stcm: An R Package for Applying and Evaluating Set-Theoretic Comparative Methods

Chris Krogslund

Department of Political Science
University of California, Berkeley
ckrogslund@berkeley.edu
http://ckro.gs

Draft date: March 19, 2015

Abstract

Social scientists have become increasingly interested in the development and application of Set-Theoretic Comparative Methods for social inquiry. These methods — notably Qualitative Comparative Analysis — are distinguished from other methods by their set-based variable codings, use of boolean reduction algorithms, and common application to small datasets. Recent methodological work has shown, however, that their results can be especially sensitive to small changes in certain calibration and reduction parameters, as well as the presence measurement error or model-specification error. The **stcm** package introduces a number of tools that are designed to help users of these methods assess the sensitivity of their results to a variety of perturbations. The package also introduces a set of functions implementing a related class of methods that relies centrally on classification and regression tree algorithms.

Introduction

Over the past 25 years, qualitative and multi-method research in the social sciences has made increasingly heavy use of Set-Theoretic Comparative Methods (STCMs), most prominently Qualitative Comparative Analysis (Mahoney and Goertz, 2006; Goertz and Mahoney, 2012; Marx et al., 2014). Originally developed by Charles Ragin in the late 1980s, QCA was intended to serve as a counterweight to some of the most popular methods used for social inquiry – notably linear regression (Ragin, 1987). QCA is built upon an explicitly set-theoretic foundation. All quantities of interest are scores that measure the consistency of a given case with membership in a particular set. Moreover, the method uses a boolean reduction algorithm to identify combinations of set memberships – often referred to as configurations – that appear to be either necessary or sufficient to produce a given outcome. Over time, a number of QCA variants have been developed that accommodate different types of set membership scores (Ragin, 2000; Rihoux and Ragin, 2009) and utilize novel algorithms for boolean reduction (Thiem and Duşa, 2013a).

While interest in this class of methods has grown considerably, methodologists have recently pointed to a number of potential problems that could jeopardize the quality of its inferences (Skaaning, 2011; Hug, 2013; Thiem, 2013; Krogslund et al., 2014; Krogslund and Michel, 2014a; Lucas and Szatrowski, 2014; Seawright, 2014). Many of these problems are specific to a particular method or a specific variant of QCA, but one can tease out three major themes amongst these issues, namely a) the sensitivity of results to minor changes in boolean reduction parameters, b) the sensitivity of results to small amounts of measurement and model specification error, and c) the validity of results with small- and medium-N sample sizes, or roughly fewer than 50 cases.

Despite the increasingly wide recognition of these problems (Collier, 2014), there are currently few tools available for empirical researchers using STCMs to assess the robustness of their results with respect to these potential issues. The **stcm** package looks to remedy this situation by providing a collection of tools for empirical researchers to identify potential result validity and reliability problems arising from parameter specifications, measurement error, and model specification error. Additionally, as a first step in attending to validity concerns over small- and medium-N results, **stcm** provides functionality implementing several new methods for set-theoretic inference using classification and regression tree algorithms, as introduced by Krogslund and Michel (2014b).

In the following sections, a number of inferential difficulties commonly encountered when using STCMs will be illustrated. For each, relevant functions from the **stcm** package will be introduced that help empirical researchers in diagnosing the issue and, if possible, addressing it. Specifically, functions for identifying and confronting sensitivity to calibration and reduction parameter choices, measurement error, and model specification error are illustrated. An introduction to several functions for implementing decision-tree based STCMs is also provided.

Parameter Specification Sensitivity

Perhaps the most significant objection to the use of many STCMs is their somewhat heavy use of user-specified parameter values. For instance, QCA requires the user to pass values for at least three parameters, including 1) a minimum score approximating how consistent a particular configuration of set memberships must be with the outcome for it to be considered a solution, 2) the minimum number of cases that must display a certain configuration of set memberships in order for that configuration to be included as a solution, and 3) a logical value indicating whether simplifying assumptions should be made about set membership configurations that are unobserved in the data. Altering these values – the minimum sufficiency inclusion score, minimum frequency threshold, and solution type, respectively – can cause QCA to produce wildly different solutions (Krogslund et al., 2014).

Consider, for instance, an example dataset containing set membership scores for 20 Middle Eastern and North African countries in a number of conceptual sets related to the success of protest movements during the 2011 Arab Spring uprisings.[1] Each set membership score is fuzzy, meaning it can range from 0 (full non-membership in a set) to 1 (full membership in a set). `success` is the outcome of interest, namely whether or not a protest movement successfully overthrew a governing a regime. Several potential explanatory set memberships are also included in the dataset, specifically:

- `gdppc`: high average wealth
- `gini`: high income inequality
- `unemp`: high unemployment
- `urban`: high urbanization
- `youth`: low population age
- `mobile`: widespread digital connectivity
- `internet`: high sophistication in digital censoring
- `fuel`: highly fuel-dependent economy
- `pol`: high regime fragility

```
# Load dataset
data(hh)

# Inspect dataset
print(hh)
      Country gdppc gini unemp urban youth mobile internet fuel  pol success
1     Bahrain  0.84 0.42  0.58  0.89  0.26   0.68     0.89 0.58 0.11    0.00
2    Djibouti  0.16 0.74  1.00  0.63  0.74   0.05     0.16 0.01 0.83    0.20
3        Iraq  0.05 0.11  0.74  0.42  0.89   0.42     0.79 0.37 0.94    0.30
4  Mauritania  0.11 0.68  0.84  0.21  0.84   0.32     0.05 0.26 0.67    0.40
5         UAE  0.95 0.11  0.11  0.79  0.05   0.79     1.00 0.53 0.11    0.40
6       Qatar  1.00 0.79  0.01  0.95  0.01   0.74     0.95 0.63 0.01    0.50
7     Somalia  0.01 0.01  0.95  0.11  1.00   0.01     0.01 0.16 0.50    0.50
8     Algeria  0.58 0.37  0.42  0.47  0.32   0.47     0.32 1.00 0.83    0.55
9       Sudan  0.26 1.00  0.79  0.16  0.79   0.11     0.21 0.74 0.67    0.55
10     Jordan  0.53 0.58  0.47  0.01  0.63   0.63     0.47 0.05 0.56    0.70
11     Kuwait  0.89 0.01  0.05  1.00  0.16   0.84     0.74 0.89 0.28    0.60
12    Lebanon  0.63 0.95  0.32  0.84  0.21   0.26     0.58 0.11 1.00    0.60
13    Morocco  0.42 0.79  0.37  0.37  0.37   0.53     0.68 0.21 0.44    0.70
```

[1]This data is taken from Hussain and Howard (2013).

```
14         Oman  0.79 0.21  0.58  0.58  0.47   0.89     0.84 0.68 0.11     0.60
15        Saudi  0.74 0.21  0.16  0.74  0.53   1.00     0.47 0.79 0.01     0.60
16        Syria  0.37 0.42  0.21  0.32  0.68   0.21     0.42 0.47 0.28     0.55
17        Libya  0.68 0.42  0.84  0.68  0.42   0.95     0.11 0.95 0.28     0.80
18        Egypt  0.32 0.21  0.26  0.26  0.58   0.37     0.37 0.42 0.56     1.00
19      Tunisia  0.47 0.79  0.53  0.53  0.11   0.58     0.63 0.32 0.50     1.00
20        Yemen  0.21 0.58  0.58  0.05  0.95   0.16     0.26 0.84 0.67     1.00
```

To identify combinations of these explanatory set memberships that may be sufficient to yield a successful protest movement, we can use the `eqmcc()` function in the *QCA* package (Thiem and Duşa, 2013b). This function takes in a dataset of set membership scores (including crisp, fuzzy, and multi-value scores) and identifies sufficient conditions using the Enhanced Quine-McCluskey algorithm for boolean reduction. At first, we will run the function with its default parameter values. Of note, this means that the minimum sufficiency inclusion score (`incl.cut1`) and the minimum frequency threshold (`n.cut`) will both be set to 1.

```
# Remove case identifier
hh<-hh[,colnames(hh)!="Country"]

# Change capitalization of outcome variable name
# NOTE: this is a bug in eqmcc() current as of QCA 1.1-3
colnames(hh)<-toupper(colnames(hh))

# Run eqmcc() with default parameter values
eqmcc(data = hh, outcome = "SUCCESS")
Error: Nothing to explain. Please check the truth table.
```

