Qual Quant (2010) 44:99–113
DOI 10.1007/s11135-008-9183-x

A mathematical approach to the boolean minimization problem

Adrian Duşa

Published online: 1 October 2008

Abstract Any minimization problem involves a computer algorithm. Many such algorithms have been developed for the boolean minimizations, in diverse areas from computer science to social sciences (with the famous QCA algorithm). For a small number of entries (causal conditions in the QCA) any such algorithm will find a minimal solution, especially with the aid of the modern computers. However, for a large number of conditions a quick and complete solution is not easy to find using an algorithmic approach, due to the extremely large space of possible combinations to search in. In this article I will demonstrate a simple alternative solution, a mathematical method to obtain all possible minimized prime implicants. This method is not only easier to understand than other complex algorithms, but it proves to be a faster method to obtain an exact and complete boolean solution.

Keywords Qualitative comparative analysis · Boolean minimization · Computer algorithms · Small-N research

1 Introduction

Shannon's (1938) master thesis had a dramatic influence on science and modern technology. His paper layed the theoretical foundation for building computers and subsequently to thousands of other applications; admitting that computers play an important role in our lives, it was one of the most important papers in the last century, spreading to every corner of any developed society.

About 20 years later, McCluskey (1956) wrote another famous paper as part of his doctoral thesis while at Massachusetts Institute of Technology in the electrical engineering field. His paper was related to simplifying switching circuits and he proposed an algorithm which is still the standard for the exact boolean minimizations. The algorithm is called Quine–McCluskey to reflect the previous work of Quine (1952, 1955) to obtain a minimum solution. The

A. Duşa (✉)
Faculty of Sociology and Social Work, University of Bucharest, 1, Schitu Măgureanu Road, 050025 Bucharest, Romania
e-mail: adi@sas.unibuc.ro

algorithm is capable of simplifying a logical expression to obtain a minimal sum of products and is based on two main properties:

– any logical expression can be rewritten using the distributive law
– the reunion of A with a ($not\,A$) is always equal to 1 (or is always TRUE)

For example, the boolean expression $ABC + aBC$ can be rewritten as $(A + a)BC$ using the distributive law. Since the sum between the brackets $A + a$ is equal to 1, the result is BC. This is just a very simple example of a boolean minimization.

In social sciences, the most famous is the Qualitative Comparative Algorithm developed by Ragin (1987) who adapted the Quine–McCluskey procedure in sociology and political sciences. In fact, the foundation of Shannon's paper comes also from the social sciences, inspired from a philosophy class he took while a student, later demonstrating how to adapt boolean algebra to electric circuits.

Ragin was the first to see the utility of this technique for the representations on causes and effects; his work is heavily influenced by Mill's (1843) inductive cannons.

QCA is a via media (DeMeur and Rihoux 2002) between the qualitative world (centered on cases and case studies) and the quantitative world (centered on variables). Qualitative in nature, QCA eliminates individual subjective perspectives by employing a boolean algorithm which always yields the same solution. Results become replicable and the analysis gains trust, as understood by quantitative researchers.

Quantitative techniques use a large number of cases to draw inferences based primarily on correlations, but they focus solely on various degrees of *presence* of causes; using binary logic, QCA focuses on both presence and absence of causal conditions. Indeed, an outcome can also be triggered by the absence of a cause; for example we could imagine that anarchy or chaos can be caused (among other factors) by the absence of a strong legal system.

QCA's solutions are presented in minimum combinations of presence/absence of causal conditions that are necessary and/or sufficient for producing the outcome. Starting from a relatively small number of *observed* combinations, the analysis performs a minimization process to derive the minimum combinations.

For a small number of causal conditions the minimization process can be performed by hand, but each new condition entered in the analysis increases the number of possible combinations exponentially.

2 Current algorithms to produce a boolean minimization solution

2.1 fsQCA

The fuzzy-set QCA software developed by Ragin (2006) and his team at the University of Arizona is the more developed version of the QCA software designed to work in the DOS environment. This software performs both *crisp-set* analysis (with dichotomous cases: either in or out) and *fuzzy-set* analysis (that permits membership in a set with a continuous range between from 0 and 1) and is useful for sociology and political sciences research, although it is by no means limited to these areas.

The crisp-set procedure employed in this software is the standard Quine–McCluskey algorithm and is currently under a heavy revision to port the code from Modula2 (the original programming language) in the C language. The fuzzy-set procedure, although the most advanced in the field, is still continually updated and refreshed with Ragin's latest innovations.

