

User manual for the QCA(GUI) package in R

Adrian Duşa

University of Bucharest

December 2006

The author wishes to thank John Fox, McMaster University for his willingness to share the code he wrote for the original Rcmdr package, as well as for his invaluable help in adapting it to the QCA needs. Little would have been possible without the open source movement where the R community is one of the most active. I would also like to thank the R-help list members at large, especially Gabor Grothendieck, Andy Liaw and Martin Maechler for their suggestions and solutions in the key parts of the R based algorithm.

Send correspondence to Adrian Duşa, 1 Schitu Magureanu Bd., 050025
Bucharest, Romania (email adi@sas.unibuc.ro)

User manual for the QCA(GUI) package in R

In the QCA field, there are only two other software that I am aware of: the first is *fsQCA* developed by Charles Ragin (2006) and his team at the University of Arizona and the second is *Tosmana* developed by Lasse Cronqvist (2006) at the University of Marburg in Germany. Both of them have complex code and the speed of the calculations is greatly improved in the compiled versions. However they both have a couple of drawbacks which lead me to write a package in R: on one hand their source code is either not open or written in a language that very few people understand, and on the other hand the programs are written exclusively for Windows where many researchers now use different other operating systems.

I believe that in order to increase the development speed, people need a common environment and R (2006) is a perfect solution. R is a phenomenon in the last years and many people believe it has already become the *lingua franca* in the statistical research.

Both QCA and QCAGUI packages are written in the R language, plus Tcl/Tk for the user interface. In order to use them one needs to install R, which has versions for almost all existing platforms, from Windows and Linux to Solaris and MacOS. Using the QCA package requires nothing more than the ability to use R. This guide is merely an extension of the existing R manuals, on-line on the CRAN website (<http://cran.r-project.org/manuals.html>) and PDF versions shipped with the installation kit. Numerous other books are written for the R language, most of them for statistical analysis. Excellent introductions to R can be found in Paradis (2005), Verzani (2006) and Burns (1998).

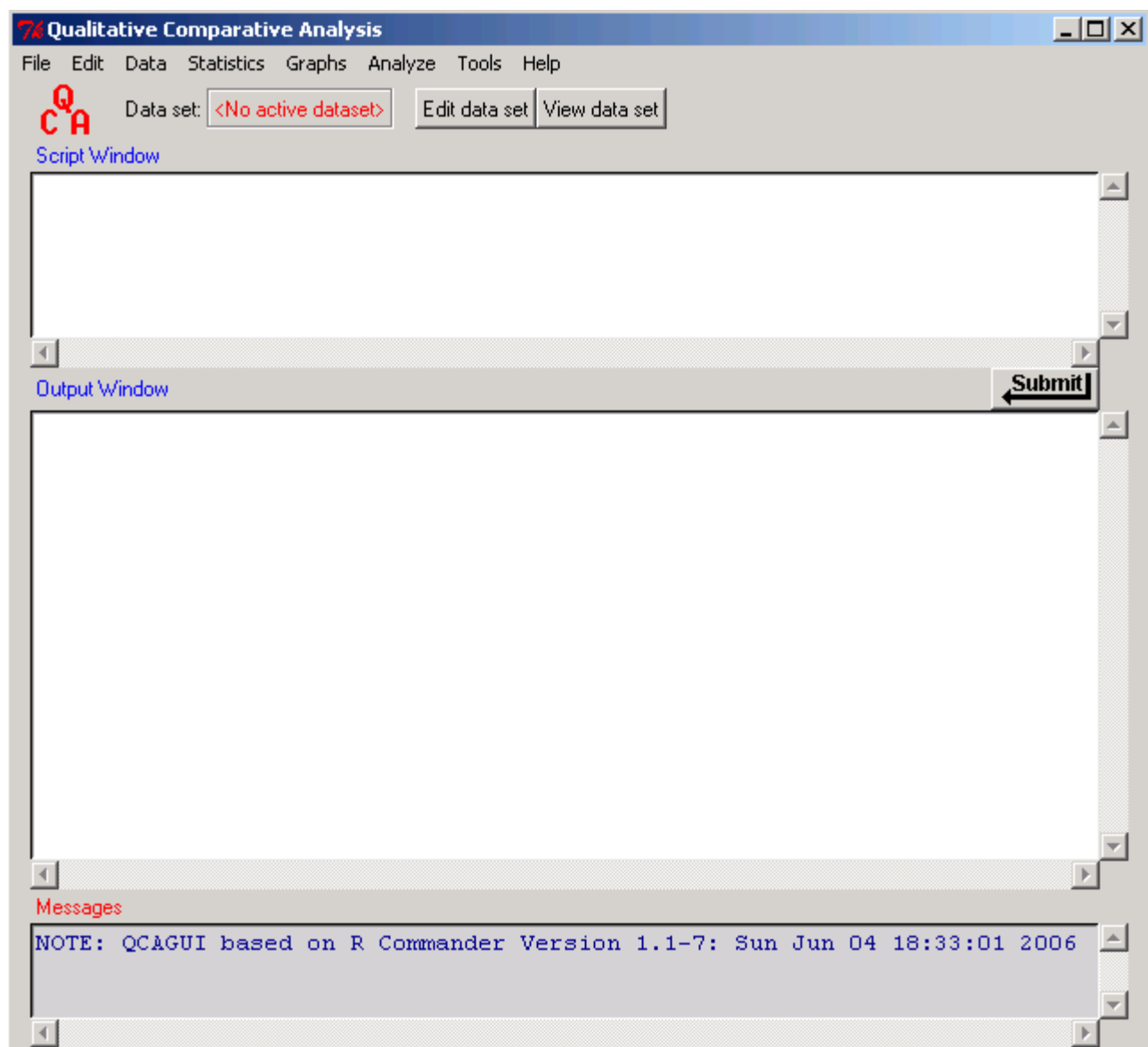
I am going to assume that users reading this guide are at least familiar with the R environment. Examples will be given both for the plain QCA package and for its companion, QCAGUI. The command used for loading the QCA package is:

```
library(QCA)
```

Similarly, the command used for loading the QCAGUI package is:

```
library(QCAGUI)
```

Because QCAGUI depends on QCA, the second command also invokes the first one; if everything goes right (all depending packages are installed) the graphical user interface should appear.



1. Opening data from external files

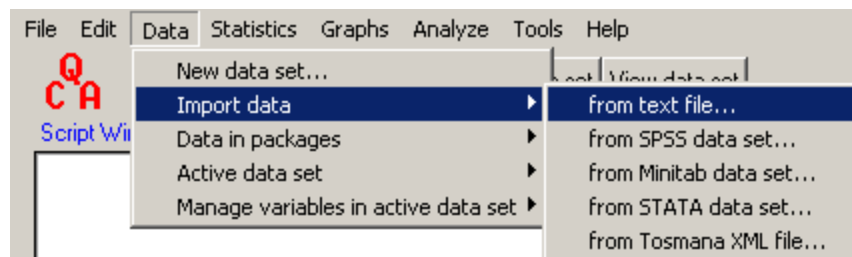
Unlike many proprietary software who deliver specific file formats, R does not encourage such thing. Instead it is extremely versatile in opening almost any existing file format. It can read SPSS, SAS and STATA formats, as well as Excel, Access and DBF. It can also connect to more advanced database engines like MySQL, PostgreSQL or even Oracle. Even with all this fancy stuff, R is happy to work with simple ASCII files with various delimiters: tab, comma, space etc.

