unique identifier is used. The *Name* and *Note* are those entered in the fields of the *Options* pane. If both are specified then the *Note* is placed in parentheses after the *Name*. The *Date* and *Time* items are those item that RepGrid automatically creates when a grid is created. If no identifier fields are specified then the default of *Name* and *Note* is used.

Numeric Text Data The *Numeric Text Data* specifies the font to be used for data such as arrays. The font should be a monospaced to ensure the proper alignment of the data.

6.0.1 Colour selection

The colour swatches allow colours to be specified using either standard named colours or the colour selection widgets native to the platform on which Rep Plus is running. Clicking in the left half of a swatch brings up a menu of the 140 standard colours specified in the World Wide Web CSS documentation from which a colour may be selected (Figure 139).

Rep Plus tracks the last colour selected and this is presented as the first option at the top of the menu so that it is easy to replicate the previous colour selected. Clicking in the right half of a swatch brings up a colour selection widget native to the platform on which Rep Plus is running allowing for custom colours to be specified. Examples are provided in the RepNet Manual which also details how the grid colours are translated into colour styles in RepNet.

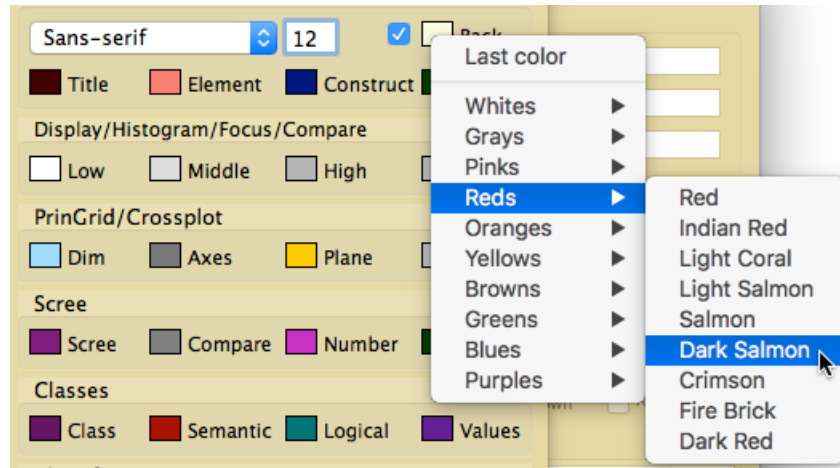


Figure 139: Colour selection when left half of swatch is clicked

7 Editing and exporting RepGrid output

The graphic plots and textual output from the RepGrid analyses may be used in reports, papers, theses or web pages, and Rep Plus supports their conversion to forms appropriate for publication. The plots produced by RepGrid analyses are vector graphics nets in *RepNet* format. The text outputs are fully styled documents in *RepDoc* format. Both can be saved, exported, dragged and copied/pasted in a variety of formats to document processors to produce a publication-quality documents.

RepNet organizes graphics as a set of nodes and links between them. Nodes can be selected by clicking on them and selected nodes are indicated with a light blue surround. Mousing down or CTL-clicking or right clicking in the graphic outside the nodes brings up a popup menu allowing the selected items to be exported as SVG, PNG or JPEG files (Figure 140). All three are standard graphic formats for the web. PNG and JPEG are generally accepted by word processors. SVG is a graphic interchange format accepted by graphics editors such as such as Illustrator, EazyDraw and Inkscape.

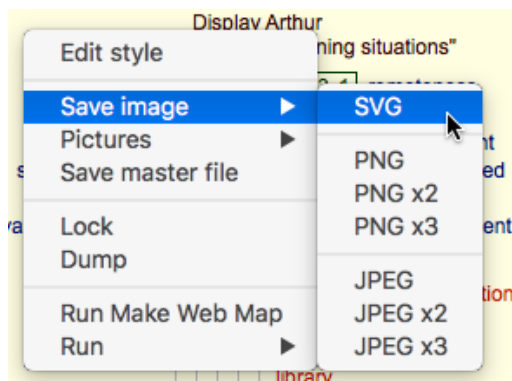


Figure 140: RepGrid graphic display of the data in a grid

RepNet also provides extensive graphic editing facilities such that the RepGrid analyses can be annotated for publication. More details are available in the RepNet manual.

Copy and paste of vector graphics is problematic with some word processors, particular for images containing Unicode text. When you select *Copy* from the *Edit* menu a bitmap image at screen resolution is made available in the clipboard for applications that cannot decode a RepNet file. You can increase the image resolution by using the *Scale* setting in each analysis dialog, and the font size setting in the *Style* dialog. Increase each by the same factor of 2 or 3 and generate a plot that is two or three times as large as usual. Copy this as a bitmap, paste it into your word processor, and rescale it to be 2 or 3 times smaller. The image quality when printed will then be substantially higher.

Exporting the graphic to a file, SVG or a 2x or 3x PNG or JPEG, provides an alternative process for exporting high-quality images to publications and presentations.

Text in RepDoc can be dragged or copied/pasted to other RepDoc documents or other document processes whilst in RTF retaining all styling. Graphic output from RepGrid can also dragged or copied/pasted to RepDoc documents to provide composite documents that can be saved or exported as RTF.

8 RepGrid/WebGrid integration

WebGrid (Gaines, 1995) was developed in 1994 as a way of making RepGrid functionality available on the World Wide Web and public WebGrid servers have long been used as an alternative to the stand-alone RepGrid program (Gaines and Shaw, 1996, 1997, 1998; Shaw and Gaines, 1996b, 1998). Whilst WebGrid proves the same functionality as RepGrid, the user interfaces for grid entry, elicitation and analysis are different, and Rep Plus is designed to make it simple to move grid data back and forth between the two programs so that users may move freely between them.

Rep Plus includes WebGrid and can act as a web server over the Internet, a local intranet, or on the same machine that is running the Rep Plus suite of programs. If RepGrid and WebGrid are running on the same machine the user interfaces of each provide capabilities to move grid data back and forth between them so that the two program can be used in an integrated fashion for different aspects of a task as the user prefers.

8.1 Transferring grid data from RepGrid to WebGrid

At the bottom left of the RepGrid window is a button labelled *WebGrid* (Figure 141). If clicked it starts up the WebGrid server for local use (if it is not already running) and copies the data from the grid being edited in RepGrid to WebGrid.

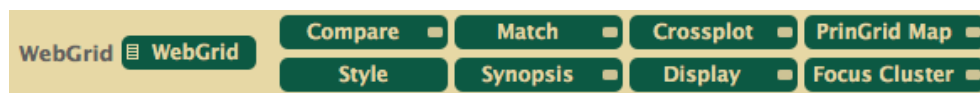


Figure 141: RepGrid to WebGrid transfer button

In addition, clicking on the menu symbol on the left of the WebGrid button brings up a menu that allows one to select what WebGrid page will open with the grid data (Figure 142), for example one may transfer to *Display* and see the grid data displayed. The default choice *WebGrid* takes one to the WebGrid main page.

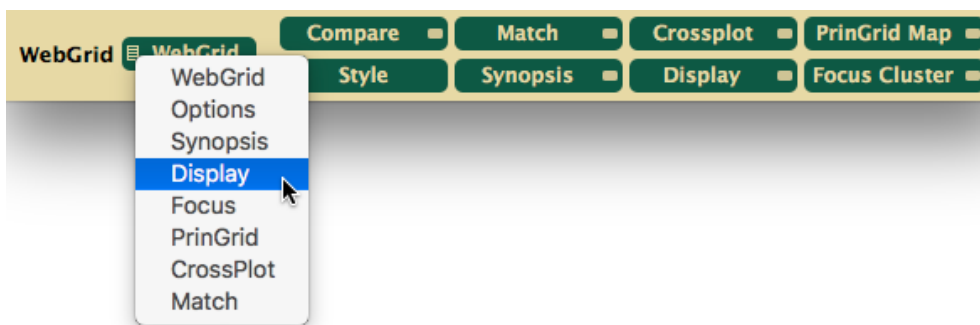




Figure 142: RepGrid to WebGrid transfer menu

Figure 143 shows the house choice grid opening in WebGrid after *WebGrid* has been clicked in its RepGrid window (Figure 20).



Rumpole's Realty



You are considering 9 homes and 9 qualities in the context of **choosing a new home**

You can choose from the options listed below, have the system choose, or request other choices

Choose for me ?

Other choices

The following homes are very similar,
1, Abraham Point, NW and **Ideal home**

Do you want to enter another quality to distinguish them?

Add quality ?

The following qualities are very similar,
extensive modernization—inadequate modernization
old year—recent year

Do you want to enter another home to distinguish them?

Add home ?

Can you think of a quality that distinguishes between the three homes,
92, Lexington Avenue NW, Ideal home and **325, Oaklands Drive NW**,
such that two are alike and differ from the third?

Add quality ?

Can you think of a quality that distinguishes between the two homes,
5778, Melina Drive NW and **23,080 120th Road, NW**?

Add quality ?

▼ You may add, delete, edit, or sort homes ?