Given these parameter values, `eqmcc()` has dichotimized the data in such a way that there is actually no variation in the outcome across all cases. Examining the truth table shows that, while many different configurations satisfy the minimum frequency threshold requirement (shown in column `n`), no case has a sufficiency inclusion score (shown in column `incl`) higher than the minimum required value.

```
# Examine truth table
truthTable(data = hh, outcome = "SUCCESS")

  OUT: outcome value
    n: number of cases in configuration
 incl: sufficiency inclusion score

     GDPPC GINI UNEMP URBAN YOUTH MOBILE INTERNET FUEL POL OUT n  incl
 17    0     0     0     0     1      0        0      0    0   0  1  0.953
 18    0     0     0     0     1      0        0      0    1   0  1  0.955
 86    0     0     1     0     1      0        1      0    1   0  1  0.843
141    0     1     0     0     0      1        1      0    0   0  1  0.959
210    0     1     1     0     1      0        0      0    1   0  1  0.830
212    0     1     1     0     1      0        0      1    1   0  2  0.927
242    0     1     1     1     1      0        0      0    1   0  1  0.786
260    1     0     0     0     0      0        0      1    1   0  1  0.949
303    1     0     0     1     0      1        1      1    0   0  2  0.837
315    1     0     0     1     1      1        0      1    0   0  1  0.953
363    1     0     1     1     0      1        0      1    0   0  1  0.953
367    1     0     1     1     0      1        1      1    0   0  2  0.791
410    1     1     0     0     1      1        0      0    1   0  1  0.952
422    1     1     0     1     0      0        1      0    1   0  1  0.957
431    1     1     0     1     0      1        1      1    0   0  1  0.822

It seems that all your outcome values have been coded to zero.
May we suggest lowering the inclusion score for the presence of the outcome?
The relevant argument is "incl.cut1", which now has a value of 1.
```

To remedy this problem, one can begin by altering the value for the sufficiency inclusion score, setting it now to 0.8. Unfortunately, the configurations found by QCA to be sufficient to produce the outcome are numerous. Increasing the minimum frequency threshold clarifies the result somewhat, as does increasing the minimum sufficiency inclusion to 0.9. The clarifying gains of the sufficiency inclusion score top out, however, when the parameter is set to 0.95.

```
# Lower minimum sufficiency inclusion score to 0.8
eqmcc(data = hh, outcome = "SUCCESS", incl.cut1 = 0.8)

M1: gdppc*gini*unemp*urban*YOUTH*mobile*internet*fuel +
    gdppc*GINI*UNEMP*urban*YOUTH*mobile*internet*POL +
    GDPPC*unemp*URBAN*youth*MOBILE*INTERNET*FUEL*pol +
    gdppc*gini*UNEMP*urban*YOUTH*mobile*INTERNET*fuel*POL +
    gdppc*GINI*unemp*urban*youth*MOBILE*INTERNET*fuel*pol +
    GDPPC*gini*unemp*urban*youth*mobile*internet*FUEL*POL +
    GDPPC*gini*unemp*URBAN*YOUTH*MOBILE*internet*FUEL*pol +
    GDPPC*gini*UNEMP*URBAN*youth*MOBILE*internet*FUEL*pol +
    GDPPC*GINI*unemp*urban*YOUTH*MOBILE*internet*fuel*POL +
    GDPPC*GINI*unemp*URBAN*youth*mobile*INTERNET*fuel*POL => SUCCESS

# Increase minimum frequency threshold to 2
eqmcc(data = hh, outcome = "SUCCESS", incl.cut1 = 0.8, n.cut=2)

M1: gdppc*GINI*UNEMP*urban*YOUTH*mobile*internet*FUEL*POL +
    GDPPC*gini*unemp*URBAN*youth*MOBILE*INTERNET*FUEL*pol => SUCCESS

# Increase minimum frequency threshold to 3
eqmcc(data = hh, outcome = "SUCCESS", incl.cut1 = 0.8, n.cut=3)
Error: replacement has 0 items, need 180

# Increase minimum sufficiency inclusion score to 0.9
eqmcc(data = hh, outcome = "SUCCESS", incl.cut1 = 0.9, n.cut=2)

M1: gdppc*GINI*UNEMP*urban*YOUTH*mobile*internet*FUEL*POL => SUCCESS

# Increase minimum sufficiency inclusion score to 0.95
eqmcc(data = hh, outcome = "SUCCESS", incl.cut1 = 0.95, n.cut=2)
Error: Nothing to explain. Please check the truth table.
```

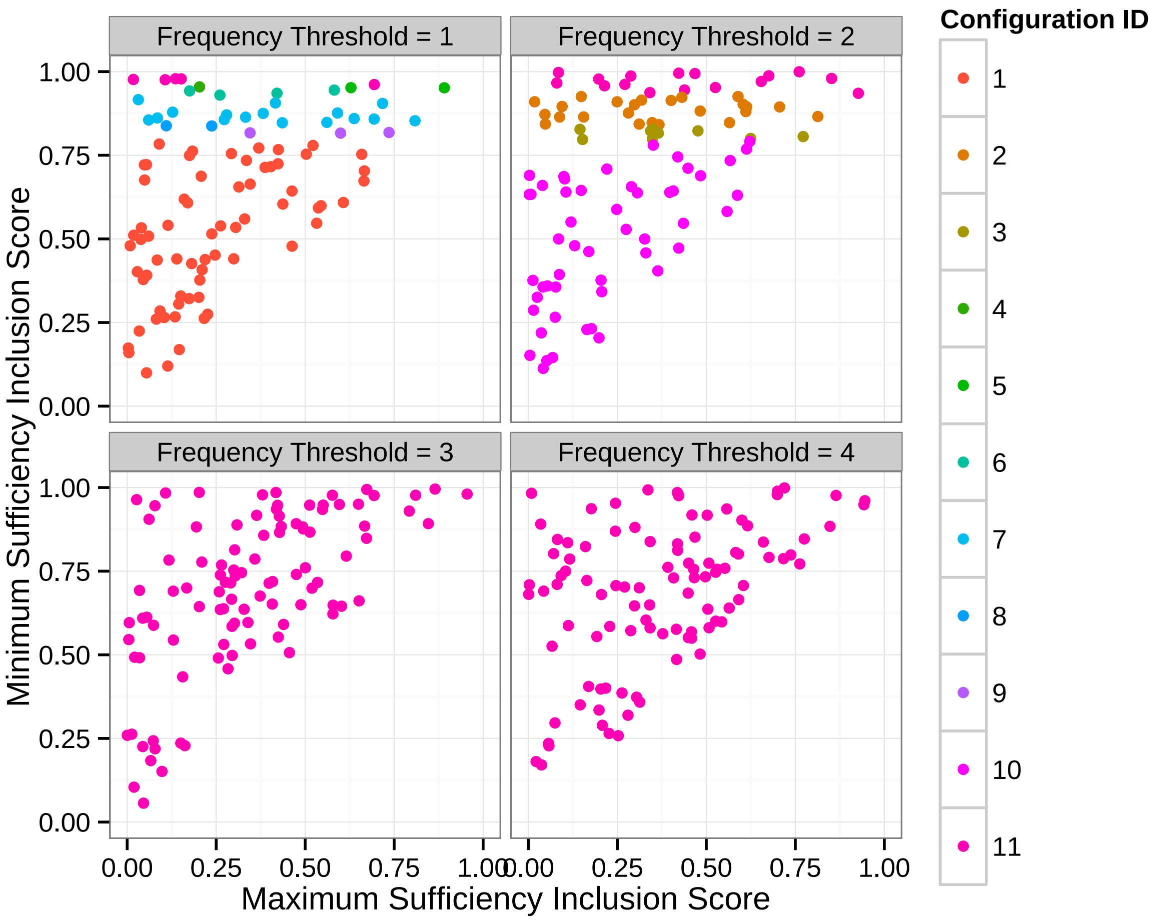
As should be clear from the example above, much time can be spent contrasting the different solutions produced by `eqmcc()` when executing with different reduction parameters. Indeed, given fuzzy set membership scores, the combinations of the minimum sufficiency inclusion scores and minimum frequency thresholds are literally infinite. This is troublesome for researchers, as many hours can be spent finding an optimal combination of parameter values for inference. This is also troublesome for readers, as QCA work rarely reports results for a large number of these combinations. This makes findings vulnerable to invalidity due to artifacts of the method, and prevents consumers of QCA work from knowing whether a given finding is truly robust or, alternatively, specific to just a few of the possible parameter combinations.

Originally developed in Krogslund et al. (2014), the function `fsqca_sim()` provides a facility for QCA users to automatically search for solutions over all possible combinations minimum sufficiency inclusion scores and minimum frequency thresholds. `fsqca_sim()` ordinarily produces three objects:

1. A dataframe where each show gives a combination of minimum and maximum sufficiency inclusion scores with a minimum frequency threshold and the observed QCA result
2. A plot where each point represents a unique combination of minimum and maximum sufficiency inclusion scores and minimum frequency threshold, and its color corresponds to a particular solution
3. A legend connecting the QCA result to the unique identification number for each plotted by point and, thereby, combination of QCA reduction parameters

```
# Run fsqca_sim, turning on plot and turning of verbose execution
out<-fsqca_sim(data = hh, outcome = "SUCCESS", plot = TRUE, verbose = FALSE)

# Examine first 50 rows of results
head(x = out$results, n = 50)
   incl.cut1 incl.cut0                       n.cut config.id
1     0.9997  0.760861 Frequency Threshold = 2        11
2     0.9989  0.719337 Frequency Threshold = 4        11
3     0.9974  0.084675 Frequency Threshold = 2        11
4     0.9954  0.864892 Frequency Threshold = 3        11
5     0.9954  0.422449 Frequency Threshold = 2        11
6     0.9943  0.468035 Frequency Threshold = 2        11
7     0.9941  0.673141 Frequency Threshold = 3        11
8     0.9930  0.336020 Frequency Threshold = 4        11
9     0.9891  0.700002 Frequency Threshold = 4        11
10    0.9875  0.675427 Frequency Threshold = 2        11
11    0.9870  0.287931 Frequency Threshold = 2        11
12    0.9854  0.202468 Frequency Threshold = 3        11
13    0.9852  0.417880 Frequency Threshold = 3        11
14    0.9849  0.418594 Frequency Threshold = 4        11
15    0.9836  0.107816 Frequency Threshold = 3        11
16    0.9828  0.008914 Frequency Threshold = 4        11
```