Users may wonder how come a mighty software like R does not have its own file format. The reason, in my opinion, lies in its strength. Most (if not all) other statistical software only handle dataframes (with variables as columns and cases as rows); R is far superior because it can handle all the other objects used in programming: scalars, vectors, matrices, arrays and lists. It would be easy to create a file format for a dataframe, but it would not be the same when saving a multidimensional list. This is why R offers a variety of functions, both for reading and for writing different types of objects.

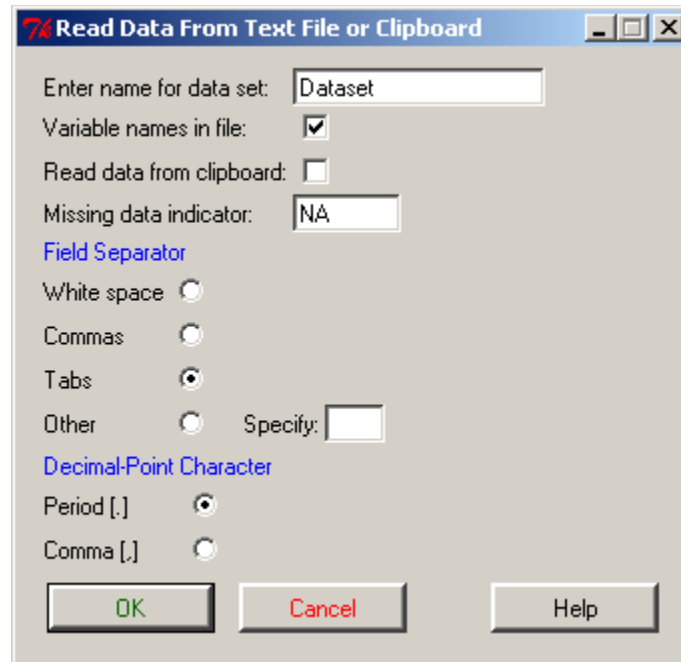
Both fsQCA and Tosmana can read simple delimited files. Presuming a tab delimited file (where the first row contains the name of the variables) called “myfile.dat” is stored in R's working directory, an example command to read it is:

```
Dataset <- read.table("myfile.dat", header=TRUE, sep="\t")
```

The user interface does a similar job; first select the appropriate menu (Import data / from text file...):



In the new window the user can set the various details; the name of the object to be created is by default “Dataset” and it can also be changed:



It is worth mentioning that R reads all the data in the computer's memory (RAM), and it can work with multiple objects in the same time, therefore naming the objects is an important step. Following Rcmdr, the QCAGUI package works as follows: the settings clicked in the command window are transformed into arguments for the written command, which is pasted in the “Script window” and automatically submitted to the “Output window”. When reading a different dataset, the user can choose between selecting the menus or simply change a few arguments in the command line from the Script window.

The “Import data” menu shows SPSS, Minitab, STATA and Tosmana in addition to the regular, delimited text file. The list is certainly not exhaustive, R having many more functions to import data. For example, the fsQCA format uses two different types of files: .QDM for the data and .QVN for the list of the variable names. The .QDM file is nothing but a plain, fixed width ASCII file which can be read using the function `read.fwf()`

Using fixed width format has a major drawback: it expects all values from the same variable (column) to have the exact same length. For this reason fsQCA fails to

save the case (row) names in its file format (as they may have different lengths) and it works best with comma, space or tab separated text files.

Starting with version 0.2-5, QCA has an additional function for opening Tosmana's XML file format. Another advantage of R over the other QCA software is the ability to allow case names in the same way as variable names. Just like columns, each row can have a name thus saving from the need of a separate variable to describe cases. The import function detects the variable referred to as “Case descriptor” in Tosmana and it assigns that variable as row names, leaving a clean data set containing the relevant data only.

If any data set comes with a special variable that contains the case names, it is very easy to transform it into row names, either via the menu:

Data

Active data set

Set case names...

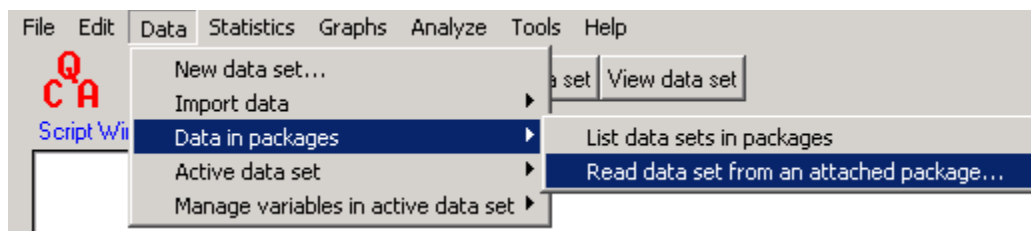
or via command line:

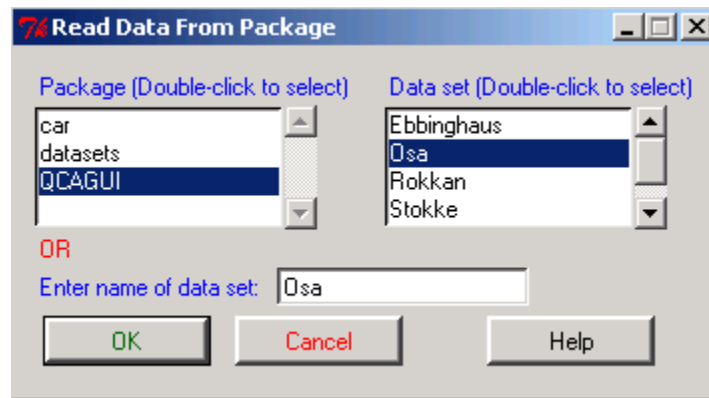
```
rownames(Dataset) <- Dataset$XYZ
```

where XYZ is the variable from Dataset containing the case names.

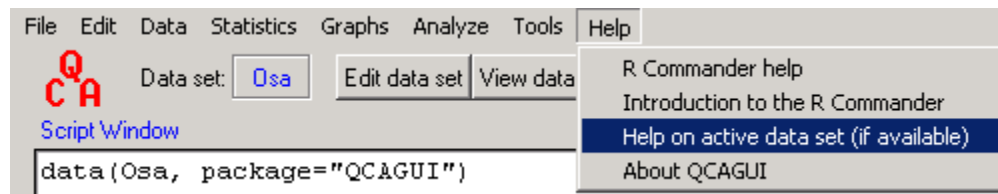
2. Opening data from attached datasets

In R it is mandatory to create a help file for each and every function contained in a package (except for the internal functions which are not user visible). Along with these help files it is also customary to wrap the package with a few test datasets for demonstration purposes. The QCA suite has about 5 datasets included, which can be opened using “Read data set from an attached package...”:

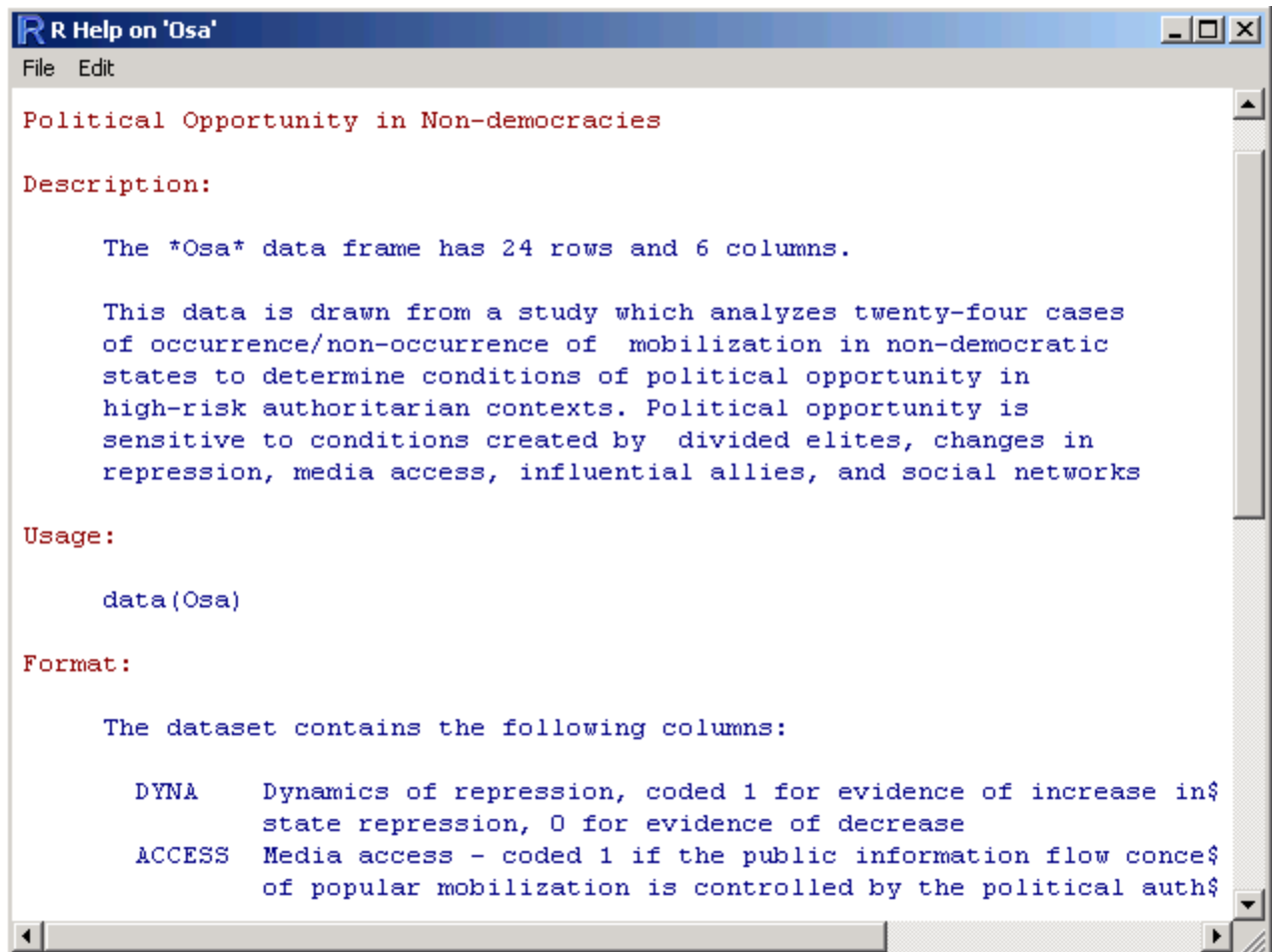




The advantage of having datasets into an R package is the documentation provided along with the data itself, which serves as a metadata repository where useful information can be easily found. The user can reach this information using the help menu. The next window presents the relevant information for the “Osa” data set we just opened:



The help window contains minimal, yet critical information: a description of the data, how it was produced, the main hypotheses followed, then each variable is described down to the last value and finally the bibliographical reference where the original data was used.



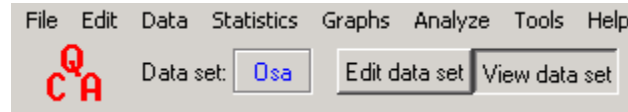
This kind of documentation is crucial to a proper use of the data. It can certainly be found in the relevant bibliography but wrapping it up with the data makes its use considerably easier.

3. Working with the data

Up to this point the menus and the commands used are fairly similar to any other software. After a successful import, the user may be a little perplexed as all the other software present (without fail) a data window with the familiar rectangle of rows and columns. However in R there is no such thing (at least not presented automatically) and

for a very good reason. If the user feels urged, there are two buttons to edit and to view the data, right below the main menu.

Push the “View data set” button:



Which opens this window:

74 Osa						
	DYNA	ACCES	INFLU	ELITE	SOCIAL	OUT
SP52	1	0	0	0	0	0
EG53	0	1	0	1	0	1
PL56	0	1	0	1	1	1
PO58	1	0	1	1	1	1
PL60	0	0	1	0	0	0
PO62	1	0	1	1	1	1
SP62	0	1	1	1	1	1
SA64	1	0	0	0	0	0
GR68	0	1	1	1	1	1
HO68	0	1	1	1	1	1
PL68	1	0	1	1	1	1
CH73	1	0	1	0	1	1
UR73	1	0	0	0	0	0
AR77	1	1	0	0	1	1
SA76	1	1	0	0	1	1
BR77	0	0	1	0	1	1
RO77	1	0	0	0	1	1
RO78	1	0	0	0	0	0
IR80	0	0	1	1	1	1
CH83	0	1	1	1	1	1
CH83LCP	0	1	0	1	0	0
RO87	1	0	0	0	1	1
BU88	1	1	0	0	0	1
CN89	0	1	1	1	1	1

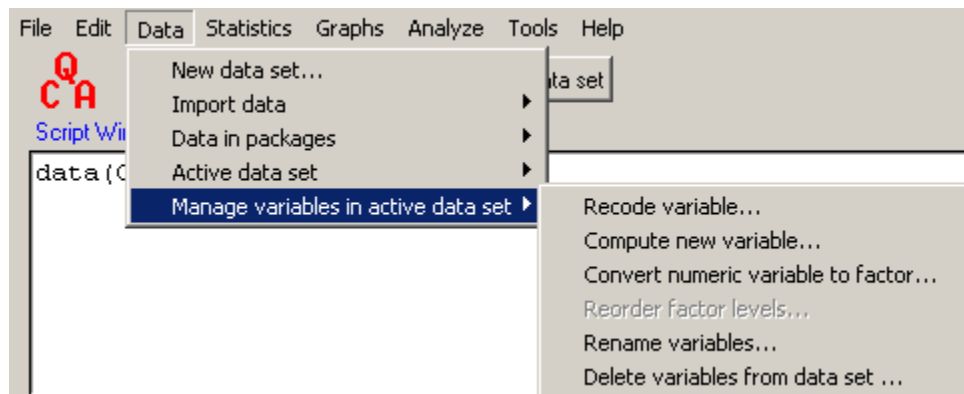
The “View data set” button produces a window that has an immediate relaxing effect: the data was successfully imported. At first it might be bothering not to see the data at all times, but there is actually no need for this. Seeing the data is like drug dependence; it may not be obvious for QCA data having a fairly small number of both cases and variables, but for large datasets (as in the quantitative world) the data view presents only a tiny rectangle from a much larger dataset. It is just a nothing but a visual

trick that actually slows the user down. On the other hand, R users usually interrogate their data; for example, if a user wishes to see if there are any missing values in the XYZ variable, a very simple command is:

```
any(is.na(XYZ))
```

This is actually a double command; first `is.na()` checks every element of XYZ returning TRUE if it's missing and FALSE otherwise, and second `any()` checks if any of the elements returned by `is.na()` is TRUE. It is almost a natural language.