Performing both crisp and fuzzy-set analysis in the same package has great advantages; for example Ragin has a technique to transform the data from fuzzy sets spaces to crisp truth tables and subsequently perform a boolean minimization.

2.2 Tosmana

Tosmana is another software designed in the political sciences field by Cronqvist (2006) at the University of Marburg. The main contribution in this package is the introduction of the Multi-Value logic in the minimization process (a technique called *MVQCA*). Unlike fuzzy-sets that present a continuum from 0 to 1, causal conditions in Tosmana present multiple discrete categories.

The minimization logic of the algorithm is similar to the boolean logic, returning not only minimum presence/absence combinations, but also minimum combination of levels in causal conditions responsible for producing the outcome.

The algorithm behind this software is called the Graph Based Engine and has a very fast implementation in the compiled .Net code. This engine is a departure from the exact method, stressing its power on speed even with a high number of causal conditions; it is debatable however how far the results are from the standard exact procedure, given that any graph based algorithm is only an approximation of an exact solution. For a small number of causal conditions the engine returnes the same solutions, but there are sometimes significant differences when performing QCA with more than 10 causal conditions.

2.3 Espresso

This software is the work of the engineers from UC Berkeley and is in many ways an extention of the classic boolean minimization problem. It uses the so-called PLA (programmable logic arrays) minimization and it can handle multiple values in the input file. This technique can deal with up to 64 variables in the input, using a heuristic method to arrive at a minimum solution. The only impediment is that it returnes only the first minimum solution found, whereas multiple alternative solutions are found in the exact boolean procedure. While perfect for electrical engineering (obtain a result with the lowest possible cost, as low power consumption is a primary concern) it is not suited for social science research that needs all possible minimum solutions.

Unfortunately, the development of this software is currently stopped due to lack of funding.

2.4 BOOM-II (from BOOlean Minimization)

Fišer and Kubátová (2006) from the Czech Technical University continued the work on Espresso but under a different philosophy. Unlike all the other software that produce implicants bottom-up, this method applies a top-bottom strategy: while the classic Quine–McCluskey algorithm gradually reduces the number of literals from an initial expression, BOOM starts from the simplest possible expressions and it gradually adds literals until a minimum solution is found.

The second version of BOOM (called BOOM-II) combines the initial algorithm with another one called FC-Min in order to solve minimization problems with an extremely large number of input conditions. This approach is very appealing but as far as I understand the algorithm returnes only the first minimal solution instead of all possible ones, and is not guaranteed to find a minimum solution for very large number of input variables. While finding only one of the shortest paths to the outcome is perfectly logic for electric circuits, in the

social sciences the task is to find all paths that lead to producing the phenomenon of interest, for which purpose BOOM is not well suited.

This paper proposes a derivation of the classic Quine–McCluskey algorithm, based on simple steps and visibly faster than all the other current exact methods. Although it does not completely remove an algorithmic, iterative process of obtaining a minimized solution, the method presented in this paper does return all possible minimum solutions and is based on simple mathematical operations which not only increase the speed but also require less computer resources.

3 From base 2 to base 3

In my opinion the present "exact" boolean minimization algorithms (including QCA) pays a tribute and continues the tradition of the original Shannon and McCluskey papers. Coming from the engineering field, both are centered on values 0 and 1 (as numerical representations of the closed and opened gates). McCluskey's example (1956, p. 1420) on canonical expansion uses a combination of 0s, 1s and dashes (−):

$$\begin{bmatrix} 0\,0\,1\,1 \\ 3 \end{bmatrix} + \begin{bmatrix} 0\,1\,1\,1 \\ 7 \end{bmatrix} = \begin{bmatrix} 0\,-\,1\,1 \\ 3,7 \end{bmatrix}$$

Other algorithms present an "x" sign for a minimized literal, but generally this pattern is preserved. Interesting enough, McCluskey did use both binary and decimal notations to represent combinations of opened and closed circuits, a crucial aspect that was too easily overlooked.

In the social sciences, the QCA algorithm developed by Ragin (1987) treats 0 as the absence and 1 as the presence of a condition. While these values are perfectly valid for describing social phenomena and case studies, an algorithm restricted to only these two values is bound to be slow for a large number of conditions.

The first general explanation relates the speed to the number of possible combinations in the truth table, which increases exponentially to the powers of 2; for two causal conditions there are 2^2 combinations, for 3 causal conditions there are exactly 2^3 combinations and generally for k causal conditions there are 2^k possible combinations. For $k = 3$, the truth table (here represented as a matrix) is presented in Table 1.