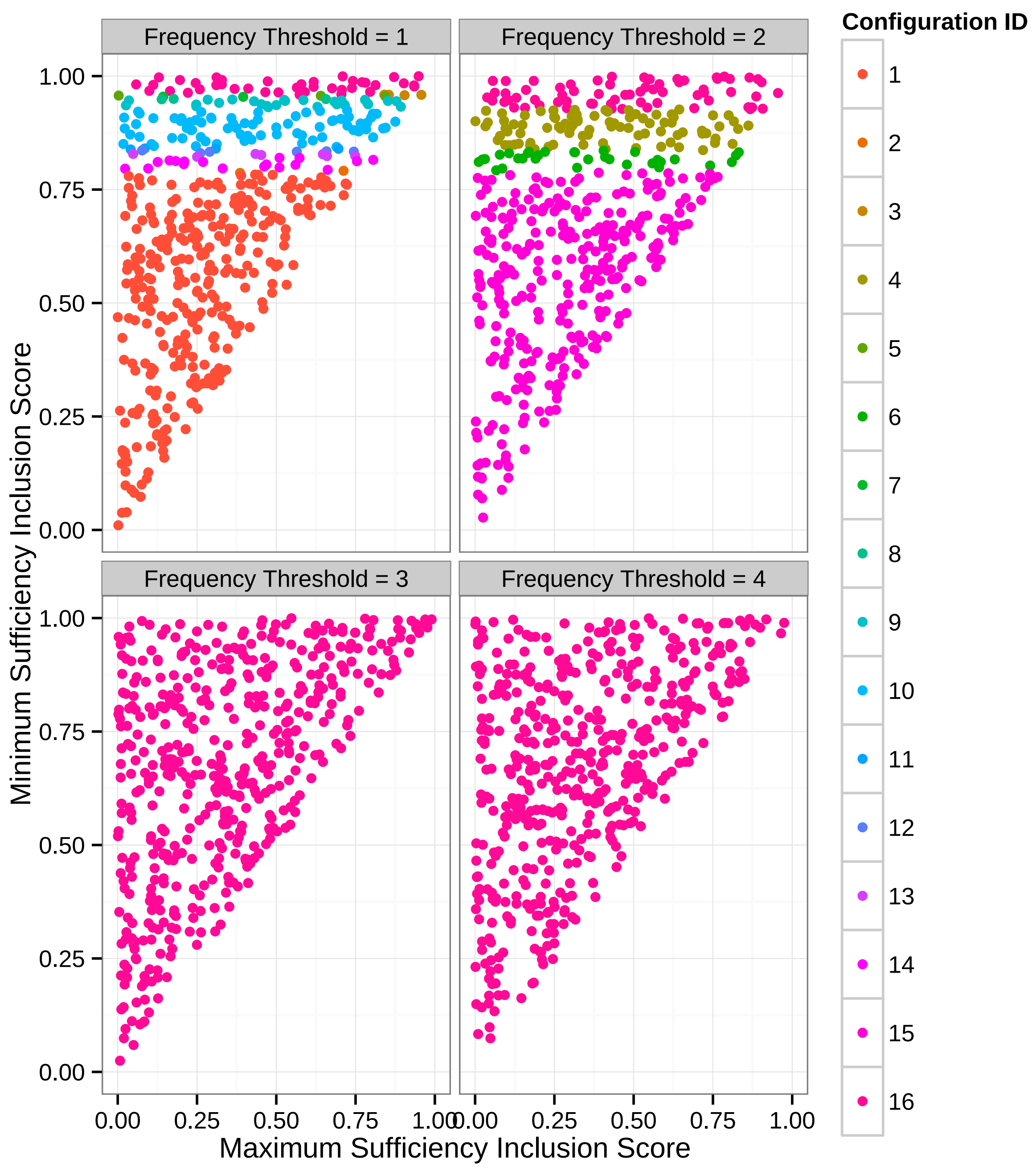
```
17     0.9805  0.954888 Frequency Threshold = 3         11
18     0.9797  0.852004 Frequency Threshold = 2         11
19     0.9790  0.699259 Frequency Threshold = 4         11
20     0.9788  0.135862 Frequency Threshold = 1         11
21     0.9783  0.151681 Frequency Threshold = 1         11
22     0.9782  0.380248 Frequency Threshold = 3         11
23     0.9778  0.197548 Frequency Threshold = 2         11
24     0.9770  0.576434 Frequency Threshold = 3         11
25     0.9770  0.810278 Frequency Threshold = 3         11
26     0.9765  0.017309 Frequency Threshold = 1         11
27     0.9764  0.864400 Frequency Threshold = 4         11
28     0.9762  0.693609 Frequency Threshold = 3         11
29     0.9757  0.422303 Frequency Threshold = 4         11
30     0.9757  0.106647 Frequency Threshold = 1         11
31     0.9705  0.654293 Frequency Threshold = 2         11
32     0.9660  0.079885 Frequency Threshold = 2         11
33     0.9638  0.026448 Frequency Threshold = 3         11
34     0.9618  0.271175 Frequency Threshold = 2         11
35     0.9614  0.694175 Frequency Threshold = 1         11
36     0.9606  0.945341 Frequency Threshold = 4         11
37     0.9576  0.213974 Frequency Threshold = 2         11
38     0.9544  0.203233 Frequency Threshold = 1          4
39     0.9531  0.244813 Frequency Threshold = 4         11
40     0.9525  0.525426 Frequency Threshold = 2         11
41     0.9520  0.628641 Frequency Threshold = 1          5
42     0.9516  0.890955 Frequency Threshold = 1          5
43     0.9501  0.649747 Frequency Threshold = 3         11
44     0.9494  0.596043 Frequency Threshold = 3         11
45     0.9487  0.942631 Frequency Threshold = 4         11
46     0.9475  0.512582 Frequency Threshold = 3         11
47     0.9474  0.550421 Frequency Threshold = 3         11
48     0.9468  0.422612 Frequency Threshold = 3         11
49     0.9457  0.077989 Frequency Threshold = 3         11
50     0.9451  0.438809 Frequency Threshold = 2         11

# Show plot
out$plot
```



```
# Extract legend information for configuration #2
out$legend[sapply(out$legend, function(x){x$config.id})==2]
[[1]]
[[1]]$config.id
[1] 2

[[1]]$config
[1] "gdppc*GINI*UNEMP*urban*YOUTH*mobile*internet*FUEL*POL"

# Extract legend information for configuration #11
out$legend[sapply(out$legend, function(x){x$config.id})==11]
[[1]]
[[1]]$config.id
[1] 11

[[1]]$config
[1] "No Solution"
```

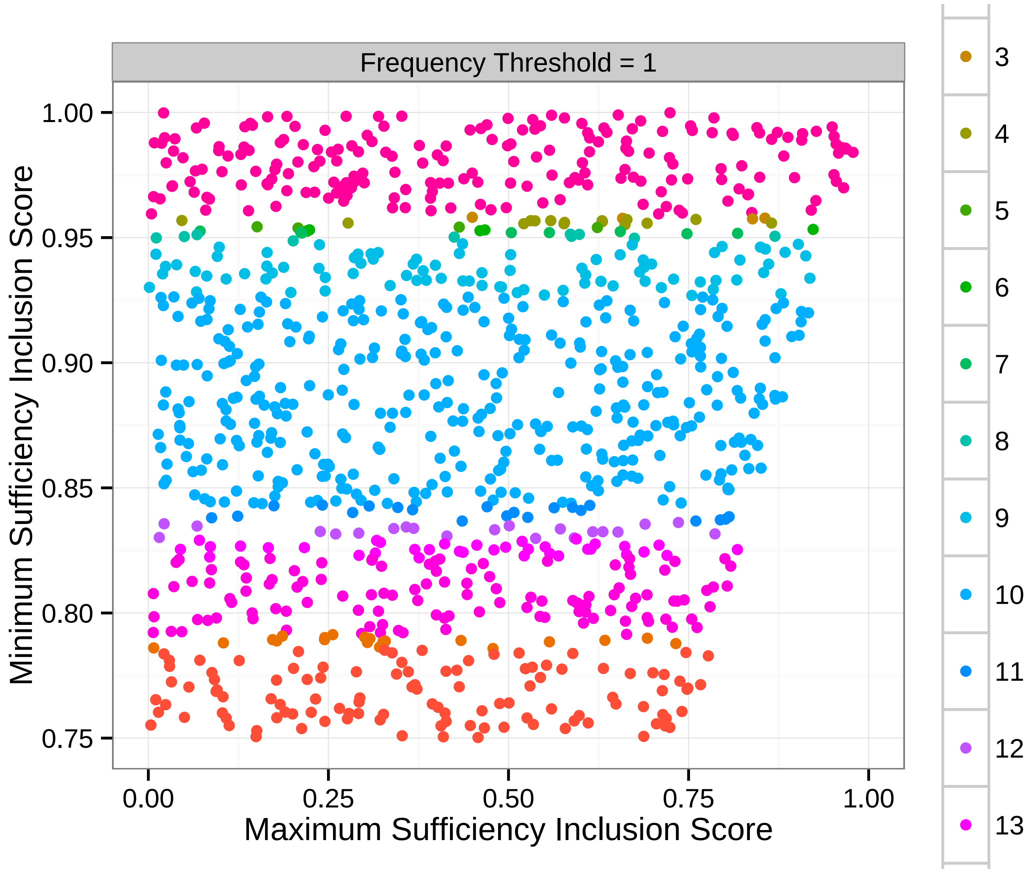
Aside from the base output, the user can increase the number of parameter combinations sampled by `fsqca_sim()` via the `reps` argument. Note that that probability `fsqca_sim()` results display all possible QCA results increases asymptotically with the number of parameter combinations sampled. Whereas only 11 unique solutions were identified when `reps` was set to 100, five extra solutions were identified when the `reps` argument was increased to 500.

```
# Increase repetitions from 100 (default) to 500
out<-fsqca_sim(data = hh, outcome = "SUCCESS", plot = TRUE, verbose = FALSE, reps = 500)

# Show plot
out$plot
```