The user interface has menus for any data transformation needed: recode variables, compute new variables, rename variables or even delete variables:

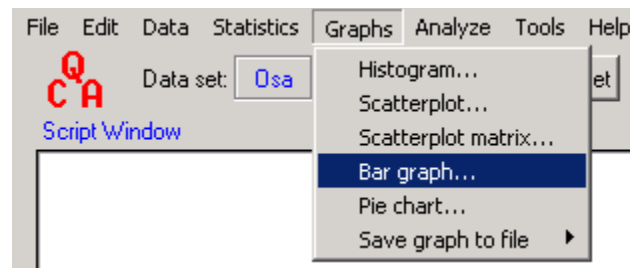


R has a special way of dealing with qualitative, categorical variables. They are called *factors* and R prevents the user from computing for example a mean on such a variable. The QCAGUI package differs from the original Rcmdr package by allowing such analyses based on a few theoretical grounds. The crisp sets QCA data is binary, having only two values: 0 and 1 (0 meaning the absence and 1 the presence of a condition). This kind of categorical variable is special because it allows computing proportions, an important property especially for fuzzy sets QCA. Binary data is both a qualitative and a quantitative variable therefore QCAGUI allows both types of numerical analyses: tables of frequencies (specific to categorical data) and means or proportions (specific to quantitative data).

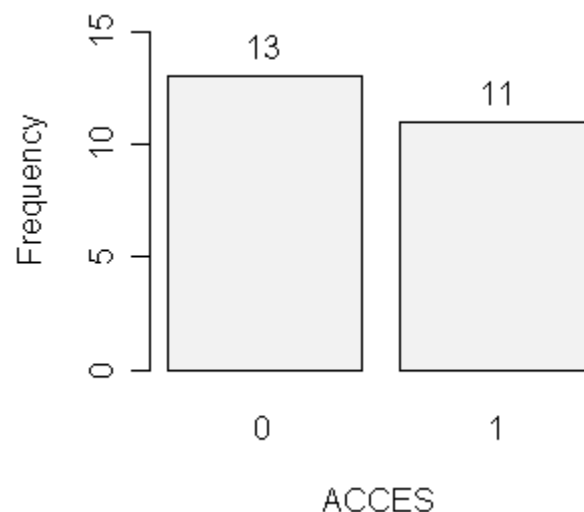
Simple statistics (frequency distributions, tables of frequencies, numerical summaries) can be performed using the Statistics menu. The original Rcmdr package

has many more statistical functions but only a few were preserved because QCA has essentially little to do with statistics.

Graphs are also easy to produce and many people believe that R's graphics engine is the best in the world. It is one of the very few software in general and almost certainly the only one in the statistics field that can produce plenty of the existing graphic formats, among which the most powerful is the vector format. In its Windows version, R can even produce WMF – Windows Meta-Files which can be used as easy as right-clicking the graph window, selecting the desired format and pasting it into a Word document.



Selecting the variable ACCESS from the Osa dataset, a bar chart is produced in a separate window:



The vector format has an amazing property of not depending on pixels (like the standard image formats); instead it can be enlarged or shrunk to any dimension without any loss of quality for different resolutions.

More over, the graphics window can be set to any dimension on width and height, using a simple command. For example, to set the output graphics window to 7x7 inches prior to producing the graphic the command is:

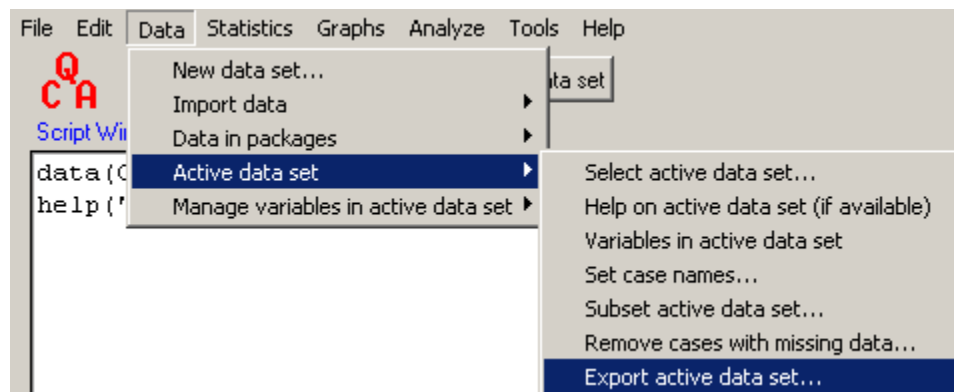
```
X11(width=7, height=7)
```

3. Exporting data

This menu may seem hidden but there is a logical explanation for this. Unlike every other software, R can work with multiple data sets at the same time, storing data in the memory. More precise, R works with multiple *objects* in the same time because data is not restricted to the rectangular form of a data set; there are also scalars, vectors, matrices, lists or arrays.

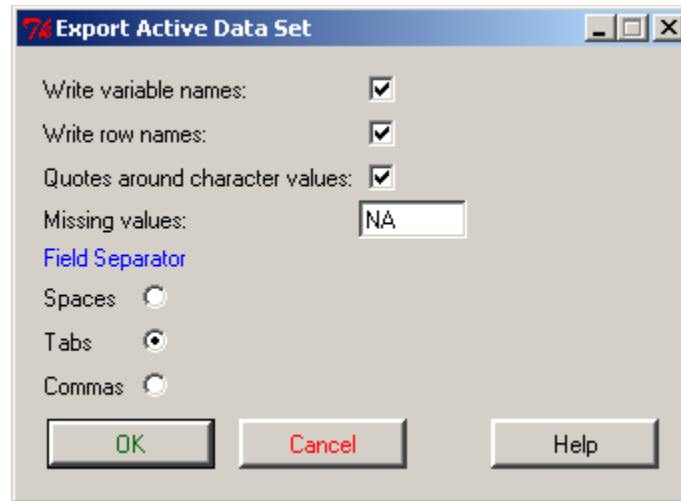
Exporting data raises therefore a reasonable question: which data?

If considering data as *data sets*, the user may wish to save / export the data set currently working with, and that is the *active data set*. The export command is therefore associated with the “Data / Active data set” menu:



The menu presents a handful of options for saving the dataset in a plain ASCII file with various delimiters. The user can choose whether or not to write the variables names in the output file, and the same thing about the row (case) names. Plain text

ASCII files are preferred because these are portable for any software and any operating system. Three most common delimiters (Field separators) may be chosen: spaces, tabs and commas.