325, Oaklands Drive NW
4227, Ranch Wheel Road NW
5778, Melina Drive NW
127, Realta Court NW
92, Lexington Avenue NW
436, Ryman Estate Drive NW
1, Abraham Point, NW
23,080 120th Road, NW
Ideal home

Add

Delete

Edit

Edit note

Sort

Select none

Click on homes to select those to be used (☐ use them in pairs or triads)

▼ You may add, delete, edit or sort qualities ?

mall near—long way from stores
mountain views—town views
friendly—oppressive
good study—poor study
completely remodeled—older style
needs redecorating—good decorations
extensive modernization—inadequate modernization
old year—recent year
low price (000s)—very high price (000s)

Add

Delete

Edit

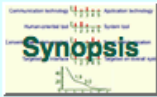





Edit note

Sort

Select none

Click on qualities to select those to be used

You can view and interpret your grid content in different ways ?

You may edit your options, save, exchange or cache your grid, send us a comment, finish, or adjust help ?

Options **Save** **Support** **Finish** **Rep Plus** **Help ? off**

Figure 143: RepGrid grid data opened in WebGrid

The house choice grid was developed in WebGrid and uses its customization capabilities to restyle its pages. The CSS and HTML code to do so is automatically included as an item of data in the grid file, and shows up in the *Items* pane (§3.5). The item has no effect on the operation of Rep-Grid but is recognized as styling data in WebGrid.

The notes attached to the elements in the grid are in HTML with links to images and these are displayed when the elements themselves occur in appropriate contexts in WebGrid. Figure 144 shows the WebGrid page when the user clicks on *Add quality* in the fourth row to elicit a new construct from the triad of elements suggested. The element annotation contains links to relevant images that are displayed to help the use recollect the nature of the element.



Rumpole's Realty



Elicit a new quality from a triad of homes ?

Think of the following three topics in the context of **choosing a new home**

In what way are **two of them alike** and **different from the third?**

Select the **one which is different**




Mary--bit oppressive. John--study spacious.
☐ 436, Ryman Estate Drive NW




Grounds damp--check basement. Wood finish matches our furniture.
☐ 4227, Ranch Wheel Road NW




Mary--very isolated. John--good hunting in own garden!
☐ 1, Abraham Point, NW

Enter a phrase characterizing the way in which the **selected home is different**

Enter a phrase characterizing the way in which the **other two homes are alike**

Figure 144: Hypertext annotation of elements in WebGrid elicitation

Figure 143 shows how the annotation appears in the table of top matches for *Ideal home*..








 Rumpole's Realty 	
Matches Between Homes	
Home	M%
Ideal home	
What would we want of an ideal home?	
<p>1, Abraham Point, NW</p>  <p>Mary--very isolated. John--good hunting in own garden!</p>	80.8
<p>92, Lexington Avenue NW</p>  <p>Good fencing for dogs. John really liked this one.</p>	74.9
<p>325, Oaklands Drive NW</p>  <p>Drive needs resurfacing. Mary--kitchen has a lovely view.Good master bedroom?</p>	68.9
<p>23,080 120th Road, NW</p>  <p>Near the Alberts, but long way from anyone else. Mary--would needa gardener. John--studio would make magnificent study.</p>	63.2
<p>127, Realta Court NW</p>  <p>John--plumbing is shoddy. Mary--dining room magnificent.</p>	52.3

Figure 145: Hypertext annotation of elements in WebGrid list of matches

The element annotation for the top element is:

```
<p align="center">
<br>
Mary--very isolated. John--good hunting in own garden!<br>
```

The hypertext links are to images on the local server but could be to images or sounds anywhere on the web. Thus, the multimedia capabilities of web browsers enhance elicitation processes in WebGrid beyond those available in RepGrid. See the WebGrid manual for further details.

If WebGrid is not running on the same machine then grid data from a RepGrid file may be uploaded using the *Upload* link on WebGrid's startup page.

8.2 Transferring grid data from WebGrid to RepGrid

When the WebGrid server recognizes it is running on the same machine as its client it modifies the user interface in the web browser to enable grid data being processed in the browser to be copied to Rep Plus to create a new grid window in RepGrid. A *Rep Plus* button is created at the bottom right of the WebGrid main page (Figure 143). Clicking on this copies the grid file to Rep Plus.

In addition to the *Rep Plus* button, the grid icon normally at the top right of every WebGrid page becomes a button with *Rep +* superimposed on the icon, and clicking on this also copies the grid data to Rep Plus and opens it in RepGrid. Thus, transferring data between WebGrid and RepGrid is a simple process that can be performed at any time.



Note that the grid that opens on either side of the transfer is a *copy* of that in the originating application. When transferring back to RepGrid the copy may be saved with the same name as the original grid or as a new version. If the old version is still open the user needs to close it without saving or otherwise manage the existence of multiple versions of the grid.

If WebGrid is running on a different machine then the grid data may be downloaded as a file either by clicking on the *Save* button at the main page and following the instructions there, or by simply saving the page as HTML. Rep Plus will open a WebGrid HTML file in RepGrid and extract the grid data from it.

9 Data Formats

Rep Plus supports a number of data formats for information transfer, including the basic file format of our earlier grid programs which is useful as a transfer format for grid data that needs to be digitally encoded for use in RepGrid. There is also an alternative spreadsheet format which can also be useful to encode data for fir RepGrid.

9.1 Basic grid format

The basic grid format used in Shaw's (1980) original repertory grid elicitation and analysis programs has been adopted by others and also provides a simple format for transfer of any grid data. Grids in this format have the following structure:-

Line 1	4 numbers separated by commas:- Number of elements—E Number of constructs—C Lower limit of rating scale Upper limit of rating scale
<hr/> <i>The next 3 lines are optional</i>	
Line 2	Purpose
Line 3	Name
Line 4	Note
<hr/> <i>The ratings may be entered in one of two formats, packed or separated</i>	
Line 5	Rating of Construct 1 on each element with no space <i>e.g. 12345—ratings may include meta-values</i>
Line 5	Rating of Construct 1 on each element separated by commas <i>e.g. 1,2,3,4,5—ratings may include meta-values</i>
<hr/> <i>The remaining items may be truncated and missing information will be filled appropriately</i>	
Line 5+C	Left Pole of Construct 1
Line 6+C	Right Pole of Construct 1
Lines 7+C to 4+3*C	Remaining Constructs
Line 5+3*C	Element 1
Lines 5+3*C to 4+3*C+E	Remaining Elements
Line 5+3*C+E	Singular for Construct
Line 6+3*C+E	Plural for Construct
Line 7+3*E+C	Singular for Element
Line 8+3*E+C	Plural for Element

The grid data on “exploring the nature of learning situations” used in previous sections is shown below in this format.

```

9,7,1,5
exploring the nature of learning situations
Arthur
after class discussion
432151231
443251521

```

```

455113215
544351221
524451113
442351551
542251551
involvement
remoteness
flexible
rigid
equipment
no equipment
self-organised
staff-organised
small group
large group
variable content
specific content
like
dislike
lecture
tutorial
seminar
practical
film
library
programmed text
video tape
informal interaction
situation
situations
quality
qualities

```

A file in this format may be opened as a grid in RepGrid, or the file or the text may be dragged to the *Rep 5 Manager* window to create a new grid as (Figure 146).

As documented above, some of the lines in the grid data may be omitted and will be filled appropriately. This enables existing grid ratings to be transferred rapidly to RepGrid with other fields being added in the RepGrid editor as required. Below is a minimal version of the grid above.

```

9,7,1,5
432151231
443251521
455113215
544351221
524451113
442351551
542251551

```

If a file with this data is opened, or the data is dragged to a grid window and dropped on it, and the *Display* button is clicked then the output will be as shown in Figure 147. The elements and constructs have been given identifiers sufficient to support meaningful display and analysis, and the name, purpose, and full construct and element names may be entered in the appropriate RepGrid fields as required.



Figure 146: Grid data being dragged and dropped on the Rep Plus Manager window

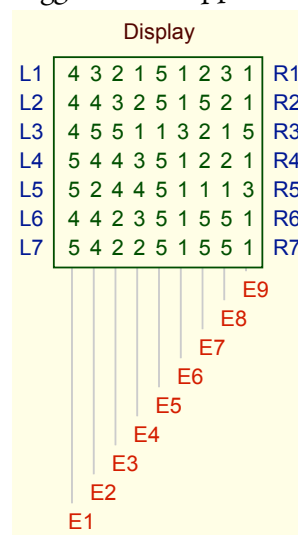


Figure 147: Minimal grid data displayed after being dragged and dropped

9.2 Spreadsheet grid entry format

A simple format for single or multiple grid data entry using spreadsheet programs was defined for RepGrid 2 and is supported by Rep Plus. It is based on the *tab-delimited* file format available in most spreadsheet programs.

The format is very simple, consisting of text separated by tab characters, and can also be generated in text editors and word processors, so that it provides another easy way to get data into RepGrid. Multiple grids may be stored in the same spreadsheet, and the format is designed to avoid the need to enter redundant data. For example, if several grids have the same scale this need be entered once only, similarly if several grids use the same elements, or the same constructs, these need be specified once only. This is particularly useful in entering grid data where multiple grids have common elements and/or common constructs.