In the social sciences, it is customary to represent causal conditions as letters (literals), replacing 0 (the absence of the causal condition) with lower letters and 1 (the presence of the causal condition) with capital letters. If three causal conditions were coded A, B and C, the binary combination '1 0 1' would be represented by 'A b C'.

Table 1 The truth table for three causal conditions

Line	A	B	C	Base 10
1	0	0	0	0
2	0	0	1	1
3	0	1	0	2
4	0	1	1	3
5	1	0	0	4
6	1	0	1	5
7	1	1	0	6
8	1	1	1	7

In his later shift to fuzzy sets, Ragin (2000, p. 127) correctly observes there is a finite number of possible prime implicants, which are included in what he calls "logically possible groupings" and he shows there are $3^k - 1$ such groupings.

His groupings idea is close to the essence of the algorithm I propose, but once again it remained insuficiently exploited.

In fact, there are exactly 3^k possible combinations (groupings). The last one (actually the first one in the matrix) is not even considered anywhere in the literature: it is the case when all causal conditions have been minimized, therefore no condition is responsible for the appearance of the outcome. Even though this case never really occurs, it is still logically possible; this algorithm uses the complete 3^k matrix (Ragin's groupings plus the remaining combination) for ease of computation. An exact solution to a boolean minimization problem is always found in the 3^k space.

To completely benefit from this idea, the algorithm presented in this paper uses a slightly (but crucially) different notation:

- a minimized literal is coded with 0
- the absence of a causal condition is coded with 1 (instead of 0)
- the presence of a causal condition is code with 2 (instead of 1)

It is still a boolean minimization problem, even though it now uses three numbers: 0, 1 and 2. This algorithm shifts from the base 2 matrix (the 2^k combinations in the truth table) to a base 3 matrix (the 3^k space); for three causal conditions, the matrix is presented in Table 2.

If, for example, after a boolean minimization process the essential prime implicants remained are found on the lines 2 and 7 (binary combinations '0 0 1' and '0 2 0'), transforming

Table 2 The 3^k space for three causal conditions

Line	A	B	C	Base 10	Truth table		
1	0	0	0	0			
2	0	0	1	1			
3	0	0	2	2			
4	0	1	0	3			
5	0	1	1	4			
6	0	1	2	5			
7	0	2	0	6			
8	0	2	1	7			
9	0	2	2	8			
10	1	0	0	9			
11	1	0	1	10			
12	1	0	2	11			
13	1	1	0	12			
14	1	1	1	13	0	0	0
15	1	1	2	14	0	0	1
16	1	2	0	15			
17	1	2	1	16	0	1	0
18	1	2	2	17	0	1	1
19	2	0	0	18			
20	2	0	1	19			
21	2	0	2	20			
22	2	1	0	21			
23	2	1	1	22	1	0	0
24	2	1	2	23	1	0	1
25	2	2	0	24			
26	2	2	1	25	1	1	0
27	2	2	2	26	1	1	1

these lines back in the base 2 form results in '– – 0' and '– 1 –' and the solution is 'c + B': the outcome is produced either by the absence of the third condition or by the presence of the second condition.

4 Possible methods of comparing cases

A possible, classic algorithm is an exhaustive one that checks everything: from all the literals in any combination of causal conditions to comparing every possible pair of combinations. This is the most inneficient and most resource consuming type of algorithm.

For $k = 10$, the total number of combinations is $2^{10} = 1,024$. It may not seem a very large number, but no minimization is possible unless two combinations differ by one and only one literal. In order to obtain an exact solution, a clasic computer algorithm would check all possible pairs of combinations, first to determine which combinations can be minimized and second to actually perform the minimization process (to produce the prime implicants).

This process is repeated until no further minimization is possible. The number of all possible paired comparisons is potentially huge; the minimization process resembles a bubble which quickly inflates in the first few iterations and deflates towards the last ones. For a large number of causal conditions, in its moment of maximum inflation there are millions of prime implicants; for only 10,000 prime implicants, the number of all possible paired comparisons is about 50 m (or about 25 m unique pairs, given that it is unnecessary to compare the same pair twice): $C^2_{10,000} \simeq 50,000,000$.

For millions of prime implicants this number becomes so large that no ordinary computer would ever produce a solution in a reasonable time.