At this point the user should be warned about R's way of writing the row names: it does it in the first column of the output file, but it does *not* write a column name; instead it writes a first delimiter (the one chosen by the user). In order to open the output file with other software, the user has two choices: either to create a new variable taking its values from the row names or to manually modify the output file by writing a variable name in the first column.

3. Performing crisp sets QCA

The QCA package is in its early stages of development. Currently the only algorithm implemented is the exact Quine-McCluskey as described in McCluskey (1956) and Ragin (1987). This algorithm is guaranteed to produce a certain solution but it has one weakness: being exhaustive it consumes memory exponentially as the number of conditions increase.

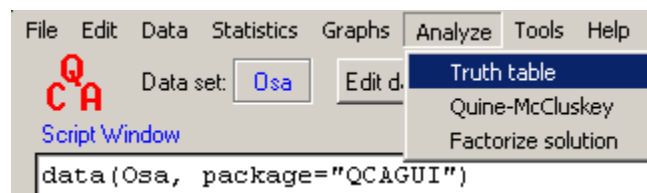
The R algorithm starts by defining a matrix of all possible combinations of two cases that can be compared, minimizing those that differ by only one literal. In several iterations it produces implicants that are further minimized until no minimization is possible. The final implicants are called *prime implicants*. The iterative process of

comparing every possible combination of two implicants is best described as a bubble that quickly inflates at the beginning then deflates until it finds the prime implicants. Depending on the number of conditions entered in the analysis, in its moment of maximum growth the bubble-matrix can take up to several tens of millions of rows, choking an average computer.

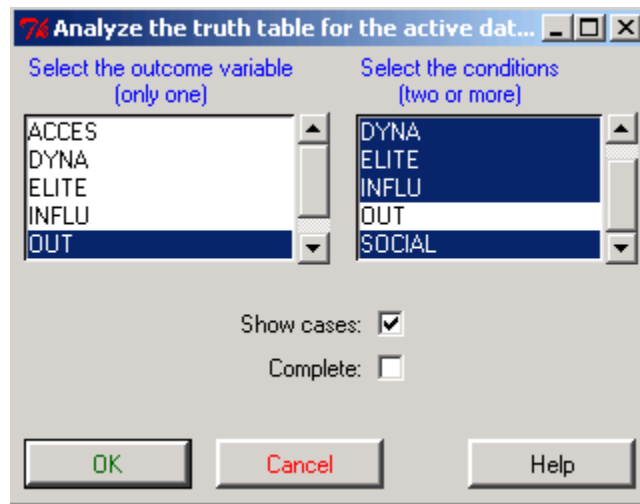
This algorithm works very well with 8 or 9 conditions but it cannot handle more. A different solution is worked upon to handle more than 9 conditions, possibly but not necessarily based on graphs. There are some existing solutions, one provided by the engineers from Berkeley called Espresso which unfortunately was abandoned in 1995, and another promising one provided by Fišer and Hlavička (2003) called BOOM (from boolean minimization) which is followed by another optimization called FC-Min for multi-output minimizations.

QCA(GUI) offer three main functions for the classic exact algorithm. First it is useful to inspect the data by creating the truth table and second to perform the actual minimization process which results in the final minimum solution. Third, as far as I know QCA is the only software that offers more than just a solution: it computes all possible common factors of a solution through the `factorize()` function.

3.1. Creating the truth table



The menu uses the function `truthTable()` which starts by creating the matrix with all 2^k combinations of 0 and 1 for each condition. This is essentially the entire, exhaustive truth table. It expects the user to indicate which variable is the outcome and which variables are the conditions (if not specified, all variables but the outcome are considered conditions).



The user is free to decide whether or not to print the case names near the truth table combinations and similarly whether or not to print the entire matrix with the 2^k combinations.

The function finds the observed combinations (the input data) among all possible ones, and establishes a value for the outcome following a very simple decision process:

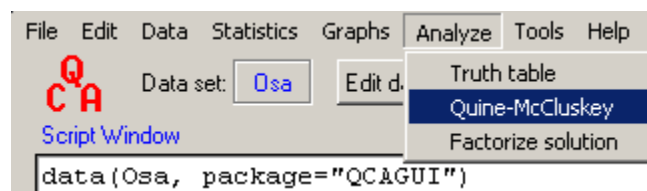
- if all the observed combinations are consistent with respect to the outcome value (either 0 or 1) then the outcome will be set to that value
- if any two cases with the same combination of conditions present both a 0 and a 1 then the outcome is a contradiction and coded with the letter “C”
- for any combination of conditions that is not found in the data the outcome is set to missing and coded with a question mark “?”

By default, the names of the conditions are replaced with letters (for reasons of saving space and for consistency with the other QCA software) and the output variable is replaced with “OUT”. A legend is printed right above the truth table showing which letter corresponds to which condition. Two more columns are displayed: “freq0” and “freq1” showing the frequency of the outcome equal to 0 or to 1 for a specific combination of conditions.

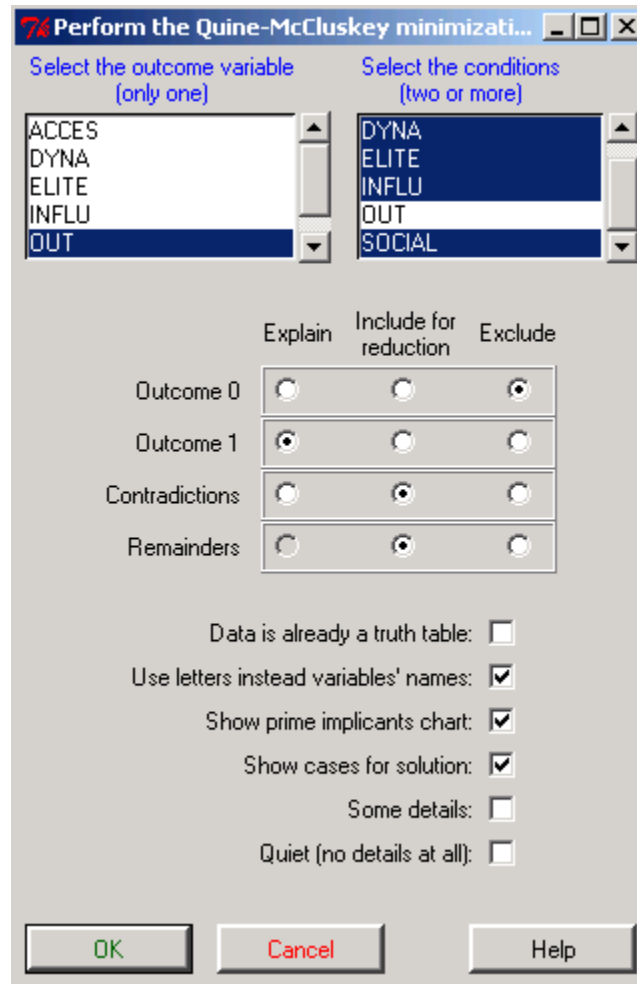
	A	B	C	D	E	OUT	freq0	freq1	cases
3	0	0	0	1	0	0	1	-	PL60
4	0	0	0	1	1	1	-	1	BR77
8	0	0	1	1	1	1	-	1	IR80
9	0	1	0	0	0	0	4	-	SP52, SA64, UR73, RO78
10	0	1	0	0	1	1	-	2	RO77, RO87
12	0	1	0	1	1	1	-	1	CH73
16	0	1	1	1	1	1	-	3	PO58, PO62, PL68
21	1	0	1	0	0	C	1	1	EG53, CH83LCP
22	1	0	1	0	1	1	-	1	PL56
24	1	0	1	1	1	1	-	5	SP62, GR68, HO68, CH83, CN89
25	1	1	0	0	0	1	-	1	BU88
26	1	1	0	0	1	1	-	2	AR77, SA76