Data is read in row by row commencing with the first row, and the various value settings persist unless overwritten by later settings. The first cell in a row normally contains a keyword such as *Name*, *Note*, *Purpose*, *Range* or *Grid*. The rest of the cells in that row contain data relating to the keyword. The keywords can be any order except that *Grid* must be the final one, and signifies that the grid data follows on several rows. The grid data must be terminated by a blank line (or by being the final data in the spreadsheet). Outside grid data, blank lines, and those not commencing with a keyword, are skipped and may be used to improve appearance and for comments.

Figure 148 shows some grid data in a spreadsheet:

	A	B	C	D	E
1	Name	pets			
2	Note	test data			
3	Purpose	illustrate data transfer			
4	Range	1	9		
5	Grid	dog	cat	rabbit	
6	remote	7	9	3	friendly
7	clean	3	1	7	messy
8					
9	Range	-3	3		
10	Grid				
11		2	-2	2	
12	quiet	2	-2	-2	noisy
13					

Figure 148: Two grids entered in a spreadsheet

Row 1 is optional and specifies the name for the first grid to be *pets*. If this row is omitted the name is set to be *Grid-1*.

Row 2 is optional and specifies the note for the first grid to be *test data*.

Row 3 is optional and specifies the purpose for the first grid to be *illustrate data transfer*.

Row 4 is optional and sets the scale range to be 1 to 9. If this row is omitted the range is set by default to be 1 to 5.

Row 5 is required and specifies that a grid dataset follows, with 3 elements, *dog*, *cat* and *rabbit*. The count of the number of elements in the grid following is obtained from the number of entries on this line.

Rows 6 and 7 specify two constructs and the ratings for each of the elements on these constructs. Row 8 is blank and terminates the building of the first grid. Further data will belong to further grids.

Row 9 changes the scale to be from -3 to +3. If this line was not present the scale would remain as 1 to 9.

Because the name, note and purpose keywords have been omitted, the name will be generated as *Grid-2* and the note and purpose fields will be the same as those already specified.

Row 10 specifies that a grid dataset follows. Since no new elements are specified, the three elements already specified continue to apply.

Row 11 specifies the first construct of a new grid. Since no pole names are specified, the corresponding construct in the previous grid applies, that is, *remote—friendly*.

Row 12 specifies another construct.

Row 13 is blank and end of the spreadsheet. It terminates building the second grid.

If this spreadsheet data is dragged to the *Rep 5 Manager* window, or is saved using the *Text (tab delimited)* option and opened in Rep Plus, two RepGrid windows are created, one for each of the grids in the spreadsheet file. These are shown below together with a Display of each grid.

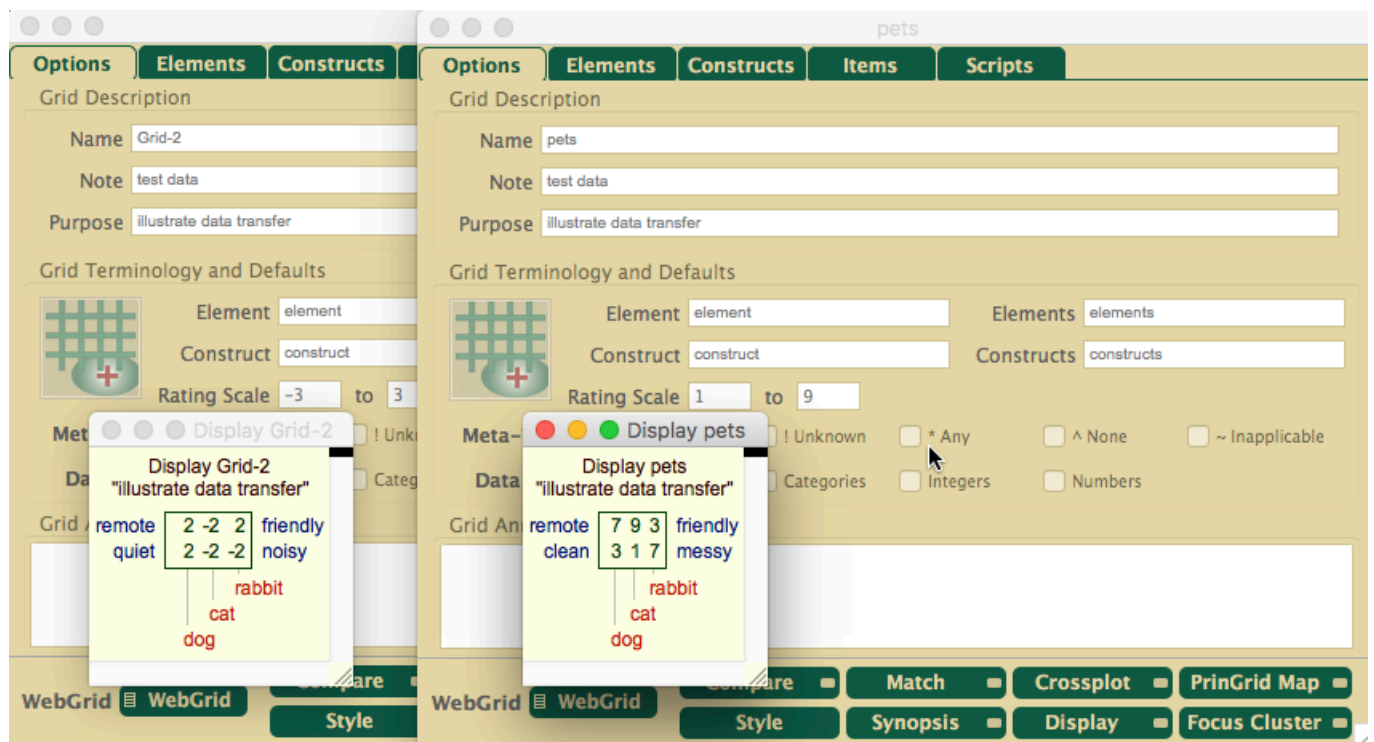


Figure 149: RepGrid windows created from spreadsheet tab-delimited data

10 Appendix: Elicit Scripts

The sample scripts for elicitation are in English and emulate Shaw's (1980) PEGASUS program. Facilitators may wish to edit them to make them more appropriate to particular communities and to translate them into other languages. The scripts are listed here to illustrate what is involved.

For many purposes there will be no need to change the programming and modifying the quoted text which constitutes the dialog will be sufficient. The programming language is fully documented in the *RepScript Manual* and editable in the Rep Plus *Script Editor* which automatically lays it out and colours it to make the syntactic form more apparent.

10.1 Script: Elicit Grid.repscript

The **Elicit Grid** script sets up the text styles, shows the welcome message, and passes control to the *Main* script at the *Initial* entry point.

```
// PEGASUS-style (Shaw 1980) conversational elicitation with feedback from matches

// set up standard text styles
StyleAdd("Text",1,0,12,RGB(0,100,0),"") // 12 pt green
StyleAdd("Center",2,0,12,RGB(0,100,0),"") // 12 pt green centred
StyleAdd("CenterBold",2,1,12,RGB(0,100,0),"") // 12 pt green centred bold
StyleAdd("HeadBig",2,1,18,RGB(255,0,0),"") // 18 pt red centred bold
StyleAdd("Head",2,1,14,RGB(255,0,0),"") // 14 pt red centred bold
StyleAdd("CenterBoldBig",2,1,14,RGB(0,100,0),"") // 14 pt green centred bold

// initialize appearance
SetBackColor(RGB(240,240,255))
StyleSet("Text")

// Commence flow of interactive dialog
Select Case vGet
Case ""
    UndoSave("Elicit Grid")
    SetMessage("")
    hSet("true","Initial") // set a flag to tell other code we are in initial phase
    if gGet("Context")="" or gGet("Name")="" or gGetI("ne")<6 then
        TextClear
        Output("PEGASUS Plus"+EOL+EOL,"HeadBig")
        Output("Program Elicits Grids and Sorts Using Similarities"+EOL+EOL,"Head")
        Output("This is a program to elicit a Repertory Grid. ")
        Output("A repertory grid is a technique devised by George Kelly to help you explore the
            dimensions of your thinking. ")
        Output("Please read carefully everything that is shown, and make sure you understand what you
            have to do. ")
        Output(EOL+EOL)
    end if
    ScriptFlow("/Elicit/Main/Initial") // normal entry
Case "*" // re-entry to script window when grid edited elsewhere
    TextClear
    Output("Continue Repertory Grid Elicitation"+EOL+EOL,"Head")
    ScriptFlow("/Elicit/Main/Initial")
End Select
```