One can also use `fsqca_sim` arguments to zoom in on certain locations in the parameter space. For instance, the plots above show great solution diversity when `n.cut` is set to 1 and `incl.cut1` is above 0.75. Using the minimum and maximum value arguments in `fsqca_sim()`, the sampling process can be limited to only certain parameter combinations.

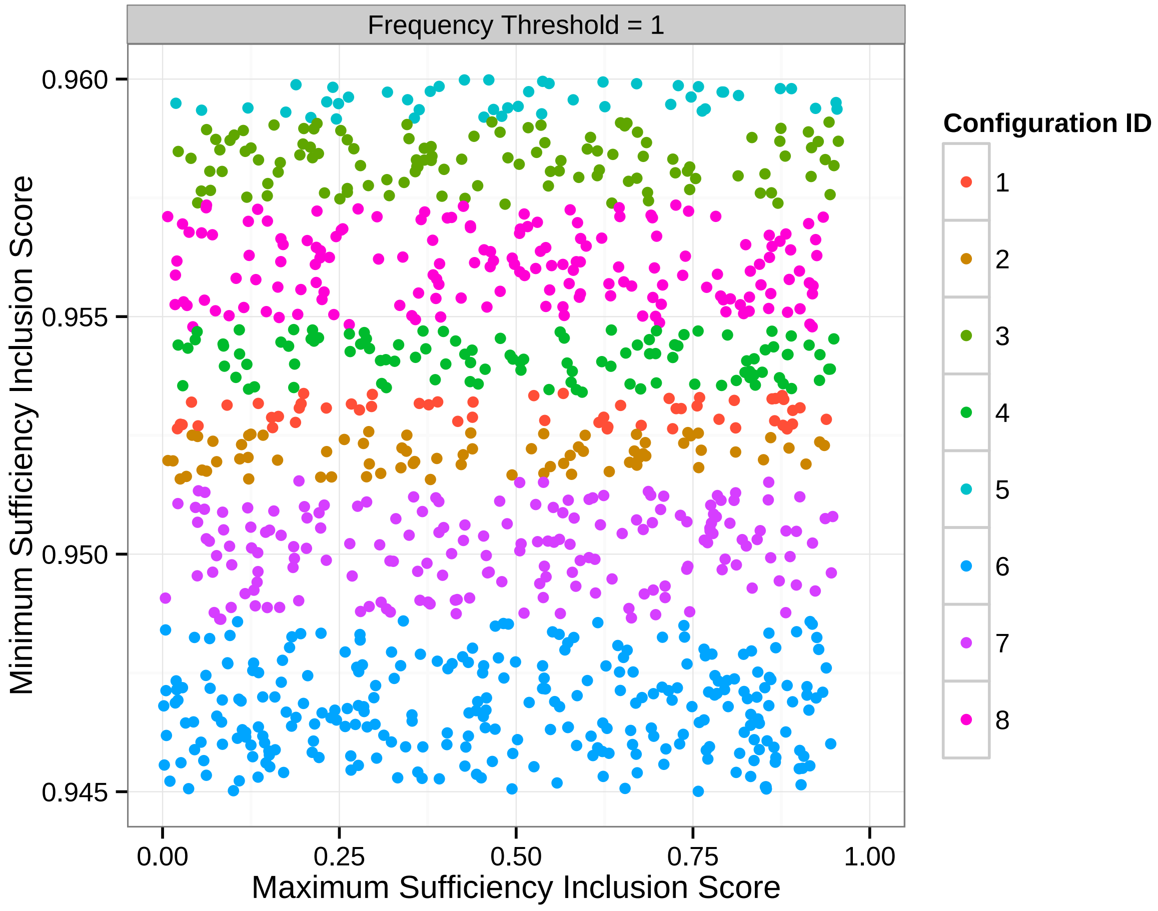
```
# Limit parameter search space to n.cut=1 and incl.cut>=0.75, increase reps to 1000
out<-fsqca_sim(data = hh, outcome = "SUCCESS", plot = T, verbose = F, reps = 1000,
                 min.incl.cut = 0.75, max.n.cut = 1)

# Show plot (using ggplot to limit y-axis range)
out$plot+ggplot2::ylim(0.75,1)
```



Though `fsqca_sim()` only explicitly iterates over the `eqmcc()` parameters `incl.cut1`, `incl.cut0`, and `n.cut`, the user can pass additional arguments to the reduction function. For instance, one could request that the solution type returned not be "complex" but rather "parsimonious", meaning that the algorithm will attempt to simplify QCA solutions by assuming certain outcomes for paths not observed in the data. The argument used to register such a request (`include`) can be passed directly to `eqmcc()`.

```
# Request parsimonious QCA solution
out<-fsqca_sim(data = hh, outcome = "SUCCESS", plot = TRUE, verbose = FALSE, reps=1000,
                 min.incl.cut = 0.945, max.incl.cut = 0.96, max.n.cut = 1, include=c("1",

# Show plot (using ggplot to limit y-axis range)
```

```
out$plot+ggplot2::ylim(0.945,0.96)
```



Often, a researcher will be interested in examining how QCA results change following alterations to more than just the sufficiency inclusion scores and minimum frequency thresholds. **stcm** provides a facility for more flexibly carrying out such simulations with the function `eqmcc_options()`. Consider a second data set containing information on the consolidation of income security programs in the early 1920s in Western Europe and North America.[2] Unlike in the previous dataset, set membership scores here are of the binary or "crisp" variety. Aside from the outcome of interest (noted as `CON`), the dataset also contains membership scores for a number of explanatory set memberships, including:

- `LG`: liberal government
- `CG`: catholic government

[2]This data is taken from Hicks et al. (1995).

- `PS`: patriarchal statism
- `DU`: unitary democracy
- `WM`: high working-class mobilization

```
# Load dataset
data(hicks_20)

# Inspect dataset
print(hicks_20)
   Case LG CG PS DU WM CON
1  AT20  0  0  1  0  1   1
2  AU20  0  0  0  0  1   0
3  BE20  0  1  1  1  1   1
4  CA20  1  0  0  0  0   0
5  CH20  1  0  0  0  0   0
6  DE20  0  0  1  0  1   1
7  DK20  1  0  1  1  1   1
8  FR20  1  0  1  1  0   0
9  IT20  0  0  1  0  1   1
10 NL20  0  1  1  1  0   1
11 NO20  1  0  0  1  0   0
12 NZ20  1  0  0  1  0   0
13 SE20  1  0  1  1  1   1
14 UK20  1  0  0  1  1   1
15 US20  1  0  0  0  0   0
```

While altering the minimum sufficiency inclusion score will not alter QCA results for crisp set data (save for extreme values), several reduction parameters remain relevant, including the minimum frequency threshold and the solution type. In addition, QCA users may wish to variably alter the values passed to other arguments in `eqmcc()`, such as the explanatory variables included in the minimzation (`conditions`), whether the negation of the outcome should be investigated (`neg.out`), and whether the row dominance principle should be used to exclude dominated prime implicants from the solution (`row.dom`).