Analyzing the truth table is crucial prior to the minimization process because the researcher needs to assess how to deal with contradictions and remainders. Contradictions and negative cases have been the subject of a long debate in the QCA world, especially in the new approach to the fuzzy sets described by Ragin (2000). Among many others Lieberman (1991, 1994), Savolainen (1994), Goldstone (1997), Ragin (1997, 2002) have debated over the deterministic vs. probabilistic nature of the QCA algorithm. For example, if the truth table shows 15 positive cases and only 1 negative case, according to the crisp sets QCA that combination is a contradiction, whereas in the fuzzy sets approach the ratio of 15:1 might be considered high enough in order to refute the one negative case. Depending on this analysis the researcher prepares the input data for the boolean minimization process.

3.2. Applying the Quine-McCluskey algorithm



This menu opens a window very similar to the other QCA software. The user can choose what to explain, exclude or include for reduction. In this version of the algorithm there are four types of outcomes: equal to 0, equal to 1, contradictions and the rest (the remainders, missing combinations from the observed data set).



Using the Osa dataset as an example, I chose to include the contradictions and the remainders for reduction, leaving out only the combinations resulting in the absence of the outcome. The command window also offer some other useful options, passed to the `qmcc()` function as arguments:

- By default the `qmcc()` function first calls the `truthTable()` function in order to determine the unique combinations in the data showing positive, negative or

contradictory cases. If the input data is already a truth table (it is a common practice to work like that) this step is not necessary therefore skipped

- The user can decide to use the original names of the conditions or to replace them with letters. The default and recommended way is to replace the names with letters, a legend being printed along with the final solution
- For many QCA researchers inspecting the prime implicants chart is an important step, but printing it can be skipped if the user only wants just a quick solution
- Similar to the truth table case, the user can decide to print the case names associated with each and every prime implicant from the final solution. In my opinion it is always a good idea to see whether the cases studied and the theory about them are consistent with the solution provided by the algorithm
- For the interested users who need more information about each iteration involved in the minimization process, there is another option available which will provide some details about how many prime implicants have entered in the iteration, how many possible paired comparisons are there and how many of them differ by only one literal
- The final option acts like a radio button for all the other options that involve printing additional information. Activating it disables the three check boxes above it but it remembers which one was clicked. Just like any radio button, deactivating the Quiet option will reactivate only those options that were active before

Again similar to the “Truth table” menu, the only variable mandatory to select is the outcome variable in the left list box, which accepts only one selection. The right list box accepts multiple selections for the relevant conditions that are to be entered in the minimization process.

The algorithm behind this function is very simple, taking advantage of R's vectorizing power. First, it selects only those rows from the truth table that correspond to the options selected in the menu. To find the pairs of rows that differ by only one literal, the algorithm uses the function `dist()` from the standard *stats* package, which

computes the Manhattan distance between any two rows, returning a matrix of distances: if a pair differ by only one literal the distance is set to 1, if it differ by two literals the distance is set to 2 and so on.

The next step is to select the values of 1 from this matrix, using the function `which()` and its argument `arr.ind=TRUE` in order to preserve the row/column positions: if a value of 1 is found at the intersection of the 4th row and the 6th column, it means that rows 4 and 6 from the truth table differ by one literal. The result of this process is another matrix of row positions from the initial truth table that differ by only one literal. This information is further used to the actual minimization process: the minimized literals are set to -1 and a new matrix with a first set of implicants is produced.

Each iteration uses the matrix of unique implicants remained from the previous iteration, producing a new set of implicants until no further minimizations are possible. The implicants remained are the prime implicants which are further reduced using the prime implicants chart shown in the output:

	abcDE	abCDE	aBcdE	aBcDE	abcDE	AbCdE	AbCDE	ABcde	ABcdE
A	-	-	-	-	-	x	x	x	x
C	-	x	-	-	x	x	x	-	-
E	x	x	x	x	x	x	x	-	x
bd	-	-	-	-	-	x	-	-	-
BD	-	-	-	x	x	-	-	-	-

Solution: A + E

A correspond to lines: EG53; PL56; SP62; GR68; HO68; AR77; SA76; CH83;
CH83LCP; BU88; CN89

E correspond to lines: PL56; PO58; PO62; SP62; GR68; HO68; PL68; CH73;
AR77; SA76; BR77; RO77; IR80; CH83; RO87; CN89

A is the presence of ACCES
E is the presence of SOCIAL

Solving the prime implicants chart was another challenge for the QCA algorithm in R. The solution found is elegant, using the linear programming function called `lp()` from the *lpSolve* package. This function finds the minimum number of prime implicants needed to cover all columns (the observed combinations included in the analysis), the rest of the algorithm trivially using that number to find all combinations of prime implicants covering all columns.

In the example presented, the number returned from the `lp()` function is 2 and the only combination of two prime implicants that cover all columns is A + E. Since the program was asked to print the case names corresponding to the prime implicants from the final solution, this is the next piece of output printed on the screen. Upper case letters generally represent the presence of a condition, whereas lower case letters represent the absence of a condition.

This algorithm is the first version of a series intended to become both fast and accurate. Although the limit of 9 conditions is fairly sufficient for any decent QCA research, the next version definitely needs to overcome this limitation. The next step after that is to adapt the fuzzy sets and multi-value family of algorithms.

4. Factoring a solution

One of the long awaited functions in this package is designed to help the user making sense of the solution returned by the boolean minimization process. Sometimes this solution can have four or five different prime implicants, each consisting of a range from one to several conditions. It would be very helpful to factor those conditions which are common for multiple prime implicants. Let us suppose that a final solution may be this (relatively simple) one:

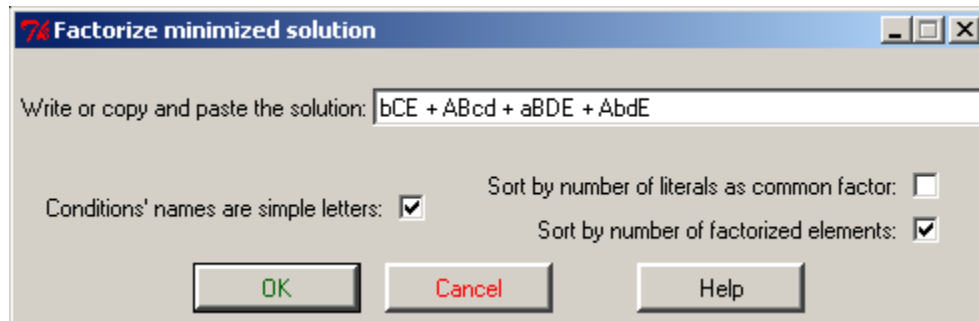
$$AB + BE + ACD$$

Extracting the common literals in the prime implicants finds two possible solutions:

$$\text{Solution 1: } B(A + E) + ACD$$

$$\text{Solution 2: } A(B + CD) + BE$$

More complicated combinations of prime implicants, consisting of both presence and absence of conditions, makes the extraction even more difficult. The third option from the Analyze menu is called “Factorize solution” and it opens the following window:



The first and most important part is to write the solution in the text box; the recommended way is to copy and paste it directly from the output window. In this example we chose to factorize a slightly more complicated string:

$$bCE + Abcd + aBDE + AbdE$$

The program splits the input string according to the key separator “+” and tries hard to find all possible combinations of common factors using a recursive algorithm

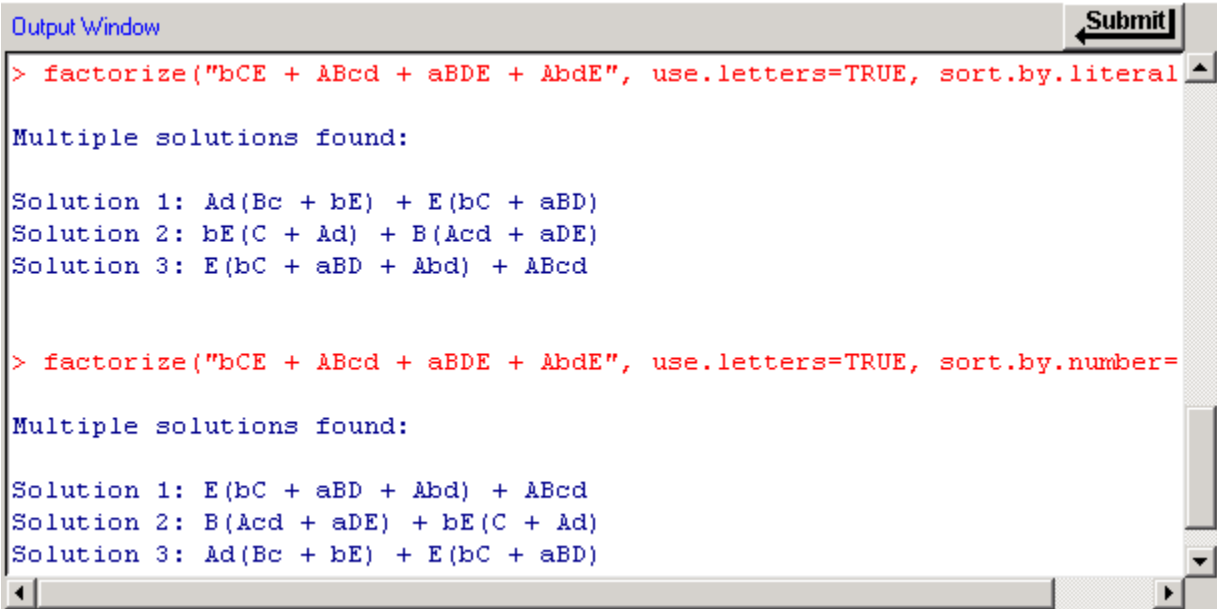
which stores the result from each iteration in a multidimensional list. After it finishes exploring every possible combination of literals, it compares the results and preserves only the unique ones (some solutions might be identical, only the route obtaining it being different).

If the names of the conditions are not simple letters, they are written in the form: $\text{second*THIRD*FIFTH} + \text{FIRST*second*third*fourth}$... and so on.

In this form there are two kinds of key separators: the first one is the plus sign “+” to separate the prime implicants and the second is the star sign “*” which separates the names of the conditions (the literals). There are two different ways to order a factor solution:

- by the number of literals in the common factor
- by the number of factorized elements

In both cases, the algorithm presents all possible solutions, only the list is sorted according to the preferred option.



```

> factorize("bCE + ABcd + aBDE + AbdE", use.letters=TRUE, sort.by.literal

Multiple solutions found:

Solution 1: Ad(Bc + bE) + E(bC + aBD)
Solution 2: bE(C + Ad) + B(Acd + aDE)
Solution 3: E(bC + aBD + Abd) + ABcd

> factorize("bCE + ABcd + aBDE + AbdE", use.letters=TRUE, sort.by.number=

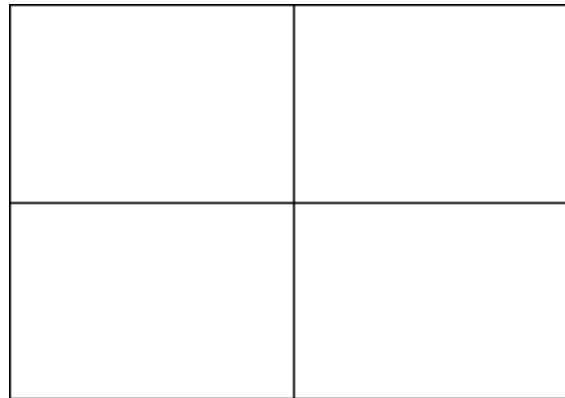
Multiple solutions found:

Solution 1: E(bC + aBD + Abd) + ABcd
Solution 2: B(Acd + aDE) + bE(C + Ad)
Solution 3: Ad(Bc + bE) + E(bC + aBD)
  
```

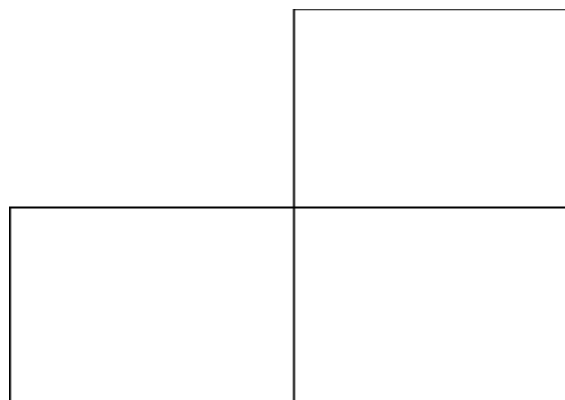
The next version of this function intends to find the best combination of common factors, which *highlights* a certain literal in its presence and absent states, as described in Ragin (1997, p.101).