Another possible approach is to use a cluster analysis for the entire input matrix at once. The clustering algorithms compare every pair of two lines in order to show how similar (or dissimilar) the pairs are. One such algorithm has a so called Manhattan distance which counts the number of differences in a pair of lines. If this distance is equal to 1 it means that pair is minimizable. Even though this technique is far superior in terms of speed to the previous method (it can return the exact pairs of line numbers that can be minimized), it still doesn't prevent iterating through every pair in order to extract the prime implicants.

The matrix to search is dramatically reduced but for a large number of conditions it still quickly builds up to such an extent that an algorithm of this kind becomes extremely slow.

Finally, the method used in the current algorithm uses the following ideas:

- any combination of causal conditions is a line number in the 3^k matrix
- a prime implicant produced by minimizing two other combinations (groupings) is itself a line number in the 3^k matrix

To free the algorithm from the burden of checking an entire matrix, a better idea is to find a mathematical function that takes two line numbers (the groupings) and produce another line number (the prime implicant). Instead of dealing with a large matrix, it could work with only a vector of line numbers; this is not only extremely fast but it also require dramatically fewer computer memory and resources. For example, a matrix of 10 causal conditions requires about 10 times more memory than a simple vector of line numbers.

This algorithm does not produce the entire truth table, but only selects the corresponding line numbers of the combinations included in the analysis. The problem is the decimal representation of the truth table (a 2^k, base 2 matrix) is not the same with the corresponding combinations in the 3^k, base 3 matrix. The italicized rows in Table 2 are the corresponding binary combinations in the truth table.

It is possible to transform the 2^k matrix into its base 3 equivalent (by simply adding 1) and to transform the resulted combinations in their base 10 representation, but this is inefficient because it uses valuable computer memory even to create the complete 2^k matrix.

Instead, it is possible to directly create the relevant line numbers using a few simple mathematical tricks.

5 Finding the relevant line numbers in the 3^k matrix

For binary data, this algorithm uses a function based on few empirical observations (for multi-value data, the formulae are different):

– the starting line number is always in the middle of the 3^k matrix, at the value:

$$\frac{3^k - 1}{2}$$

– the relevant lines are always grouped in blocks of two consecutive numbers
– all blocks are found in a series of three symmetric regions: the upper region and the lower region always have the same pattern of lines, and the middle region is a gap which always has possible prime implicants.

The last observation is more evident for a matrix with four causal conditions; the second relevant half is presented in Table 3.

Given the fact that line numbers are always grouped in blocks of two, the function only computes half of the numbers (the others are easily obtained by adding 1). The starting block is already known, so the only remaining task is to determine the gaps between the series of three symmetric regions.

The formula to compute the gaps (in terms of number of lines) is: $3x + 1$ where x is the last gap found.

The first x value to start with is 0 (because there is no gap between the first two lines), therefore the first gap is $3 \cdot 0 + 1 = 1$

The second gap is $3 \cdot 1 + 1 = 4$; because there is a series of three symmetric regions, the vector of gaps has the values: 1, 4, 1.

The next gap is $3 \cdot 4 + 1 = 13$; again, the function models the three symmetric regions and the vector of gaps has the values: 1, 4, 1, 13, 1, 4, 1.

This procedure is repeated for a number of $k - 1$ times, where k is the number of causal conditions.

Another equivalent formula to compute the gaps is:

$$\frac{3^n - 3}{6} \text{ where } n = 2, \ldots, k.$$

For $n = 2$, the first gap is:

$$\frac{3^2 - 3}{6} = 1$$

For $n = 3$, the second gap is:

$$\frac{3^3 - 3}{6} = 4$$

Table 3 The second half of the 3^k matrix with four causal conditions

Line	A	B	C	D	Base 10	Truth table			
41	1	1	1	1	40	0	0	0	0
42	1	1	1	2	41	0	0	0	1
43	1	1	2	0	42				
44	1	1	2	1	43	0	0	1	0
45	1	1	2	2	44	0	0	1	1
46	1	2	0	0	45				
47	1	2	0	1	46				
48	1	2	0	2	47				
49	1	2	1	0	48				
50	1	2	1	1	49	0	1	0	0
51	1	2	1	2	50	0	1	0	1
52	1	2	2	0	51				
53	1	2	2	1	52	0	1	1	0
54	1	2	2	2	53	0	1	1	1
55	2	0	0	0	54				
56	2	0	0	1	55				
57	2	0	0	2	56				
58	2	0	1	0	57				
59	2	0	1	1	58				
60	2	0	1	2	59				
61	2	0	2	0	60				
62	2	0	2	1	61				
63	2	0	2	2	62				
64	2	1	0	0	63				
65	2	1	0	1	64				
66	2	1	0	2	65				
67	2	1	1	0	66				
68	2	1	1	1	67	1	0	0	0
69	2	1	1	2	68	1	0	0	1
70	2	1	2	0	69				
71	2	1	2	1	70	1	0	1	0
72	2	1	2	2	71	1	0	1	1
73	2	2	0	0	72				
74	2	2	0	1	73				
75	2	2	0	2	74				
76	2	2	1	0	75				
77	2	2	1	1	76	1	1	0	0
78	2	2	1	2	77	1	1	0	1
79	2	2	2	0	78				
80	2	2	2	1	79	1	1	1	0
81	2	2	2	2	80	1	1	1	1