The `eqmcc_options()` function allows the user to specify a number of different values passed to each `eqmcc()` argument, and then returns the results produced for all possible combinations of those parameter values. The first step in using the function is to call `eqmcc_args()`, which generates a named list of all the arguments contained in the `eqmcc()` function. The user can then add desired argument values by replacing the appropriate list element. For instance, to run the function using minimum frequency thresholds of 1, 2, and 3, one can add these values to the `n.cut` element of the list via `arglist$n.cut<-1:3`, where the options list is stored under the object name `arglist`. Note that, since some `eqmcc()` functions take concatenated vectors as arguments, these various options must be added to the `eqmcc_options` list as lists, rather than vectors. For example, requesting complex and parsimonious solution types via the `include` argument would be speified as `arglist$include<-list("1", c("1", "?"))`.

```
# Generate list of eqmcc arguments
arglist<-eqmcc_args()

# Add values for outome, neg.out, conditions, n.cut, include, and row.dom
arglist$outcome<-"CON"
arglist$neg.out<-c(TRUE, FALSE)
arglist$conditions<-list(c("LG", "CG", "PS"), c("LG", "CG", "PS", "DU", "WM"))
arglist$n.cut<-1:3
arglist$include<-list("1", c("1", "?"))
arglist$row.dom<-c(TRUE, FALSE)

# Inspect first 10 elements of "filled-in" arglist
head(arglist,10)
$outcome
[1] "CON"

$neg.out
[1]  TRUE FALSE

$conditions
$conditions[[1]]
[1] "LG" "CG" "PS"

$conditions[[2]]
[1] "LG" "CG" "PS" "DU" "WM"


$relation
[1] NA
```

```
$n.cut
[1] 1 2 3

$incl.cut1
[1] NA

$incl.cut0
[1] NA

$explain
[1] NA

$include
$include[[1]]
[1] "1"

$include[[2]]
[1] "1" "?"


$row.dom
[1]  TRUE FALSE
```

Once the user has added all parameters of interest to the `eqmcc_args()` list, these options are then forwarded to the `eqmcc_options()` function via the `eqmcc.options.list` argument. The only other argument required is a specification of the dataset object. Any call to `eqmcc_options` will yield a named list containing two objects. The first is a data frame entitlted `opts`, where each row gives to parameter values for a particular run that were fed into `eqmcc()`. The second is a list named `results` that contains the `solution` object created by a given call to `eqmcc()`.

```
# Run eqmcc_options
out<-eqmcc_options(data = hicks_20, eqmcc.options.list = arglist, verbose = FALSE)

# Inspect "options" data frame
head(out$opts)
  outcome neg.out         conditions n.cut include row.dom list.id
1     CON    TRUE         LG, CG, PS     1       1    TRUE       1
2     CON   FALSE         LG, CG, PS     1       1    TRUE       2
```

```
3      CON    TRUE LG, CG, PS, DU, WM      1       1    TRUE        3
4      CON   FALSE LG, CG, PS, DU, WM      1       1    TRUE        4
5      CON    TRUE         LG, CG, PS      2       1    TRUE        5
6      CON   FALSE         LG, CG, PS      2       1    TRUE        6

# Inspect "results" list
head(out$results)
[[1]]
[[1]][[1]]
[1] "lg*cg*ps"


[[2]]
[[2]][[1]]
[1] "lg*PS"


[[3]]
[[3]][[1]]
[1] "LG*cg*DU*wm"    "LG*cg*ps*wm"    "lg*cg*ps*du*WM"


[[4]]
[[4]][[1]]
[1] "LG*cg*DU*WM"    "lg*CG*PS*DU"    "lg*cg*PS*du*WM"


[[5]]
[1] NA

[[6]]
[[6]][[1]]
[1] "lg*PS"
```

Used in tandem, these two objects permit the user to easily jump between different QCA parameter specifications and the corresponding QCA results. For example, using the dataset on the consolidation of income security programs in the early 1920s, a researcher might be interested in assessing how QCA results change as the minimum frequency threshold is increased. A quick examination of the `opts` data frame shows that a suitable basic result is found in row 26, where `neg.out==FALSE`, `incl.cut==1`, `n.cut==1`, row.dom==FALSE, and only three explanatory factors are used. Results that differ only in the value of n.cut are given in rows 30 and 34. Using the row number of a given parameter set (or, alternatively, the value given in the list.id column), the appropriate QCA solution can be retreived from the results list by extracting the list node of the same index number. For instance, the QCA solution to the parameter combination in row 26 can be extracted via out$results[[26]], where out is the name of the object containing eqmcc_options() results.

Here, the results show that increasing the minimum frequency threshold from one to two has no impact on the solution, while the move from two to three does change the result. Other examples shown below indicate that the base results are not impacted by moving from a complex to a parsimonious solution, but that adding new causal conditions to the algorithm does produce notably different results.

```
# Inspect how increasing n.cut changes solutions
out$results[[26]] # Base solution, n.cut=1
[[1]]
[1] "lg*PS"
out$results[[30]] # Base solution, n.cut=2
[[1]]
[1] "lg*PS"
out$results[[34]] # Base solution, n.cut=3
[[1]]
[1] "lg*cg*PS"

# Alternative way to inspect solutions
out$results[c(26,30,34)]
[[1]]
[[1]][[1]]
[1] "lg*PS"


[[2]]
[[2]][[1]]
[1] "lg*PS"
```

```
[[3]]
[[3]][[1]]
[1] "lg*cg*PS"

# Inspect how altering include changes solutions
out$results[[26]] # Base solution, include="1"
[[1]]
[1] "lg*PS"
out$results[[38]] # Base solution, include=c("1", "?")
[[1]]
[1] "lg*PS"

# Inspect how modifying included conditions changes solutions
out$results[[26]] # Base solution, conditions=c("LG", "CG", "PS")
[[1]]
[1] "lg*PS"
out$results[[28]] # Base solution, conditions=c("LG", "CG", "PS", "DU", "WM")
[[1]]
[1] "LG*cg*DU*WM"    "lg*CG*PS*DU"    "lg*cg*PS*du*WM"
```

Measurement and Model Specification Error Sensitivity

Aside from result sensitivity due to the selection of boolean reduction parameters, a number of critiques have pointed to measurement error and model specification error as a possible culprit behind the unreliability of QCA results (Hug, 2013; Lucas and Szatrowski, 2014; Krogslund and Michel, 2014b). In response to these concerns, **stcm** contains several functions that take a given dataset and create an arbitrarily large number of copies that contain certain types and amounts of measurement or model error. `eqmcc()` can then be easily applied over the entire list of perturbed datasets, allowing the user to get a sense of how sensitive his or her results may be to these types of error.

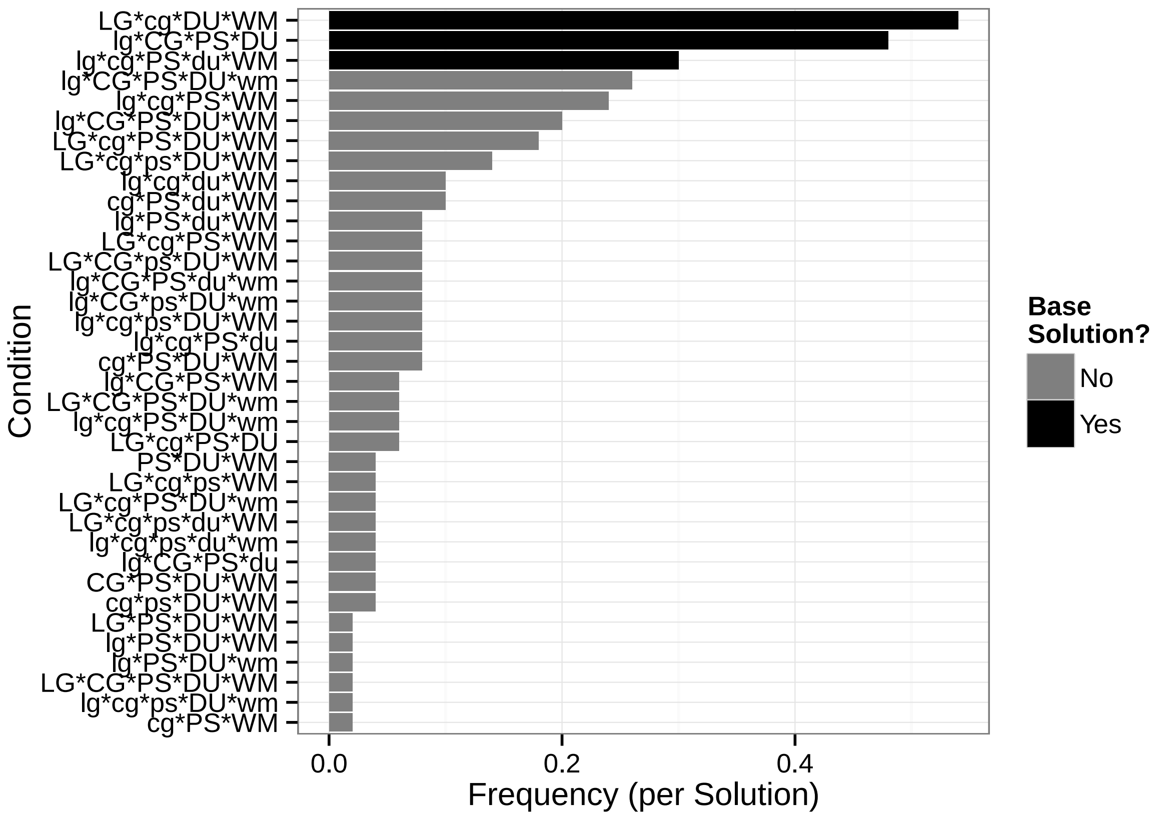
The first of these functions is `meas_error()`, which creates datasets perturbed with measurement error. The manner in which these errors are introduced to the dataset depends upon whether the data contain crisp- or fuzzy-set membership scores. In the former case (passed to the `meas_error` function by setting `type=="crisp"`), a given proportion of all data points are switched from either zero to one, or one to zero. The proportion of points affected can be controled by the `error.level` argument, which takes proportion values ranging from zero to one. For example, we can use the following code to generate 50 perturbed copies of the `hicks_20` data set, where five percent of points will exhibit measurement error.

```
# Run meas_error
out<-meas_error(data = hicks_20, case.ids = "Case", outcome = "CON", type = "crisp",
                error.level = 0.05, reps = 50, analyze = F)

# Inspect parameter specifications
out$parameter.specs
  error.level reps
1        0.05   50

# Inspect a few datasets
cbind(out$datasets[[15]]," "=":::::", out$datasets[[39]])
   LG CG PS DU WM CON       LG CG PS DU WM CON
1   0  0  1  0  0   1 :::::  0  0  1  0  1   1
2   0  0  0  0  1   0 :::::  0  0  0  0  0   0
3   0  1  1  1  1   1 :::::  0  1  1  1  1   1
4   1  1  0  0  0   0 :::::  1  0  0  0  0   0
5   1  0  0  0  0   0 :::::  1  0  0  0  0   0
6   0  0  1  0  1   1 :::::  0  1  1  0  1   1
7   1  0  1  1  1   1 :::::  1  0  1  0  1   1
8   1  0  1  1  0   0 :::::  1  0  1  1  0   0
9   0  0  1  0  1   1 :::::  0  0  1  0  1   1
10  0  1  1  1  0   1 :::::  0  1  1  1  0   1
11  1  0  0  1  0   0 :::::  1  0  0  1  0   0
12  1  0  1  1  0   0 :::::  1  0  0  1  0   1
13  1  0  1  1  1   1 :::::  1  0  1  1  1   1
14  1  0  0  1  1   1 :::::  1  0  0  1  1   1
15  1  1  0  0  0   1 :::::  1  1  0  0  0   0
```