10.2 Script: Elicit/Main.repscript

The **Main** script controls the flow of elicitation, passing control to other scripts which take action and then return to the Main script.

```
dim ne,nc,status As integer, te,tes,ten,tc,tcs,tcn As string
ne=gGetI("ne")
nc=gGetI("nc")
status=gGetI("Status")
te=gGet("E")
tes=gGet("Es")
if ne=1 then ten=te else ten=tes
tc=gGet("C")
tcs=gGet("Cs")
if nc=1 then tcn=tc else tcn=tcs

sub ClearMatches()
  dim i,n As integer
  n=gGetI("MatchC","80","0","M","MV") //get rid of construct matches in existing grid
  for i=0 to n-1
    hSet(vGetI(i,"MV"),vGet(i,"M"),"MatchC")
  next
  n=gGetI("MatchE","80","0","M","MV") //get rid of element matches in existing grid
  for i=0 to n-1
    hSet(vGetI(i,"MV"),vGet(i,"M"),"MatchE")
  next
end sub

Select case ScriptState

Case "Initial" // initial entry point
  // main ScriptFlow of control
  hEmpty("MatchE")
  hEmpty("MatchC")
  ClearMatches
  ScriptFlow("CheckName","Context")

Case "Context"
  ScriptFlow("CheckContext","InitialElements")

Case "InitialElements"
  SetMessage(gGet("Name")+" is considering ""+gGet("Context")+""")
  ScriptFlow("GetInitialElements","CheckExchange")

Case "CheckExchange"
  if status=3 or status=5 then // exchange or constructs
    ScriptFlow("Constructs/RateAnyOpen","Options") // don't test for matches
  else
    ScriptFlow("Triads","CheckMatches") // new grid or elements
  end if

Case "CheckMatches"
  ScriptFlow("Match/DoMatch","Options")

Case "Options"
  hRemove("Initial") // unset flag
  TextClear
  Output("Select an option"+EOL+EOL,"Head")
  Output("Your grid has "+str(ne)+" "+ten+" and "+str(nc)+" "+tcn+", and you may chose what to do
    next. ")
  Output("Click on one of the options below to select it."+EOL+EOL)
  if nc<gGetI("LimitC") and ne>2 then Output("T\Elicit another "+tc+" from a triad"+EOL,"
    CenterBold")
  Output("E\List and edit your "+tes+EOL,"CenterBold")
```

```

Output("C\List and edit your "+tcs+EOL,"CenterBold")
Output("X\Finish now"+EOL,"CenterBold")
ScriptWait("OptionsClick",kClick)
Case "OptionsClick"
select case InCode
case "T"
    ScriptFlow("AddConstruct/DoTriadChosen","CheckMatches")
case "E"
    ScriptFlow("Elements/List","CheckMatches")
case "C"
    ScriptFlow("Constructs/List","CheckMatches")
case "X"
    ScriptFlow("Constructs/RateAnyOpen","Halt") // Check for unrated given constructs
case "$"
    ScriptFlow("Options")
else
    Alert("Not editable","The item in which you clicked cannot be edited")
    ScriptWait("OptionsClick",kClick)
end select

Case "Halt"
    TextClear
    Output(EOL+"User has finished"+EOL,"Head")
    Halt("")

Case "CheckName"
    if gGet("Name")="" then
        hSet("false","Initial")
        Output("    What is your name or identification: ")
        ScriptWait("InputName",kText)
    else
        ScriptFlow
    end if
Case "InputName"
    gSet("Name",Input)
    Output(EOL+EOL)
    ScriptFlow

Case "CheckContext"
    if gGet("Context")="" then
        hSet("false","Initial")
        Output("State your purpose"+EOL+EOL,"Head")
        Output("You must decide on a purpose for doing the grid and keep this in mind when you chose
the "+tes+"--")
        Output("the things you are going to think about during the program. These "+tes+" will then
be used to elicit "+tcs+".")
        Output(EOL+EOL)
        Output("If you make a typing error press the delete key as many times as you want to erase a
character, then carry on.")
        Output(EOL+EOL)
        Output("    What is your purpose? ")
        ScriptWait("InputContext",kText)
    else
        ScriptFlow
    end if
Case "InputContext"
    gSet("Context",Input)
    Output(EOL+EOL)
    ScriptFlow

Case "GetInitialElements"
    if ne<6 then
        if hGet("Initial")="false" then TextClear // if we collected name or context
        hSet("false","Initial")
        Output("Name six or more "+tes+EOL+EOL,"Head")

```

```

Output("You must choose a set of "+tes+" keeping in mind why you want to do this grid. ")
if tes="Elements" and status<>5 then // only output following text if type of elements has
    not been specified and not a "Constructs" copy
    Output("They could be people, events, pieces of music, pictures, books or what you want, ")
    Output("but whatever you chose they must be of the same type and each must be well known to
        you. ")
    Output("Try to choose specific things. ")
end if
Output("Now type each one after each colon. ")
Output("Do not forget to press return after each. ")
Output("When you have entered as many "+tes+" as you wish, just press return alone to
    continue")
Output(EOL+EOL)
ScriptFlow("GetElements")
else
    ScriptFlow
end if

Case "GetElements"
    Output(" What is "+te+" "+str(ne+1)+": ")
    ScriptWait("InputElement",kText)
Case "InputElement"
    Output(EOL)
    if Input="" then
        if ne>=6 or status=5 then // "Constructs" copy allowed to avoid triadic elicitation
            Output(EOL)
            ScriptFlow
        else
            Alert("Not enough "+tes,"You have entered only "+str(ne)+" "+tes+". You need at least 6 to
                elicit a grid.")
            ScriptFlow("GetElements")
        end if
    else
        hEmpty("X")
        hSet(Input,"Name","X")
        gSet("NewE","X")
        ScriptFlow("GetElements")
    end if

Case "Triads"
    hSet("", "E") // select random triads
    select case nc-sGetI(gGet("OpenC")) // don't include any given constructs that are totally
        unrated
    case 0
        ScriptFlow("AddConstruct/DoTriad","Triads")
    case 1
        TextClear
        Output("How to think about "+tcs+EOL+EOL,"Head")
        Output("Now you have one "+tc+" you know what to do. ")
        Output("You may think of "+tcs+" as lines along which each of your "+tes+" has a place in
            relation to all the other "+tes+. ")
        Output("Please do not use "+tcs+" which do not apply to all your "+tes+. ")
        Output("An example of this is redhead--blond, as it is impossible to rate a person with black
            hair on this "+tc+. ")
        Output("One pole must be in some sense what the other is not, and they must divide your "+tes
            +" into two approximately equal groups, ")
        Output("so please try to avoid "+tcs+" where nearly all the "+tes+" are at one end. ")
        Output("An example might be ""extremely tall--not extremely tall"""+EOL+EOL)
        ScriptFlow("AddConstruct/DoTriadNoClear","Triads")
    case 2
        ScriptFlow("AddConstruct/DoTriad","Triads")
    case 3
        ScriptFlow("AddConstruct/DoTriad","Triads")
    else
        ScriptFlow

```

```

        end select

Case "Compile"
    ScriptFlow

else
    Halt("Flow error--no label "+ScriptState+EOL)

end select

```

10.3 Script: Elicit/Elements.repscript

The **Elements** script lists, adds and rates elements, and allows members of a triad to be chosen.

```

dim ne,nc,status As integer, te,tes,ten,tc,tcs,tcn As string
ne=gGetI("ne")
nc=gGetI("nc")
status=gGetI("Status")
te=gGet("E")
tes=gGet("Es")
if ne=1 then ten=te else ten=tes
tc=gGet("C")
tcs=gGet("Cs")
if nc=1 then tcn=tc else tcn=tcs

dim i,j,jj,n As integer, lhp,rhp,v,e,e1,e2,s,t As string, b As Boolean

Select case ScriptState

Case "List"
    TextClear
    Output("List of "+tes+EOL+EOL,"Head")
    for j=0 to ne-1
        n=gGetI("OpenE",str(j))
        if n=0 then s="" else s=" (" +str(n)+" "+tcs+" not rated)"
        Output("E"+str(j)+"\ "+" "+gGet("E",str(j))+s+EOL)
    next
    Output(EOL)
    if ne<gGetI("LimitE") then Output("A\Click here to add another "+te+"."+EOL,"CenterBold")
    Output("$\Click in an item to edit it, or here if you have finished editing."+EOL,"CenterBold")
    ScriptWait("ListClick",kClick)
Case "ListClick"
    select case Left(icode,1)
    case "$"
        ScriptFlow
    case "A"
        ScriptFlow("AddElement","List")
    case "E"
        hSet(Right(icode,len(icode)-1),"E")
        ScriptFlow("EditElement/DoEditElement","List")
    else
        Alert("Not editable","The item in which you clicked cannot be edited")
        ScriptWait("ListClick",kClick)
    end select

Case "AddElement"
    TextClear
    Output("Add another "+te+EOL+EOL,"Head")
    Output(" What is "+te+" "+str(ne+1)+": ")
    ScriptWait("AddElementIn",kText)
Case "AddElementIn"
    if Input="" then