The third gap is:

$$\frac{3^4 - 3}{6} = 13, \text{ and so on.}$$

The rest of the function computes a first vector of relevant line numbers using the formula:

$$x + y + 2$$

where x is the last number computed (starting with $\frac{3^k+1}{2}$), and
y is taken from the vector of gaps.

In Table 3 the starting line number is:

$$\frac{3^4+1}{2}=41$$

The next line number computed is:

$$41+1+2=44$$

The next one is:

$$44+4+2=50$$

and so on.

Finally, adding 1 to each line number previously computed forms the entire vector of relevant line numbers in the base 3 matrix.

6 Minimizing two lines: the $2x-y$ rule

The mathematical formula to minimize two line numbers (groupings, combinations of causal conditions or even prime implicants) is:

$$2x-y$$

where x is the smaller line number and y is the larger line number.

In Table 2, it can be observed that line 14: the equivalent of the '0 0 0' combination in the truth table ('a b c') and line 15: the equivalent of the '0 0 1' combination in the truth table ('a b C') can be minimized in the line 13: the equivalent of the '0 0 –' prime implicant ('a b –').

Another example of line numbers that can be minimized is the pair consisting from line 14: the equivalent of the '0 0 0' combination in the truth table and line 23: the equivalent of the '1 0 0' combination in the truth table.

Applying the $2x-y$ rule gives: $2\cdot 14-23=5$.

Line 5 is the equivalent of the '– 0 0' prime implicant.

Similarly, two prime implicants could further be minimized. For example, line 5 (which is '– 0 0 ', or '– b c') can be minimized with line 6 (which is '– 0 1', or '– b C') to produce the essential prime implicant '– 0 –' or simply the 'b' literal.

Not all pairs of line numbers can be minimized. Just as the extended combinations of causal conditions, the only pairs of combinations that can be minimized are those that differ by one literal (and one literal only). For example the pair of line numbers 14 ('0 0 0') and 24 ('1 0 1') cannot be minimized because they differ by two literals.

The next question to answer is: how does the algorithm decide which line numbers are minimizable and which are not?

7 Finding the pairs of numbers that can be minimized: the forward rule

Let us take the simple example of the truth table with three causal conditions presented in Table 1. In Fuzzy Sets Social Science, Ragin (2000) defines fuzzy sets as a multidimensional vector space with 2^k corners (where k is the number of causal conditions).

The answer to the last question uses this sharp observation: for three causal conditions, the vector space with $2^3 = 8$ corners is a cube. From any corner of a cube there are exactly three connections to other corners, which means one line in the truth table can be minimized with exactly three other lines in the same truth table.

In Table 1, the truth table is a 2^k matrix where line 1 is the combination '0 0 0'. The other combinations in the truth table that can be used to minimizing line 1 are: line 5 ('1 0 0'); line 3 ('0 1 0') and line 2 ('0 0 1'). Notice the corresponding numbers in the base 10 equivalent and their equivalents in the powers of 2:

Combination	Base 10	Equivalent
1 0 0	4	2^2
0 1 0	2	2^1
0 0 1	1	2^0

The minimizable pairs of numbers are found in the powers of 2, in the following order: $2^2\ 2^1\ 2^0$. For k causal conditions, the powers of 2 start with 2^{k-1} and end with 2^0.

Some lines are different with other lines both forward and backward from their position in the truth table. For example line 3 ('0 1 0') can be minimized with line 7 ('1 1 0'), line 1 ('0 0 0') and line 4 ('0 1 1').

Lines 7 and 4 are forward whereas line 1 is backward: 0 means 'forward' and 1 means 'backward'.

The combination '0 1 0' is thus 'forward backward forward', so that:

$3 + 2^2 = 7$

$3 - 2^1 = 1$

$3 + 2^0 = 4$

When minimizing a vector of line numbers, there is no need to duplicate an already minimized pair. Suppose we started with lines 1, 3, 4 and 7.