While the user is free to apply the `eqmcc()` function across these datasets in whatever manner is most expedient, `meas_error()` also contains a simple analytical facility that will apply `eqmcc()` and a given set of parameter values to each of the perturbed datasets. It then calculates and plots the frequency with which each configuration of explanatory factors is found in a QCA solution. Calling this functionality requires the user to specify `analyze=TRUE`, as well as to pass any required `eqmcc()` arguments directly to that function. The frequency table and plot can be extracted from the resulting object via the names `results.freq` and `plot`, respectively. Both of these objects contain indications for whether a given configuration appears in the `eqmcc()` solution produced when there is *no* measurement error present. In the `results.freq` data frame, this information is contained in the `base.sol` column, while the plot shows these solutions by shading each bar in black, rather than grey.

```
# Run meas_error requesting analysis
out<-meas_error(data = hicks_20, case.ids = "Case", outcome = "CON", type = "crisp",
                error.level = 0.05, reps = 50, analyze = T)

# Inspect result frequency table
out$result.freq
           config count base.sol
1        cg*PS*WM     1       No
2  lg*cg*ps*DU*wm     1       No
3  LG*CG*PS*DU*WM     1       No
4     lg*PS*DU*wm     1       No
5     lg*PS*DU*WM     1       No
6     LG*PS*DU*WM     1       No
7     cg*ps*DU*WM     2       No
8     CG*PS*DU*WM     2       No
9     lg*CG*PS*du     2       No
10 lg*cg*ps*du*wm     2       No
11 LG*cg*ps*du*WM     2       No
12 LG*cg*PS*DU*wm     2       No
13    LG*cg*ps*WM     2       No
14       PS*DU*WM     2       No
15    LG*cg*PS*DU     3       No
16 lg*cg*PS*DU*wm     3       No
17 LG*CG*PS*DU*wm     3       No
18    lg*CG*PS*WM     3       No
19    cg*PS*DU*WM     4       No
20    lg*cg*PS*du     4       No
21 lg*cg*ps*DU*WM     4       No
22 lg*CG*ps*DU*wm     4       No
23 lg*CG*PS*du*wm     4       No
```

```
24 LG*CG*ps*DU*WM      4        No
25     LG*cg*PS*WM     4        No
26     lg*PS*du*WM     4        No
27     cg*PS*du*WM     5        No
28     lg*cg*du*WM     5        No
29 LG*cg*ps*DU*WM      7        No
30 LG*cg*PS*DU*WM      9        No
31 lg*CG*PS*DU*WM     10        No
32     lg*cg*PS*WM    12        No
33 lg*CG*PS*DU*wm     13        No
34 lg*cg*PS*du*WM     15       Yes
35     lg*CG*PS*DU    24       Yes
36     LG*cg*DU*WM    27       Yes

# Inspect plot
out$plot
```



In the case where the data contain fuzzy set membership scores, measurement error is introduced to the data in a slightly different fashion. As before, `error.level` governs the proportion of data points that will be altered, and `type` indicates the type of set membership scores in the data (set to `type="fuzzy"` in this case). Unlike with crisp-set data, however, relatively small amounts of measurement error can be introduced into the scores. Binary set membership scores can only be equal to zero or one – thus measurement error can only be zero or one hundred percent. Fuzzy set membership scores, however, can take on any value ranging from zero to one, which allows the introduction of measurement error ranging from zero up to one hundred percent. Accordingly, the user of `meas_error()` must specify the `fuzzy.range` argument, which gives the size of a window around a given fuzzy set score from which its purtubed value will be randomly selected. For instance, for a fuzzy set score of 0.75 and where `fuzzy.range` is set to 0.1, the purturbed value for this score will be selected an random from a uniform distribution between 0.7 and 0.8.

An example using the Arab Spring protest data set is shown below, where the proportion of data points impacted by measurement error is set at five percent, and the `fuzzy.range` is set to 0.1. To make the solutions more presentable, the call has also requested that the sufficiency inclusion score be set relatively high (`incl.cut1=0.9`) and that only the parsimonious solution be returned (`include=c("1", "?")`).

A quick comparison of two perturbed copies of the dataset (specifically numbers 14 and 35) gives a sense of how the observed values are impacte. The value of `GINI` in the second row, for example, is set at 0.740 in one copy and 0.746 in another. Similarly, whereas `GDPPC` in the third row stands at 0.082 in the first dataset, it is changed to 0.050 in the second. Overall, a quick inspection confirms that a relatively small amount of data points change values across each dataset, and that these changes all fall within a +/- 0.05 window of their original values.

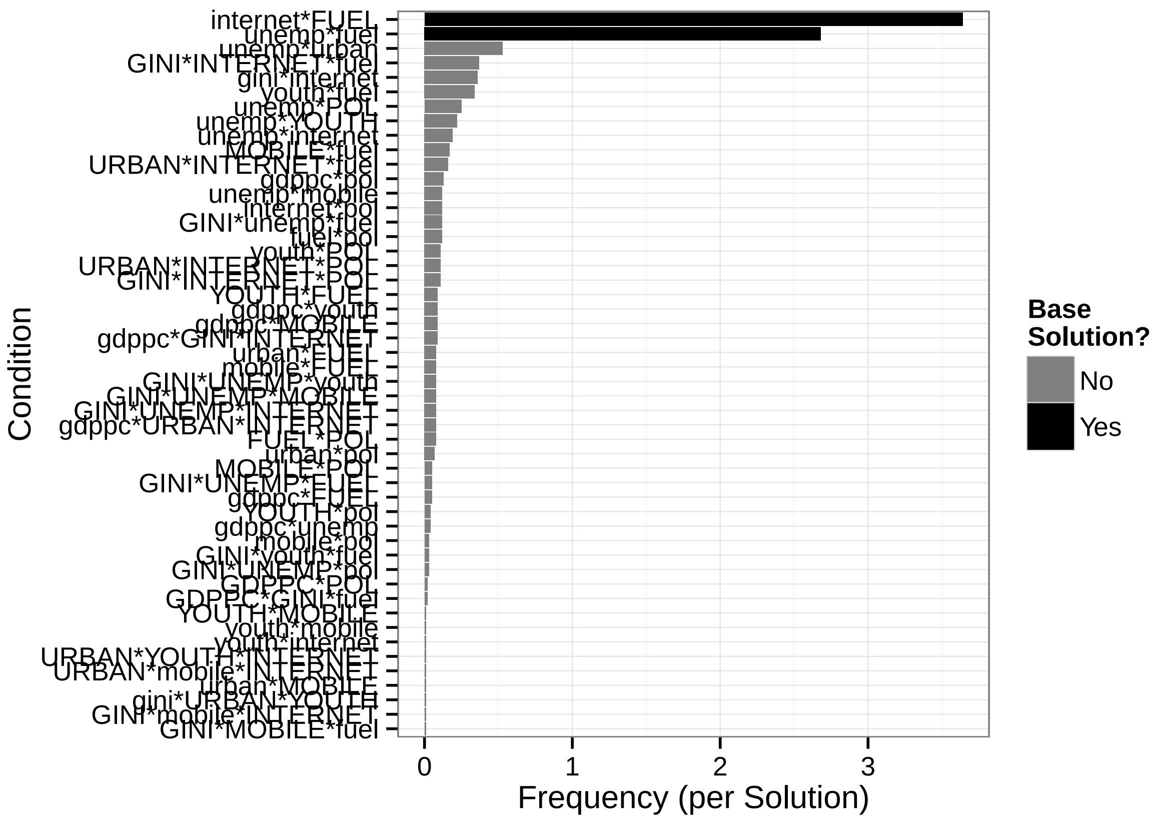
```r
# Run meas_error requesting analytics
out<-meas_error(data = hh, outcome = "SUCCESS", type="fuzzy",
                error.level = 0.05, fuzzy.range = 0.1, reps = 100, analyze = T,
                incl.cut1=.9, include=c("1", "?"))

# Inspect a few datasets
cbind(out$datasets[[14]][1:10,1:3], " "=":::::", out$datasets[[35]][1:10,1:3])
```

```
   GDPPC GINI UNEMP       GDPPC   GINI  UNEMP
1    0.84 0.42  0.58 :::::  0.84 0.4200 0.5800
2    0.16 0.74  1.00 :::::  0.16 0.7400 1.0000
3    0.05 0.11  0.74 :::::  0.05 0.1100 0.6901
4    0.11 0.68  0.84 :::::  0.11 0.7237 0.8400
5    0.95 0.11  0.11 :::::  0.95 0.1100 0.1100
6    1.00 0.79  0.01 :::::  1.00 0.7900 0.0100
7    0.01 0.01  0.95 :::::  0.01 0.0100 0.9897
8    0.58 0.37  0.42 :::::  0.58 0.3700 0.4200
9    0.26 1.00  0.79 :::::  0.26 1.0000 0.7900
10   0.53 0.58  0.47 :::::  0.53 0.5800 0.4700
```