```

```

    Output(EOL)
    ScriptFlow
else
    hSet(ne,"E")
    hEmpty("X")
    hSet(Input,"Name","X")
    gSet("NewE","X")
    Output(EOL+EOL+"Give your "+te+", "+Input+", a rating on each "+tc)
    Output(" by entering a number or clicking to use a popup menu."+EOL+EOL)
    ScriptFlow("CRatingLoop")
end if

Case "CRating" // Entry after an element match
    TextClear
    Output("Rate your "+te+" on the "+tcs+EOL+EOL,"Head")
    Output("Give your "+te+", "+gGet("E",hGet("E"))+", a rating on each "+tc)
    Output(" by entering a number or clicking to use a popup menu."+EOL+EOL)
    ScriptFlow("CRatingLoop")
Case "CRatingLoop"
    j=hGetI("E")
    vCountSet(0,"Sort") // set up a vector of unrated constructs
    for i=0 to nc-1
        v=gGet("V",str(i),str(j))
        if v<>"?" then Output(" "+gGet("C",str(i))+": "+v+EOL) else vPush(i,"Sort")
    next
    ScriptFlow("CRatingLoopOpen")
Case "CRatingLoopOpen"
    if vOK("Sort") then
        ScriptFlow("CRatingLoopOut")
    else
        Output(EOL)
        ScriptFlow("EditElement/DoEditElement")
    end if
Case "CRatingLoopOut"
    i=vPopI("Sort")
    hSet(i,"C")
    Output(" "+gGet("C",str(i))+": ")
    ScriptWait("CRatingLoopIn",kCMenu)
Case "CRatingLoopIn"
    Output(EOL)
    i=hGetI("C")
    j=hGetI("E")
    s=NthField(Input," ",1)
    gSet("V",str(i),str(j),s)
    if gGet("V",str(i),str(j))<>s then
        Call gGet("C",str(i),"X")
        s=hGet("Range","X")
        Alert ("Value not appropriate","Enter a value in the range "+sGet(s,1)+" to "+sGet(s,2))
        ScriptFlow("CRatingLoopOut")
    else
        ScriptFlow("CRatingLoopOpen")
    end if

Case "ChooseTriad"
    hRemove("E")
    ScriptFlow("ReListTriad")
Case "ReListTriad"
    TextClear
    Output("Choose a triad of "+tes+EOL+EOL,"Head")
    Output("Press the return key to have the program select a triad. ")
    Output("Or, you may select up to three "+tes+" yourself. If you select less than three the
        others will be selected at random."+EOL+EOL)
    s="Triad: "
    e=hGet("E")
    jj=sGetI(e)

```

```

for j=1 to jj
  t=gGet("E",sGet(e,j))
  if j=1 then s=s+t else s=s+", "+t
next
Output(s+EOL+EOL)
e=hGet("E")
for j=0 to ne-1
  if sFind(e,str(j))=0 then
    Output("E"+str(j)+"\ "+" "+gGet("E",str(j))+EOL)
  end if
next
Output(EOL)
Output("$\Click in your chosen "+te+" to select it, or here if you have finished selecting."+
  EOL,"CenterBold")
ScriptWait("TriadClick",kClick)
Case "TriadClick"
  select case Left(incode,1)
  case "$"
    ScriptFlow
  case "E"
    e1=hGet("E")
    e2=Right(incode,len(incode)-1)
    j=sGetI(e1)
    if j=0 then hSet(e2,"E") else hSet(sMake(e1,e2),"E")
    if j>1 then
      ScriptFlow
    else
      ScriptFlow("ReListTriad")
    end if
  else
    Alert("Not editable","The item in which you clicked cannot be edited")
    ScriptWait("TriadClick",kClick)
  end select

Case "Compile"
  ScriptFlow

else
  Halt("Flow error--no label "+ScriptState+EOL)

end select

```

10.4 Script: Elicit/EditElement.repscript

The **EditElement** script supports editing an element name and ratings.

```

dim ne,nc,status As integer, te,tes,ten,tc,tcs,tcn As string
ne=gGetI("ne")
nc=gGetI("nc")
status=gGetI("Status")
te=gGet("E")
tes=gGet("Es")
if ne=1 then ten=te else ten=tes
tc=gGet("C")
tcs=gGet("Cs")
if nc=1 then tcn=tc else tcn=tcs

dim n,i,j As integer, s,v As string, ok As Boolean

Select case ScriptState

Case "DoEditElement"

```

```

TextClear
j=hGetI("E")
s=gGet("E",str(j))
Output("Edit "+te+" ""+s+"""+EOL+EOL,"Head")
Output("E\      Name of "+te+": "+s+EOL+EOL)
for i=0 to nc-1
    Output("C"+str(i)+"\      "+gGet("C",str(i))+": "+gGet("V",str(i),hGet("E"))+EOL)
next
Output(EOL)
if hGet("Initial")="" then
Output("D\Click here to delete the "+te+EOL,"CenterBold")
end if
Output("$\Click on the "+te+" name or the "+tc+" to edit it, or here when you have finished
    editing"+EOL,"CenterBold")
ScriptWait("EditClick",kClick)
Case "EditClick"
s=Incode
Select Case Left(s,1)
case "D"
    ScriptFlow("EDelete")
case "E"
    ScriptFlow("EOut")
case "C"
    hSet(Right(Incode,len(Incode)-1),"C")
    ScriptFlow("CRatingOut")
case "$"
    ScriptFlow // return as specified by caller
else
    Alert("Not editable","The item in which you clicked cannot be edited")
    ScriptWait("EditClick",kClick)
end select

Case "EDelete"
j=hGetI("E")
s=gGet("E",str(j))
ok=Confirm("OK to delete "+s+"?", "Deleting the "+te+", "+s+", is irreversible. Click on OK if
    you really want to delete it.")
if ok then
    gSet("RemoveE",str(j))
    ScriptFlow
else
    ScriptFlow("DoEditElement")
end if

Case "EOut"
TextClear
s=gGet("E",hGet("E"))
Output("Edit name of "+te+" ""+s+"""+EOL+EOL,"Head")
Output("      Name of "+te+": ")
ScriptWait("EIn",kText)
Output(s)
Case "EIn"
if Input="" then
    ScriptFlow("DoEditElement")
else
    Output(EOL)
    hSet(Input,"Name","X")
    gSet("E",hGet("E"),"X")
    ScriptFlow("DoEditElement")
end if

Case "CRatingOut"
TextClear
i=hGetI("C")
j=hGetI("E")

```

```

Output("Edit rating for "+te+" "+gGet("E",str(j))+""+EOL+EOL,"Head")
Output(" "+gGet("C",str(i))+": ")
ScriptWait("CRatingIn",kCMenu)
OutputSelect(gGet("V",str(i),str(j)))
Case "CRatingIn"
Output(EOL)
if Input="" then
ScriptFlow("DoEditElement")
else
i=hGetI("C")
j=hGetI("E")
s=NthField(Input," ",1)
gSet("V",str(i),str(j),s)
Call gGet("C",str(i),"X")
if gGet("V",str(i),str(j))<>s then
s=hGet("Range","X")
Alert ("Value not appropriate","Enter a value in the range "+sGet(s,1)+" to "+sGet(s,2))
ScriptFlow("CRatingOut")
else
ScriptFlow("DoEditElement")
end if
end if

Case "Compile"
ScriptFlow

else
Halt("Flow error--no label "+ScriptState+EOL)

end select

```

10.5 Script: Elicit/Constructs.repscript

The **Constructs** script lists and edits constructs.

```

dim ne,nc,status As integer, te,tes,ten,tc,tcs,tcn As string
ne=gGetI("ne")
nc=gGetI("nc")
status=gGetI("Status")
te=gGet("E")
tes=gGet("Es")
if ne=1 then ten=te else ten=tes
tc=gGet("C")
tcs=gGet("Cs")
if nc=1 then tcn=tc else tcn=tcs

dim i,k,n As integer, s,c As string

Select case ScriptState

Case "List"
TextClear
Output("List of "+tcs+EOL+EOL,"Head")
for i=0 to nc-1
n=gGetI("OpenC",str(i))
if n=0 then s="" else s=" ("&str(n)&"+ "+tes+" not rated)"
Output("C"&str(i)&"+\ "+gGet("C",str(i))+s+EOL)
next
Output(EOL)
if nc<gGetI("LimitC") then Output("A\Click here to add another "+tc+"."+EOL,"CenterBold")
Output("$\Click in an item to edit it, or here if you have finished editing."+EOL,"CenterBold")
ScriptWait("ListClick",kClick)

```

```

Case "ListClick"
  select case Left(incode,1)
  case "$"
    ScriptFlow
  case "A"
    hRemove("E") // not from a triad or pair so make sure no elements
    ScriptFlow("AddConstruct/AddConstruct","List")
  case "C"
    hSet(Right(incode,len(incode)-1),"C")
    ScriptFlow("EditConstruct/DoEditConstruct","List")
  else
    Alert("Not editable","The item in which you clicked cannot be edited")
    ScriptWait("ListClick",kClick)
  end select

Case "RateAnyOpen"
  if gGetI("Open")>0 then
    hSet("true","RateOpen") // stop option to delete
    ScriptFlow("RateOpen","RateAnyOpen")
  else
    hRemove("RateOpen")
    ScriptFlow
  end if

Case "RateOpen"
  for i=0 to nc-1
    k=gGetI("OpenC",str(i))
    if k>n then
      hSet(i,"C")
      n=k
    end if
  next
  if n>0 then
    ScriptFlow("AddConstruct/RateConstruct")
  else
    ScriptFlow
  end if

Case "Compile"
  ScriptFlow

else
  Halt("Flow error--no label "+ScriptState+EOL)

end select