Line 1 can be minimized with line 3, so the pair (1, 3) is checked.

Line 3 can be minimized with all the other, but the pair (3, 1) is the same with the pair (1, 3): they both produce the same prime implicant. There is no need to search backwards: if there was a pair line to minimize with, the algorithm would produce the prime implicant when checking that line. This is a simplification that not only gets rid of redundant pairs but it substantially reduces the computer circles needed to finish the algorithm.

The very same principle holds in the base 3 matrix; Table 3 is a matrix where the pairs of numbers can be found in the powers of 3, in the order: $3^3\ 3^2\ 3^1 3^0$

The only difference from the base 2 matrix is related to defining the numbers:

- 0 means a minimized literal,
- 1 means 'forward' and
- 2 means 'backward'.

For example, to find the pairs with line 53 ('1 2 2 1'):

$53 + 3^3 = 80$

$53 - 3^2 = 44$

$53 - 3^1 = 50$

$53 + 3^0 = 54$

Following the forward rule, the only pairs of interest are lines 80 and 54.

In QCA every piece of memory counts, therefore the difficult part is to generate these numbers, without generating the whole 3^k matrix.

Table 4 depicts a 3^3 matrix stripped down to the forward bits of interest. The structure of the forward bits has a symmetric pattern: there is a sequence of $3^0 = 1$ bit every three lines

Table 4 The *forward* bits in a 3^k matrix with three causal conditions

Line	A	B	C
1			
2			1
3			
4		1	
5		1	1
6		1	
7			
8			1
9			
10	1		
11	1		1
12	1		
13	1	1	
14	1	1	1
15	1	1	
16	1		
17	1		1
18	1		
19			
20			1
21			
22		1	
23		1	1
24		1	
25			
26			1
27			

in the last column, then a sequence of $3^1 = 3$ bits in the middle column, then a sequence of $3^2 = 9$ bits in the first column. The same expansion holds for any number of columns.

Line 2 can be forward minimized with line 3 only $(2 + 3^0)$, whereas line 4 can be forward minimized with line 7 only $(4 + 3^1)$.

Generating the differences (defined as the line numbers that are forward minimizable) can be done using the powers of 3:

- generate $3^0 = 1$ sequence of $3^2 = 9$ consecutive numbers starting with line 10 $(1 + 3^2)$ and ending with line 18 $(3^3 - 3^2)$
- generate $3^1 = 3$ sequences of $3^1 = 3$ consecutive numbers, starting with line 4 $(1 + 3^1)$ and ending with line 24 $(3^3 - 3^1)$
- generate $3^2 = 9$ sequences of $3^0 = 1$ consecutive numbers, starting with line 2 $(1 + 3^0)$ and ending with line 26 $(3^3 - 3^0)$

8 Memory versus speed. The differences matrix

These sequences are used by the algorithm in order to make the necessary minimizations. Usually, there are several iterations of minimizations until the algorithm finds the prime implicants that cannot be further minimized. Generating the sequences many times is not a problem for a small number of causal conditions, but as they grow larger the generation of the sequences themselves takes up important computer resources, therefore the algorithm grows slower.

Table 5 The differences matrix for three causal conditions

$+3^2$	$+3^1$	$+3^0$
10	4	2
11	5	5
12	6	8
13	13	11
14	14	14
15	15	17
16	22	20
17	23	23
18	24	26

A better solution is to generate the differences only once at the beginning of the algorithm and store the numbers for use in all iterations. There are 3^{k-1} forward bits in each column of the 3^k matrix; in Table 4 with 3 causal conditions there are 9 forward bits on every column. The best storing object with sequences of the same length is a matrix with 3^{k-1} rows.

Table 5 is a matrix holding all possible forward differences for three causal conditions. For example, if line 10 is found in the first column, it will be minimized with line $10 + 3^2 = 19$ using the $2x - y$ rule.

This approach uses more memory: for 15 causal conditions, the matrix has $3^{14} = 4{,}782{,}969$ rows by 15 columns (a total of 71,744,535 values and the memory usage grows exponentially), but it is faster than generating the differences on the fly, on every iteration.

Fortunately, the larger the number of causal conditions included in the analysis, the larger is the number of the cases needed (otherwise the number of potential solutions is extremely large). As QCA researchers usually study a limited number of cases, it is extremely rare for a QCA researcher to work with more than a dozen of causal conditions simultaneously.