5. The graphical representation of the truth table

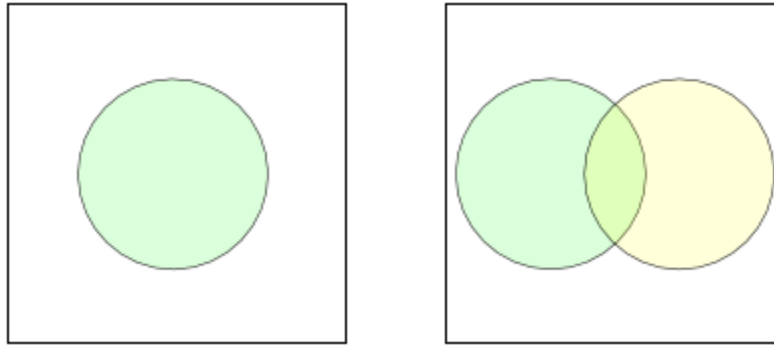
The current standard way of representing truth tables uses a rectangle for each variable (condition) included in the analysis. In my opinion this is not a very good idea, for several reasons. First, one needs to understand how they are constructed. Let's take for example this representation of two sets and their intersection:



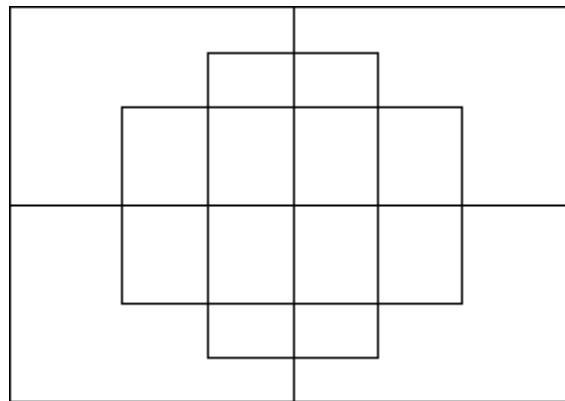
At a first glance it is not quite obvious which exactly the sets are, where do they intersect and what the space is outside both of them. The researcher is forced to a real effort of imagination to understand that in fact the sets and their intersection look like this:



In the case of Tosmana there are some guiding lines and the sets are indeed represented by their combination in the truth table (where 11 means their intersection and 00 the space outside of both), but in my opinion the sets are best represented graphically with the good old Venn diagrams. For one or two sets, the circular representation is so much easier to perceive:



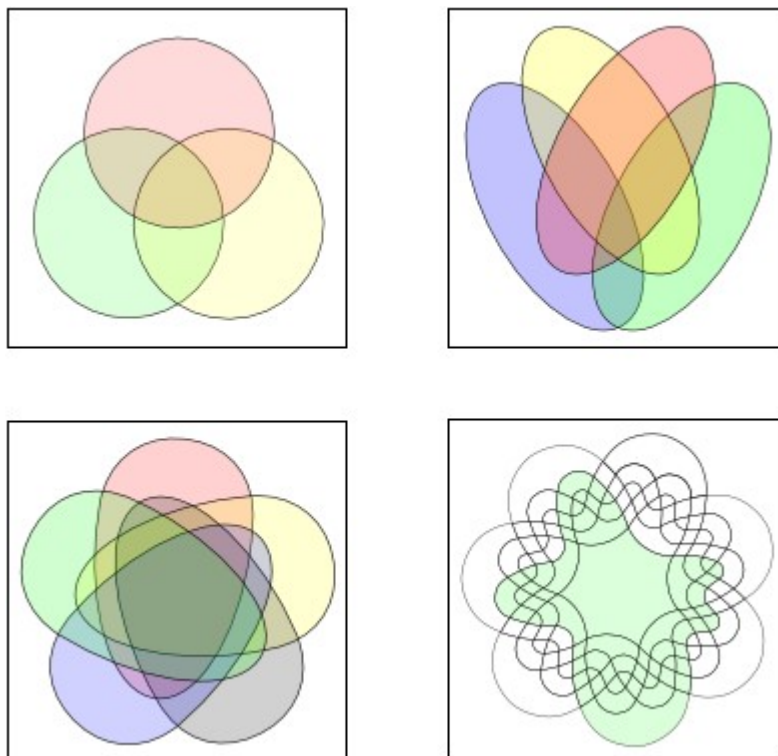
The rectangular representation has another big disadvantage: there are a finite number of sets that can be superposed in order to ensure all possible combinations of intersections (2^k , where k is the number of conditions). In fact, the number of rectangles to be superposed is limited to 4:



For 5 sets, the fifth would have to be broken in half, one for the upper and the other for the lower region (which is the current behavior in Tosmana) and for more than five sets a graphical representation is impossible. This is where curve areas prove their power, as they can take any shape imaginable in order to create a suitable intersection.

For a long time it was thought that a representation beyond three circles is impossible; Venn himself discovered the four ellipses late during his life. Recently a new combinatorics movement has emerged and researchers working with mathematical algorithms have managed to find some extremely interesting solutions.

Ruskey and Weston (2005) have even created a special survey where numerous solutions are presented in a single repository of Venn Diagrams. They show that it is in fact possible to create such diagrams for virtually any number of sets, only the human eye has difficulties discerning separate intersections of more than 11 sets. For prime number of sets, symmetric Venn diagrams can be created. The next picture has representations of three, four, five and seven sets:



The future versions of this package are planned to cover all these diagrams, possibly using packages that produce maps. Each intersection can be considered a separate polygon which can be filled according to particular results of the boolean minimization.

REFERENCES

- Burns, Patrick (1998). S Poetry. Retrieved from <http://www.burns-stat.com> in december 2006.
- Cronqvist, Lasse (2006). Tosmana - Tool for Small-N Analysis [Version 1.255]. Marburg. Retrieved from <http://www.tosmana.net> in december 2006.
- Fišer, Petr and Jan Hlavička (2003). "BOOM - A Heuristic Boolean Minimizer", Computers and Informatics, 22(1): 19-51. Retrieved from <http://service.felk.cvut.cz/vlsi/pubs.html> in June 2006
- Goldstone, Jack A. (1997). "Methodological issues in comparative macrosociology", Comparative Social Research, 16: 107-120
- Lieberson, Stanley (1991). "Small N's and big conclusions: an examination of the reasoning in comparative studies based on a small number of cases", *Social Forces*, 70: 307-320
- _____ (1994). "More on the uneasy case for using Mill-type methods in small-N comparative studies", *Social Forces*, 72: 1225-1237
- McCluskey, Edward. J. (1956). "Minimization of Boolean Functions", The Bell System Technical Journal, 35(5): 1417:1444
- Paradis, Emmanuel (2005). R for beginners. Retrieved from <http://cran.r-project.org/doc/contrib/> in November 2005.
- R Development Core Team (2006). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. Retrieved from <http://www.R-project.org> in december 2006.
- Ragin, Charles C. (1987). The Comparative Method. Moving Beyond Qualitative and Quantitative Strategies. University of California Press, Berkeley and Los Angeles, California.
- _____ (1997). "Turning the tables: how case-oriented research challenges variable-oriented research", Comparative Social Research, 16: 27-42
- _____ (2000). Fuzzy set social science. The University of Chicago Press.

- ____ (2006). User's Guide to Fuzzy-Set / Qualitative Comparative Analysis 2.0.
Tucson, Arizona: Department of Sociology, University of Arizona.
- Ruskey, Frank and Mark Weston (2005). "A Survey of Venn Diagrams", *Electronic Journal of Combinatorics*. Retrieved from <http://www.combinatorics.org/Surveys/> in november 2005.
- Savolainen, Jukka (1994). "The rationality of drawing big conclusions based on small samples: in defence of Mill's methods", *Social Forces*, 72: 1217-1224
- Verzani, John. Simple R. Retrieved from <http://www.math.csi.cuny.edu/verzani/tutorials/simpleR/> in december 2006.