```

10.6 Script: Elicit/AddConstruct.repscript

The **AddConstruct** script supports adding constructs, directly, from triads and matches.

```

dim ne,nc,status As integer, te,tes,ten,tc,tcs,tcn As string
ne=gGetI("ne")
nc=gGetI("nc")
status=gGetI("Status")
te=gGet("E")
tes=gGet("Es")
if ne=1 then ten=te else ten=tes
tc=gGet("C")
tcs=gGet("Cs")
if nc=1 then tcn=tc else tcn=tcs

function MakeTriad() As Boolean

```

```

// enter with 0 or more prescribed elements in E
// select other elements at random to make the number up to 3 and return in E
// return false if not enough elements to make a triad
dim e,s As string
if ne<3 then return false
do
    e=hGet("E")
    if sGetI(e)>=3 then return true
    s=str(GetRandom(0,ne-1))
    if sFind(e,s)=0 then
        if e="" then e=s else e=e+TAB+s
        hSet(e,"E")
    end if
loop
end function

// main program
dim i,j,n As integer, lhp,rhp,v,s,e,b As string

Select case ScriptState

Case "RateConstruct"
    TextClear
    i=hGetI("C")
    Output("Rate "+tes+" on ""+gGet("C")+""+EOL,"Head")
    ScriptFlow("ERating")

Case "AddConstruct"
    TextClear
    Output("Add another "+tC+EOL+EOL,"Head")
    ScriptFlow("LHP")

Case "DoTriadChosen"
    ScriptFlow("Elements/ChooseTriad","DoTriad")

Case "DoTriad"
    TextClear
    Output("Elicit "+tC+" from a triad"+EOL+EOL,"Head")
    ScriptFlow("Triad")

Case "DoTriadNoClear"
    ScriptFlow("Triad")

Case "Triad"
    if MakeTriad then
        e=hGet("E")
        Output("Can you choose two of this triad of "+tes+" which are in some way alike and different
            from the other one?" +EOL+EOL)
        for i=1 to 3
            s=sGet(e,i)
            Output(s+"\ "+gGet("E",s)+EOL,"CenterBoldBig")
        next
        Output(EOL)
        Output("$\Click in the "+te+" which is different, or here if you cannot do this." +EOL,"
            CenterBold")
        ScriptWait("TriadClicked",kClick)
    else
        Halt("Not enough elements for triad"+EOL)
    end if
Case "TriadClicked"
    If InCode="$" then
        Output(EOL)
        ScriptFlow
    else
        e=hGet("E")

```

```

    if sGet(e,2)=Incode then // make sure the one clicked is first
        hSet(sMake(sGet(e,2),sGet(e,1),sGet(e,3)), "E")
    elseif sGet(e,3)=Incode then
        hSet(sMake(sGet(e,3),sGet(e,1),sGet(e,2)), "E")
    end if
    TextClear
    Output("Name the poles of your "+tC+EOL+EOL,"Head")
    Output("Now I want you to think what you have in mind when you separate the pair from the
        other one. ")
    Output("Just type one or two words for each pole to remind you what you are thinking or
        feeling when you use this "+tC+ ".")
    Output(EOL+EOL)
    Output("X\":"+ "Or click here if you cannot do this"+EOL+EOL,"CenterBold")
    ScriptFlow("LHP")
end if

Case "LHP"
    // enter with 0 to 3 element numbers stored in "E"
    e=hGet("E")
    s=sGet(e,2)
    if s<>" " then
        s=gGet("E",s)
        e=sGet(e,3)
        if e<>" " then s=s+" "+gGet("E",e)
        s=" {"+s+"}"
    end if
    Output("    Left pole rated "+gGet("MinR")+s+": ")
    ScriptWait("LHPIn",kText)
Case "LHPIn"
    Output(EOL)
    if InCode="X" or Input="" then
        Output(EOL)
        ScriptFlow // return, no construct added
    else
        hSet(Input,"L") // hold LHP in "L"
        ScriptFlow("RHP")
    end if
Case "RHP"
    s=sGet(hGet("E"),1)
    if s<>" " then s=" {"+gGet("E",s)+"}"
    Output("    Right pole rated "+gGet("MaxR")+s+": ")
    ScriptWait("RHPIn",kText)
Case "RHPIn"
    Output(EOL)
    if InCode="X" or Input="" then
        Output(EOL)
        ScriptFlow // return, no construct added
    else
        i=nc
        hSet(i,"C") // hold construct number in "C"
        hEmpty("X")
        hSet("R","Type","X")
        hSet(hGet("L"),"LHP","X")
        hSet(Input,"RHP","X")
        gSet("NewC","X")
        e=hGet("E")
        b="true"
        for n=1 to 3
            s=sGet(e,n)
            if s<>" " then gSet("EndV",str(i),s,b)
            b="false"
        next
        ScriptFlow("ERating")
    end if
end if

```

```

Case "ERating"
  TextClear
  i=hGetI("C")
  Output("Rate the "+tes+" on your "+tC+" ""+gGet("C",str(i))+""+EOL+EOL,"Head")
  Output("According to how you feel about them, please assign to each of the following "+tes+" a
    rating from ")
  Call gGet("C",str(i),"X")
  s=hGet("Range","X")
  Output(sGet(s,1)+" ("+hGet("LHP","X")+") to "+sGet(s,2)+" ("+hGet("RHP","X")+")")
  Output(" by entering a number or clicking to use a popup menu.")
  Output(EOL+EOL)
  Call gGet("SortV",str(i),"Sort","false")
  ScriptFlow("ERatingLoop")
Case "ERatingLoop"
  if vOK("Sort") then
    j=vExtractI("Sort")
    hSet(j,"E")
    v=gGet("V",str(hGetI("C")),str(j))
    if v<>"?" then
      Output(" "+gGet("E",str(j))+""+v+EOL) // show already set values
      ScriptFlow("ERatingLoop")
    else
      ScriptFlow("ERatingLoopOut")
    end if
  end if
else
  Output(EOL)
  ScriptFlow("EditConstruct/DoEditConstruct") // offer option to edit
end if
Case "ERatingLoopOut"
  Output(" "+gGet("E",hGet("E"))+": ")
  ScriptWait("ERatingLoopIn",kCMenu)
Case "ERatingLoopIn"
  Output(EOL)
  i=hGetI("C")
  j=hGetI("E")
  s=NthField(Input," ",1)
  gSet("V",str(i),str(j),s)
  if gGet("V",str(i),str(j))<>s then
    s=hGet("Range","X")
    Alert ("Value not appropriate","Enter a value in the range "+sGet(s,1)+" to "+sGet(s,2))
    ScriptFlow("ERatingLoopOut")
  else
    ScriptFlow("ERatingLoop")
  end if

Case "Compile"
  ScriptFlow

else
  Halt("Flow error--no label "+ScriptState+EOL)

end select

```

10.7 Script: Elicit/Match.repscript

The **Match** script finds and displays element and construct matches, and gives the option to add constructs and elements to reduce the match.

```

dim ne,nc,status As integer, te,tes,ten,tc,tcs,tcn As string
ne=gGetI("ne")
nc=gGetI("nc")
status=gGetI("Status")

```

```

te=gGet("E")
tes=gGet("Es")
if ne=1 then ten=te else ten=tes
tc=gGet("C")
tcs=gGet("Cs")
if nc=1 then tcn=tc else tcn=tcs

dim i,j,n,i1,i2,j1,j2,v As integer, match As double, e,e1,e2,s As string, b As Boolean

Select case ScriptState

Case "DoMatch" // find any matches not already checked
    b=false
    if ne>3 and ne<gGetI("LimitE") then // test for C matches
        n=gGetI("MatchC","80","0","M","MV")
        //print str(n)+EOL
        //print vDump("M")+EOL
        //print vDump("MV")+EOL
        //print hDump("MatchC")+EOL
        for i=0 to n-1
            s=vGet(i,"M") // get match pair
            if not hCheck(s,"MatchC") then // new match
                i1=sGetI(s,1) // get first construct
                hSet(i1,"C1") // hold number in C1
                Call gGet("C",str(i1),"X1") // hold specification in X1
                i2=sGetI(s,2) // get second construct
                hSet(i2,"C2") // hold number in C2
                Call gGet("C",str(i2),"X2") // hold specification in X2
                v=vGetI(i,"MV") // get match value
                hSet(v,"M") // hold match value in M
                hSet(v,s,"MatchC") // record match pair in MatchC
                b=true // found a new construct match
                ScriptFlow("ShowCMatch","DoMatch")
            exit
        end if
    next
end if
if not b and nc>3 and nc<gGetI("LimitC") then // test for E matches
    n=gGetI("MatchE","80","0","M","MV")
    for i=0 to n-1
        s=vGet(i,"M") // get match pair
        if not hCheck(s,"MatchE") then // new match
            hSet(s,"E") // hold matching E numbers
            v=vGetI(i,"MV") // get match value
            hSet(v,"M") // hold match value in M
            hSet(v,s,"MatchE") // record match pair in MatchE
            b=true // found a new element match
            ScriptFlow("ShowEMatch","DoMatch")
        exit
    end if
    next
end if
if not b then ScriptFlow // no matches

Case "ShowCMatch"
    TextClear
    Output("Break "+tc+" match"+EOL+EOL,"Head")
    Output("The two "+tcs+" you called"+EOL)
    Output(hGet("Identifier","X1")+EOL+hGet("Identifier","X2")+EOL,"Center")
    Output("are matched at the "+hGet("M")+ "% level.")
    Output(" This means that most of the time you are saying "+hGet("LHP","X1")+ " you are also
        saying "+hGet("RHP","X2"))
    Output(" , and most of the time you are saying "+hGet("RHP","X1")+ " you are also saying "+hGet("
        LHP","X2")+ ". "+EOL+EOL)
    Output("Think of another "+te+" which is either:-"+EOL+EOL)