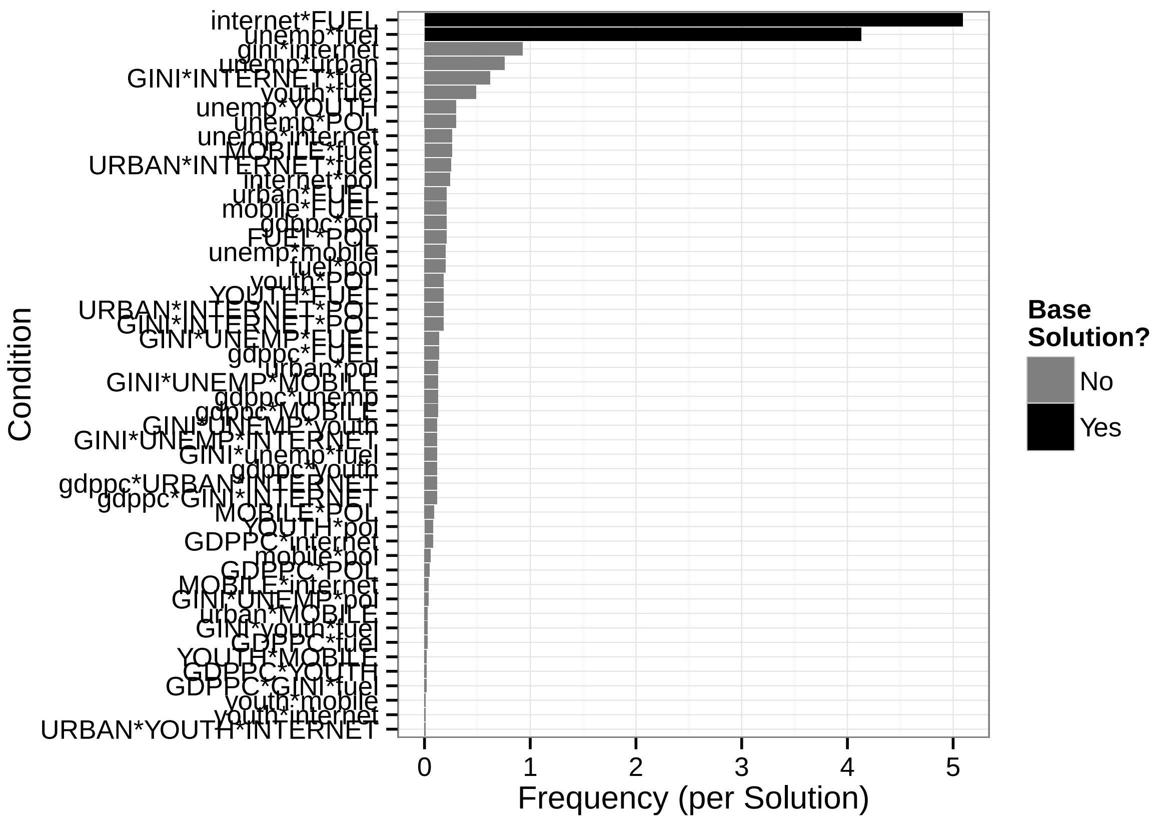
In terms of results, one can see from the frequency table and its corresponding plot that the original fsQCA solutions derived in the absence of measurement error (`internet*FUEL` and `unemp*fuel`) are relatively robust to the addition of small amounts of measurement error. Several addition configurations are yielded across the perturbed datasets, but their appearances are relatively infrequent compared to the original solutions. By contrast, increasing the percentage of points impacted by measurement error from 5 percent to 10 percent, a new fsQCA solution rises to prominence over the original set of solutions. Where as the original `internet*FUEL` and `unemp*fuel` configurations are present roughly 4-5 times per full fsQCA solution, the new configuration `gini*internet` now appears an average of more than 10 times per solution. With this information in hand, the user can return to his/her fuzzy-set codings on these variables to ensure that the correct scores have been assigned.

```
# Inspect result frequency table
tail(out$result.freq)
              config count base.sol
48        youth*fuel    34       No
49     gini*internet    36       No
50 GINI*INTERNET*fuel    37       No
51       unemp*urban    53       No
52        unemp*fuel   268      Yes
53     internet*FUEL   364      Yes

# Inspect plot
```

```
out$plot
```



```
# Run meas_error requesting analytics, with larger error level
out.larger<-meas_error(data = hh, outcome = "SUCCESS", type="fuzzy",
                error.level = 0.1, fuzzy.range = 0.1, reps = 100, analyze = T,
                incl.cut1=.9, include=c("1", "?"))

# Inspect plot
out.larger$plot
```



Much like `meas_error()`, the function `mod_error()` returns a number of datasets containing certain perturbances. Instead of measurement error, however, the function simulates model specification error for QCA by adding one or more variables to the data that are comprised of random set membership scores. The type of model specification error tested here is somewhat specific to QCA, in the sense that omitting causal variables will cause QCA solutions to *always* be incorrect. The only type of model specification problem that does not condemn QCA results to full invalidity is the *admission* of causally-irrelevant variables. `mod_error()` allows the researche to assess just how sensitive QCA result might be to this sort of admitted variable bias.