In the memory versus speed dilemma, this algorithm opts for speed even if it means more memory consumption. This is the only part of the algorithm which is intensive memory consuming.

9 A simple example for three causal conditions

Let us suppose we have a truth table with the following configuration:

Line	A	B	C	Outcome
1	0	0	0	1
2	0	0	1	1
3	0	1	0	1
4	0	1	1	1
5	1	0	0	1
6	1	0	1	1
7	1	1	0	0
8	1	1	1	0

The last two lines are excluded from the analysis because the outcome is absent. The first step of the algorithm is to select the first six line numbers from the base 3 matrix, which are: 14, 15, 17, 18, 23, 24.

In the first iteration, the algorithm performs a vector matching with all three columns in the differences matrix from Table 5:

- In the first column the matches are: 14, 15, 17, 18
  Adding $3^2 = 9$ to find their forward pairs: 23, 24, 26, 27
  Values 26 and 27 are not in the starting vector, therefore the only pairs retained are (14, 23) and (15, 24)
- In the second column the matches are: 14, 15, 23, 24
  Adding $3^1 = 1$ to find their forward pairs: 17, 18, 26, 27
  Values 26 and 27 are not in the initial vector, therefore the only pairs retained for minimization are (14, 17) and (14, 18)
- In the third column the matches are: 14, 17, 23
  Adding $3^0 = 1$ to find their forward pairs: 15, 18, 24
  All of them are in the initial vector.

There are seven comparable pairs in total; applying the $2x - y$ rule yields the first set of seven unique prime implicants: 5, 6, 11, 12, 13, 16, 22.

In the second iteration, the same procedure is repeated:

- In the first column the matches are: 11, 12, 13, 16
  Adding $3^2 = 9$ to find their forward pairs: 20, 21, 22, 23
  Values 20, 21 and 23 are not in the prime implicants vector, therefore the only pair retained is (13, 22)
- In the second column the matches are: 5, 6, 13, 22
  Adding $3^1 = 1$ to find their forward pairs: 8, 9, 16, 25
  Values 8, 9, and 25 are not in the prime implicants vector, therefore the only pair retained is (13, 16)
- In the third column the matches are: 5, 11
  Adding $3^0 = 1$ to find their forward pairs: 6, 12
  All of them are in the prime implicants vector.

There are 4 comparable pairs in total; applying the $2x - y$ rule yields the second set of prime implicants: 4, 10, 4, 10.
Two of them are duplicated, so there are two unique prime implicants left: 4, 10.

In iteration 3, no further minimizations are possible. From Table 2, the combinations of causal conditions in lines 4 and 10 (in base 3) are:

Line	A	B	C
4	0	1	0
10	1	0	0

Remember that the 3^k matrix is not created by the algorithm; Table 2 is only presented in this paper for demonstration purposes. Instead, a function to transform the line numbers from base 10 to base 3 returns the same configuration.

In base 2 equivalent, the combinations are:

A	B	C
–	0	–
0	–	–

Expressed as literals, the final prime implicants are: ‘b’ for line 4 and ‘a’ for line 10.

Table 6 The adapted Quine's prime-implicant chart example

	ABCD	ABCd	ABcD	ABcd	AbcD	Abcd	aBCD	aBCd	abCD	abCd	abcd
AB	x	x	x	x	–	–	–	–	–	–	–
BC	x	x	–	–	–	–	x	x	–	–	–
Ac	–	–	x	x	x	x	–	–	–	–	–
aC	–	–	–	–	–	–	x	x	x	x	–
abd	–	–	–	–	–	–	–	–	–	x	x
bcd	–	–	–	–	–	x	–	–	–	–	x

10 Solving the prime-implicant chart

The next step in the QCA procedure is to solve the prime implicant chart to further reduce the number of irrelevant prime implicants. A prime-implicant chart is a matrix that has the prime implicants on the rows and the initial combinations of presence-absence of causal conditions on the columns.

Given this simple example, no further reductions are possible and the final solution is: 'a + b': the outcome is produced either by the absence of 'a' or by the absence of 'b'.

A more complicated example is presented in Table 6, adapted from (Quine, 1952, p. 358).

Many procedures to solve the prime-implicant chart exist. The final minimum solution depends heavily on the procedure employed; for the results to be replicable, the routine should be properly described. Sometimes the set of rules can be altered and the final solution is more or less different, but it depends on the researchers objective: to obtain the minimum (exhaustive) combination of prime implicants or to obtain less minimal but more substantively interesting solutions.