```

```

Output("1\      "+hGet("LHP","X1")+ " and "+hGet("RHP","X2")+EOL,"CenterBold")
Output("or"+EOL,"Center")
Output("2\      "+hGet("LHP","X2")+ " and "+hGet("RHP","X1")+EOL+EOL,"CenterBold")
Output("$\Click on the appropriate combination, or here if you cannot do this"+EOL,"CenterBold
")
ScriptWait("BreakCMatch",kClick)
Case "BreakCMatch"
  Select Case Incode
  case "1"
    TextClear
    Output("Add a "+te+" to reduce a match between "+tcs+EOL+EOL,"Head")
    Output("X\ "+ "Or click here if you cannot do this"+EOL+EOL,"CenterBold")
    Output("  What is the "+te+" that is """+hGet("LHP","X1")+"" and """+hGet("RHP","X2")+""":
    ")
    ScriptWait("InElement",kText)
  case "2"
    TextClear
    Output("Add a "+te+" to reduce a match between "+tcs+EOL+EOL,"Head")
    Output("X\ "+ "Or click here if you cannot do this"+EOL+EOL,"CenterBold")
    Output("  What is the "+te+" that is """+hGet("RHP","X1")+"" and """+hGet("LHP","X2")+""":
    ")
    ScriptWait("InElement",kText)
  case "$"
    ScriptFlow // return as specified by caller
  else
    Alert("Not understood","The item in which you clicked is not an option")
    ScriptWait("BreakCMatch",kClick)
  end select
Case "InElement"
  if InCode="X" or Input="" then
    Output(EOL)
    ScriptFlow
  else
    j=ne
    hSet(j,"E")
    hEmpty("X")
    hSet(Input,"Name","X")
    gSet("NewE","X")
    b=InCode="1"
    gSet("EndV",hGet("C1"),str(j),BooStr(not b))
    gSet("EndV",hGet("C2"),str(j),BooStr(b))
    ScriptFlow("Elements/CRating")
  end if

Case "ShowEMatch"
  s=hGet("E")
  e2=gGet("E",sGet(s,1))
  e1=gGet("E",sGet(s,2))
  TextClear
  Output("Break "+te+" match"+EOL+EOL,"Head")
  Output("The two "+tes+" "+e1+" and "+e2+" are matched at the "+hGet("M")+ "% level. ")
  Output("This means that so far you have not distinguished between them."+EOL+EOL)
  Output("Think of another "+tc+" which separates "+e1+" from "+e2+"."+EOL+EOL)
  Output("X\ "+ "Or click here if you cannot do this"+EOL+EOL,"CenterBold")
  ScriptFlow("AddConstruct/LHP")

Case "Compile"
  ScriptFlow

else
  Halt("Flow error--no label "+ScriptState+EOL)

end select

```

11 Bibliography

- Adler, J. E. and Rips, L. J. (2008). *Reasoning: Studies of Human Inference and Its Foundations*. Cambridge University Press, Cambridge.
- Algom, D. and Fitousi, D. (2016). Half a century of research on Garner interference and the separability–integrality distinction. *Psychological Bulletin*, 142(12):1352–1383.
- Attneave, F. (1950). Dimensions of similarity. *American Journal of Psychology*, 43(4):516–556.
- Balme, D. M. (1987). Aristotle’s use of division and differentiae. In Gotthelf, A. and Lennox, J. G., editors, *Philosophical issues in Aristotle’s biology*. Cambridge University Press, Cambridge.
- Bellman, L. (2012). *Auktoriserade fastighetsvärderares syn på värdering: tankemönster om kommersiella fastigheter*. Licentiate thesis.
- Boose, J. H. and Gaines, B. R. (1988). *Knowledge Acquisition Tools for Expert Systems*. Academic Press, London.
- Buchanan, B. G. and Shortliffe, E. H. (1984). *Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, MA.
- Caputi, P. (2011). *Personal Construct Methodology*. Wiley, Chichester, UK.
- Cendrowska, J. (1987). An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370.
- Corbridge, C., Rugg, G., Major, N. P., Shadbolt, N. R., and Burton, A. M. (1994). Laddering: technique and tool use in knowledge acquisition. *Knowledge Acquisition*, 6(3):315–341.
- Davies, S. (1991). *Definitions of Art*. Cornell University Press, Ithaca, NY.
- Davison, A. C. and Hinkley, D. V. (1997). *Bootstrap Methods and Their Application*. Cambridge University Press, Cambridge.
- Dey, S. and Lee, S.-W. (2017). Reassure: Requirements elicitation for adaptive socio-technical systems using repertory grid. *Information and Software Technology*, 87:160–179.
- Efron, B. and Tibshirani, R. (1993). *An introduction to the bootstrap*. Chapman & Hall, New York.
- Emmons, O. (1939). *Linearity In Factor Analysis*. PhD thesis.
- Evans, J. S. B. T., Newstead, S. E., and Byrne, R. M. J. (1993). *Human Reasoning: The Psychology of Deduction*. Lawrence Erlbaum, Hove.
- Feixas, G. and Cornejo, J. M. (1996). *Manual de la técnica de rejilla mediante el programa RECORD ver. 2.0*. Paidós, Barcelona.

- Fortune, S. (1987). A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2:153–174.
- Fransella, F., Bell, R. C., and Bannister, D. (2004). *A Manual for Repertory Grid Technique*. Wiley, Chichester, UK.
- Fromm, M. (2004). *The Repertory Grid Interview*. Waxman, Munster.
- Frontier, S. (1976). Étude de la décroissance des valeurs propres dans une analyse en composantes principales: Comparaison avec le modèle du bâton brisé. *Journal of Experimental Marine Biology and Ecology*, 25(1):67–75.
- Gaines, B. R. (1995). Porting interactive applications to the web. In *4th International World Wide Web Conference Tutorial Notes*, pages 199–217. O'Reilly, Sebastopol, CA.
- Gaines, B. R. (2015). Universal logic as a science of patterns. In Koslow, A. and Buchsbaum, A., editors, *The Road to Universal Logic: Festschrift for 50th Birthday of Jean-Yves Béziau*, pages 145–189. Birkhäuser, Basel.
- Gaines, B. R., Chen, L. L.-J., and Shaw, M. L. G. (1997). Modeling the human factors of scholarly communities supported through the Internet and World Wide Web. *Journal American Society Information Science*, 48(11):987–1003.
- Gaines, B. R. and Shaw, M. L. G. (1993a). Basing knowledge acquisition tools in personal construct psychology. *Knowledge Engineering Review*, 8(1):49–85.
- Gaines, B. R. and Shaw, M. L. G. (1993b). Eliciting knowledge and transferring it effectively to a knowledge-based systems. *IEEE Transactions on Knowledge and Data Engineering*, 5(1):4–14.
- Gaines, B. R. and Shaw, M. L. G. (1996). Webgrid: Knowledge modeling and inference through the World Wide Web. In Gaines, B. and Musen, M., editors, *Proceedings of Tenth Knowledge Acquisition Workshop*, pages 65–1–65–14.
- Gaines, B. R. and Shaw, M. L. G. (1997). Knowledge acquisition, modeling and inference through the World Wide Web. *International Journal of Human-Computer Studies*, 46(6):729–759.
- Gaines, B. R. and Shaw, M. L. G. (1998). Developing for web integration in Sisyphus-IV: WebGrid-II experience. In Gaines, B. and Musen, M., editors, *Proceedings of Eleventh Knowledge Acquisition Workshop*.
- Gaines, B. R. and Shaw, M. L. G. (2012). Computer aided constructivism. In Caputi, P., Viney, L. L., Walker, B. M., and Crittenden, N., editors, *Constructivist Methods*, pages 183–222. Wiley, New York.
- Gärdenfors, P. (2000). *Conceptual Spaces: The Geometry of Thought*. MIT Press, Cambridge, MA.
- Gaut, B. (2000). “Art” as a cluster concept. In Carroll, N., editor, *Theories of Art Today*, pages 25–44. University of Wisconsin Press, Madison, WI.

- Gill, M. L. (2010). Division and definition in plato's sophist and statesman. In Charles, D., editor, *Definition in Greek philosophy*, pages 172–199. Oxford University Press, Oxford.
- Gower, J., Lubbe, S., and Le Roux, N. (2011). *Understanding Biplots*. John Wiley, Chichester, UK.
- Gower, J. C. (1966). Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53:325–338.
- Gower, J. C. and Hand, D. J. (1995). *Biplots*. Chapman & Hall, London.
- Greenacre, M. J. (2007). *Correspondence Analysis in Practice*. Chapman & Hall, London.
- Hardman, D. and Macchi, L. (2003). *Thinking: Psychological Perspectives on Reasoning, Judgment, and Decision Making*. Wiley, Hoboken, NJ.
- Hayes-Roth, F., Waterman, D. A., and Lenat, D. B. (1983). *Building Expert Systems*. Addison-Wesley, Reading, Massachusetts.
- Heckmann, M. and Bell, R. C. (2016). A new development to aid interpretation of hierarchical cluster analysis of repertory grid data. *Journal of Constructivist Psychology*, 29(4):368–381.
- Hennig, C. M., Meilä, M., Murtagh, F., and Rocci, R. (2016). *Handbook of cluster analysis*. Chapman & Hall/CRC, Boca Raton.
- Honey, P. (1979). The repertory grid in action: how to use it to conduct an attitude survey. *Industrial and Commercial Training*, 11(11):452–459.
- Jackson, D. A. (1993). Stopping rules in principal components analysis: A comparison of heuristical and statistical approaches. *Ecology*, 74(8):2204–2214.
- James, W. (1890). *The Principles of Psychology*. Macmillan, London.
- Jankowicz, D. (2004). *The Easy Guide to Repertory Grids*. Wiley, Chichester, UK.
- Kaipainen, M., Zenker, F., Hautamäki, A., and Gärdenfors, P. (2019). *Conceptual Spaces: Elaborations and Applications*. Synthese Library 405. Springer, Cham.
- Kaldis, B. (2008). The question of platonic division and modern epistemology. In Dillon, J. M., Zovko, M.-E., and Doner, J. F., editors, *Platonism and Forms of Intelligence*. Akademie, Berlin.
- Kelly, G. A. (1938). The assumption of an originally homogeneous universe and some of its statistical implications. *Journal of Psychology*, 5:201–208.
- Kelly, G. A. (1955). *The Psychology of Personal Constructs*. Norton, New York.
- Kelly, G. A. (1969). A mathematical approach to psychology. In Maher, B., editor, *Clinical Psychology and Personality: The Selected Papers of George Kelly*, pages 94–113. Wiley, New York.

- Kirkcaldy, B., Pope, M., and Siefen, G. (1993). Sociogrid analysis of a child and adolescent psychiatric clinic. *Social Psychiatry and Psychiatric Epidemiology*, 28(6):296–303.
- Kolodner, J. L. (1993). *Case-based Reasoning*. Morgan Kaufmann, San Mateo, CA.
- Korenini, B. (2014). Consistent laddering: A new approach to laddering technique. *Journal of Constructivist Psychology*, 27(4):317–328.
- Korzybski, A. (1951). The role of language in the perceptual processes. In Blake, R. R. and Ramsey, G. V., editors, *Perception, an Approach to Personality*, pages 170–205. Ronald Press, New York.
- Leach, C., Freshwater, K., Aldridge, J., and Sunderland, J. (2001). Analysis of repertory grids in clinical practice. *British Journal of Clinical Psychology*, 40:225–248.
- Mireaux, M., Cox, D. N., Cotton, A., and Evans, G. (2007). An adaptation of repertory grid methodology to evaluate australian consumers’ perceptions of food products produced by novel technologies. *Food Quality and Preference*, 18(6):834–848.
- Nosofsky, R. (1985). Overall similarity and the identification of separable-dimension stimuli: A choice model analysis. *Perception & Psychophysics*, 38(5):415–432.
- Okabe, A., Boots, B. N., and Sugihara, K. k. (1992). *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, NY.
- Peres-Neto, P. R., Jackson, D. A., and Somers, K. M. (2005). How many principal components? stopping rules for determining the number of non-trivial axes revisited. *Computational Statistics & Data Analysis*, 49(4):974–997.
- Pope, M. L. and Denicolo, P. M. (2001). *Transformative Professional Practice: Personal Construct Approaches to Education and Research*. Whurr, London.
- Rad, A., Wahlberg, O., and Öhman, P. (2013). How lending officers construe assessments of small and medium-sized enterprise loan applications: a repertory grid study. *Journal of Constructivist Psychology*, 26(4):262–279.
- Récanati, F. (2013). *Mental Files*. Oxford University Press, Oxford.
- Reynolds, T. J. and Gutman, J. (1988). Laddering theory, method, analysis, and interpretation. *Journal of Advertising Research*, 28(1):11–31.
- Rosch, E. (1978). Principles of categorization. In Rosch, E. and Lloyd, B. B., editors, *Cognition and Categorization*, pages 27–48. Lawrence Erlbaum, Hillsdale, NY.
- Rosch, E. and Lloyd, B. B. (1978). *Cognition and Categorization*. Lawrence Erlbaum, Hillsdale, NY.
- Shaw, M. L. G. (1980). *On Becoming a Personal Scientist: Interactive Computer Elicitation of Personal Models of the World*. Academic Press, London.

- Shaw, M. L. G. (1981). *Recent Advances in Personal Construct Technology*. Academic Press, London.
- Shaw, M. L. G. and Gaines, B. (1992). Kelly's 'Geometry of psychological space' and its significance for psychological modeling. *New Psychologist*, pages 23–31.
- Shaw, M. L. G. and Gaines, B. R. (1989). Comparing conceptual structures: consensus, conflict, correspondence and contrast. *Knowledge Acquisition*, 1(4):341–363.
- Shaw, M. L. G. and Gaines, B. R. (1996a). Requirements acquisition. *Software Engineering Journal*, 11(3):149–165.
- Shaw, M. L. G. and Gaines, B. R. (1996b). Webgrid: Knowledge elicitation and modeling on the web. In Maurer, H., editor, *Proceedings of WebNet96*, pages 425–432. Association for the Advancement of Computing in Education, Charlottesville, VA.
- Shaw, M. L. G. and Gaines, B. R. (1998). Webgrid II: Developing hierarchical knowledge structures from flat grids. In Gaines, B. and Musen, M., editors, *Proceedings of Eleventh Knowledge Acquisition Workshop*.
- Shaw, M. L. G. and Gaines, B. R. (2005). Expertise and expert systems: emulating psychological processes. In Fransella, F., editor, *The Essential Practitioner's Handbook of Personal Construct Psychology*, pages 87–94. Wiley, Chichester, UK.
- Shaw, M. L. G. and McKnight, C. (1981). *Think Again: Personal Problem-solving and Decision-making*. Prentice-Hall, Englewood Cliffs, NJ.
- Shepard, R. N. (1964). Attention and the metric structure of the stimulus space. *Journal of Mathematical Psychology*, 1(1):54–87.
- Slater, P. (1976). *Dimensions of Intrapersonal Space: Volume 1*. John Wiley, London.
- Slater, P. (1977). *Dimensions of Intrapersonal Space: Volume 2*. John Wiley, London.
- Spencer, H. (1862). *First Principles*. Williams and Norgate, London.
- Stenning, K. and Lambalgen, M. v. (2008). *Human Reasoning and Cognitive Science*. MIT Press, Cambridge, MA.
- Stephenson, W. (1953). *The Study of Behavior: Q-technique and its Methodology*. University of Chicago Press, IL.
- Tan, F. B., Tung, L.-L., and Xu, Y. (2009). A study of web-designers' criteria for effective business-to-consumer (b2c) websites using the repertory grid technique. *Journal of Electronic Commerce Research*, 10(3):155–177.
- Thurstone, L. L. (1935). *The Vectors of Mind: Multiple-factor Analysis for the Isolation of Primary Traits*. University of Chicago science series. University of Chicago Press, Chicago.

- Tofan, D., Avgeriou, P., and Galster, M. (2014). Validating and improving a knowledge acquisition approach for architectural decisions. *International Journal of Software Engineering and Knowledge Engineering*, 24(4):553–589.
- Torgerson, W. S. (1958). *Theory and Methods of Scaling*. Wiley, New York.
- Tversky, A. (1977). Features of similarity. *Psychological Review*, 84(4):327–352.
- Wason, P. C. (1968). Reasoning about a rule. *Quarterly Journal of Experimental Psychology*, 20(3):273–281.
- Whitehead, A. N. (1929). *Process and Reality: An Essay in Cosmology*. Free Press, New York.
- Yorke, D. M. (1978). Repertory grids in educational research: Some methodological considerations. *British Educational Research Journal*, 4(2):63–74.
- Yorke, D. M. (1983). Straight or bent? an inquiry into rating scales in repertory grids. *British Educational Research Journal*, 9(2):141–151.
- Young, S. M., Edwards, H. M., McDonald, S., and Thompson, J. B. (2005). Personality characteristics in an xp team: a repertory grid study. *SIGSOFT Software Engineering Notes*, 30(4):1–7.
- Zenker, F. and Gärdenfors, P. (2015). *Applications of Conceptual Spaces: The Case for Geometric Knowledge Representation*. Springer, Cham.

Most of our publications cited in the references are freely available on the web