In his 1952 paper, Quine presents a set of rules to extract the essential prime implicants (or what he calls "the core of the formula") and to simplify the chart until every possible solution is extracted. His first rule is based on the observation that any column that present no more than one cross on the rows is a part of the core, and the second rule applies only if the first one is applied. His third rule resembles on what is called "the elimination of dominating columns": if one column present crosses in all rows where another column present more crosses, than the second column dominates the first one and should be eliminated.

What Quine does not cover in that paper is another procedure, called "the elimination of dominated row": if one row present crosses in all columns where another row present more crosses, than the first row is dominated by the second and should be eliminated.

Indeed, a dominated row is shorter than a dominating one, and contributes less to finding a minimum number of prime implicants to cover all columns.

Later, Quine (1955) presents another method to derive a minimum combination of prime implicants without constructing the chart, but modern computers are more than capable of solving this "cumbersome" aspect at the time.

Although Quine's procedure still remains the standard in solving prime-implicant charts, this algorithm employes a different method. First, it eliminates the dominated rows, then it applies a linear programming function which returnes the minimum number of rows (prime implicants) needed to cover all columns (say X); the rest is trivial, checking all possible combinations of X rows and returning only those combinations that present a complete coverage.

The QCA researchers usually study a limited number of case studies, therefore the number of columns is relatively small. Eliminating the dominating columns does not significantly increase the speed of the algorithm. The situation is different with the rows of a prime-implicants chart: for more than 10 causal conditions, it is possible to obtain hundreds of minimized

prime implicants. Checking all possible combinations of X rows is significantly faster if the number of rows is reduced; before applying the linear programming function, the algorithm deletes all dominated rows. This exhaustive procedure returns all possible minimum solutions, just like Quine's original method.

11 Conclusions

Although very powerful algorithms have been developed in the engineering field, some of them heuristic in nature, they do not present a complete solution (i.e. all possible minimum combinations of causal conditions) therefore they are not ideally suited for social research.

The current algorithms that do produce a complete solution are either designed in the classic Quine–McCluskey procedure which is memory hungry and extremely slow, or use some approximations that are not guaranteed to produce the same solutions as the classic exact method.

In this paper I have demonstrated a different method, a variation of the classic exact algorithm that is not only fast but it requires significantly less computer resources.

The algorithm is already implemented and available as the QCA package in R (R Development Core Team 2007). It also exists a graphical user interface in the companion package QCAGUI, for which there is a user manual (Duşa 2007).

Acknowledgements The author thanks Paul Mihai for the fruitful discussions from which many of the ideas in this article have emerged, and for his strong belief in the existence of a mathematical solution. Special thanks to Roxana Purnichescu, who's mathematical skills and encouragement have been a helpful source of inspiration and to Gheorghe Ştefănescu, Professor of Computer Science at the University of Bucharest who's help arrived at exactly the right time.

References

Cronqvist, L.: Tool for small-N analysis [Version 1.255], Marburg. Retrieved from http://www.tosmana.net in January 2007

De Meur, G., Rihoux, B.: L'analyse Quali-Quantitative Comparee. Approche, techniques et applications en Science Humaines. Academia-Bruylant, Louvain-La-Neuve (2002)

Duşa, A.: User manual for the QCA(GUI) package in R. J. Business Res. **60**(5), 576–586 (2007)

Fišer, P., Kubátová, H.: Flexible two-level Boolean minimizer BOOM-II and its applications. IEEE Comput. Soc. Washington, DC, USA (2006)

McCluskey, E.J.: Minimization of Boolean functions. Bell Syst. Technical J. **5**, 1417–1444 (1956)

Mill, J.S.: A System of Logic. Longman, Green, and Co., London (1843)

Quine, W.O.: The problem of simplifying truth functions. Bell Syst. Technical J. **5**, 521–531 (1952)

Quine, W.O.: A way to simplify truth functions. Bell Syst. Technical J. **62**, 627–631 (1955)

R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2007)

Ragin, C.C.: The Comparative Method. Moving Beyond Qualitative and Quantitative Strategies. University Of California Press, Berkeley (1987)

Ragin, C.C.: Fuzzy Set Social Science. The University of Chicago Press (2000)

Ragin, C.C.: User's Guide to Fuzzy-Set/Qualitative Comparative Analysis 2.0. Department of Sociology, University of Arizona, Tucson, Arizona (2006)

Shannon, C.E.: A Symbolic Analysis of Relay and Switching Sircuits. Massachusetts Institute of Technology, Department of Electrical Engineering. Retrieved from http://theses.mit.edu in October 2005