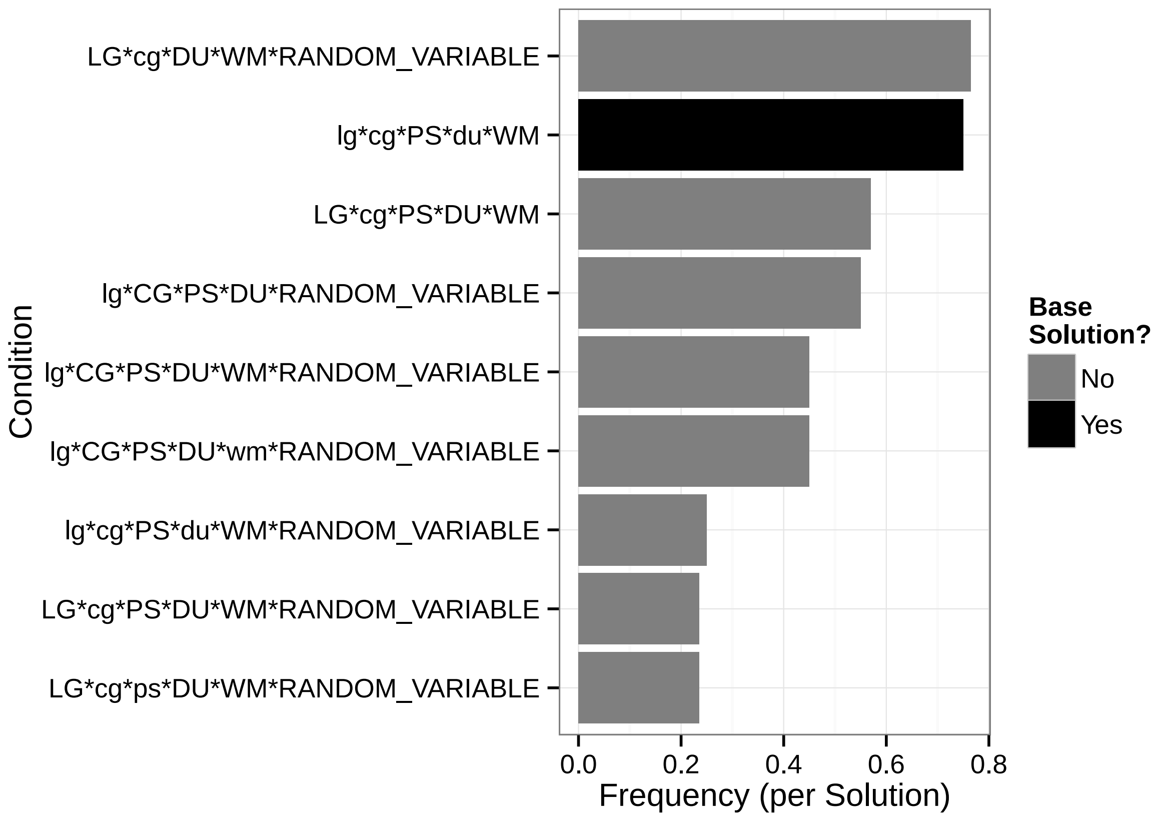
A primary use of `mod_error()` may be to generate many copies of a given dataset combined with one or more causally-irrelevant variables. The examples below show how, using the `hicks_20` dataset, one can create a number of new datasets containing one or more causally-irrelevant variables. In this case, given that we are dealing with crisp-set data, the `type` argument must be set to "crisp". The `number` argument is used to alter the number of causally-irrelevant variables added to each copy of the dataset. In the examples below, one and then three such variables are added. Finally, the `reps` argument is used to specify the number of dataset copies to be generated.

```
# Run mod_error, requesting 100 datasets containing 1 crisp-set random variable each
out<-mod_error(data = hicks_20, case.ids = "Case", outcome = "CON", type = "crisp",
               number = 1, reps = 100, analyze = F)

# Inspect a sample generated dataset
out$datasets[63]
[[1]]
   LG CG PS DU WM CON random_variable_1
1   0  0  1  0  1   1                 0
2   0  0  0  0  1   0                 1
3   0  1  1  1  1   1                 1
4   1  0  0  0  0   0                 0
5   1  0  0  0  0   0                 0
6   0  0  1  0  1   1                 0
7   1  0  1  1  1   1                 1
8   1  0  1  1  0   0                 1
9   0  0  1  0  1   1                 1
10  0  1  1  1  0   1                 1
11  1  0  0  1  0   0                 0
12  1  0  0  1  0   0                 0
13  1  0  1  1  1   1                 0
14  1  0  0  1  1   1                 1
15  1  0  0  0  0   0                 1

# Run mod_error, requesting 200 datasets containing 3 crisp-set random variables each
out<-mod_error(data = hicks_20, case.ids = "Case", outcome = "CON", type = "crisp",
               number = 3, reps = 200, analyze = F)

# Inspect a sample generated dataset
out$datasets[19]
[[1]]
   LG CG PS DU WM CON random_variable_1 random_variable_2 random_variable_3
1   0  0  1  0  1   1                 0                 1                 0
2   0  0  0  0  1   0                 1                 1                 0
3   0  1  1  1  1   1                 1                 1                 0
4   1  0  0  0  0   0                 0                 1                 1
5   1  0  0  0  0   0                 0                 0                 0
6   0  0  1  0  1   1                 0                 0                 1
7   1  0  1  1  1   1                 0                 0                 1
```

```
8   1  0  1  1  0   0                 0                 1                 0
9   0  0  1  0  1   1                 1                 1                 1
10  0  1  1  1  0   1                 0                 0                 0
11  1  0  0  1  0   0                 1                 0                 1
12  1  0  0  1  0   0                 1                 0                 0
13  1  0  1  1  1   1                 1                 1                 0
14  1  0  0  1  1   1                 1                 1                 0
15  1  0  0  0  0   0                 1                 1                 0
```

Again, as with `meas_error()`, the `mod_error()` function contains a simple frequency analysis facility. This can be accessed by setting the `analyze` argument to `TRUE`. A frequency table and plot containing these results are stored as the `result.freq` and `plot` objects, respectively. As is clear below, the base results appear relatively robust, though a random variable paired with the configuration LG*cg*DU*WM does appear often as a QCA solution.

```
# Run mod_error, requesting 100 datasets containing 1 crisp-set random variable each
out<-stcm::mod_error(data = hicks_20, case.ids = "Case", outcome = "CON", type = "crisp"
                      number = 1, reps = 200, analyze = T)

# Inspect frequency table
out$result.freq
                          config count base.sol
1 LG*cg*ps*DU*WM*RANDOM_VARIABLE    47       No
2 LG*cg*PS*DU*WM*RANDOM_VARIABLE    47       No
3 lg*cg*PS*du*WM*RANDOM_VARIABLE    50       No
4 lg*CG*PS*DU*wm*RANDOM_VARIABLE    90       No
5 lg*CG*PS*DU*WM*RANDOM_VARIABLE    90       No
6    lg*CG*PS*DU*RANDOM_VARIABLE   110       No
7                 LG*cg*PS*DU*WM   114       No
8                 lg*cg*PS*du*WM   150      Yes
9    LG*cg*DU*WM*RANDOM_VARIABLE   153       No

# Inspect frequency plot
out$plot
```



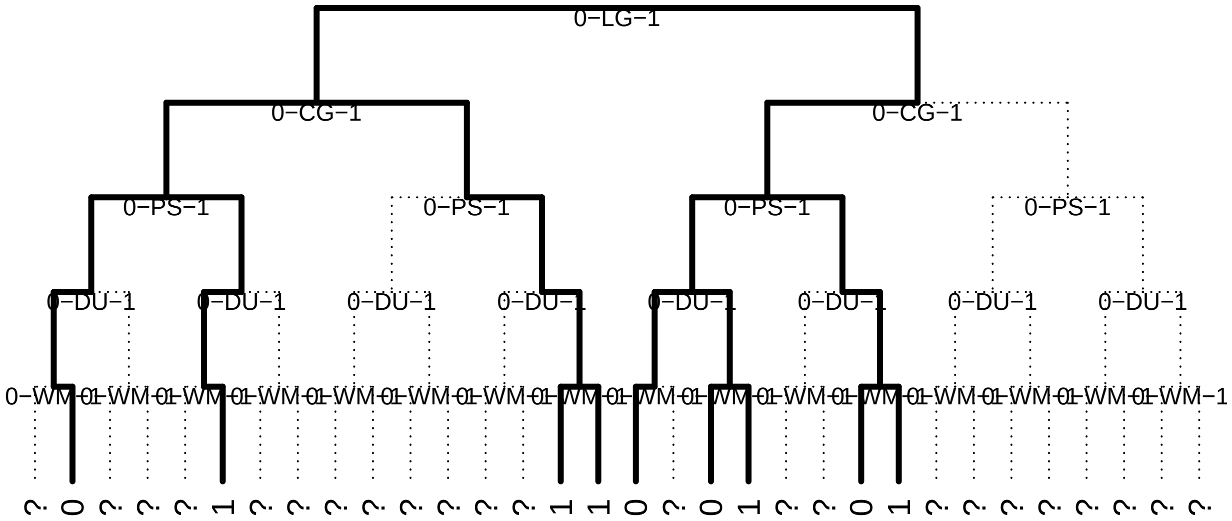
Identifying the Sources of Result Sensitivity

While there is an emerging consensus in the literature that QCA results can be unreliable under certain conditions, there is far less agreement as to *why* result sensitivity emerges to such a degree under each set of circumstances. If there is one agreed upon master source of this sensitivity, however, it almost certainly concerns the proportion of possible paths to an outcome that are actually observed in the data (Schneider and Wagemann, 2012). Altering reduction parameters, such as the sufficiency inclusion scores or the minimum frequency threshold, often changes the universe of cases that is used for analysis. Likewise, measurement error can incorrectly reveal or conceal certain causal paths, and model specification error can radically alter the set of potential paths over which QCA searches.

Identifying and rectifying the sources of result sensitivity in QCA results is a largely dataset-specific task, as the contribution of parameter values, measurement error, and model specification error to result sensitivity through the revelation of potential causal paths depends upon the initial distribution of observed paths in the data. **stcm** provides a function to help users of QCA and other set-theoretic methods identify the sources of result sensitivty in an empirical dataset. Specifically, the function `paths_summary()` returns a number of statistics and other objects that paint a full picture of the observed and unobserved paths in a dataset.

Consider the `hicks_20` dataset, which was shown to have some potential sensitivity to measurement and model perturbations. Running `paths_summary()` can help identify some places where a researcher using this dataset might look for sources of sensitivity. For instance, as is shown in the code below, the data contain a relatively small percentage of the total number of possible paths to the outcome. Across the 15 cases in the dataset, only 9 of the 32 unique paths to the outcome are observed empirically. With roughly 70 percent of the possible causal configurations unobserved – clearly shown by a sparse dendrogram plot of potential paths to the outcome – it appears likely that QCA results produced from this dataset could be easily impacted by changing reduction parameter values, or by measurement or model specification error.

```
# Run paths_summary
out<-paths_summary(data = hicks_20, case.ids = "Case", outcome = "CON", plot = T, verbos
```



```
# Get paths plot
out$paths.plot

# Get number of cases
out$n.cases
[1] 15

# Get number of possible paths to the outcome
out$n.paths.possible
[1] 32

# Get the number of possible paths empirically observed
out$n.paths.observed
[1] 9

# Get the percentage of possible paths empirically observed
out$n.paths.percent
[1] 28.12
```

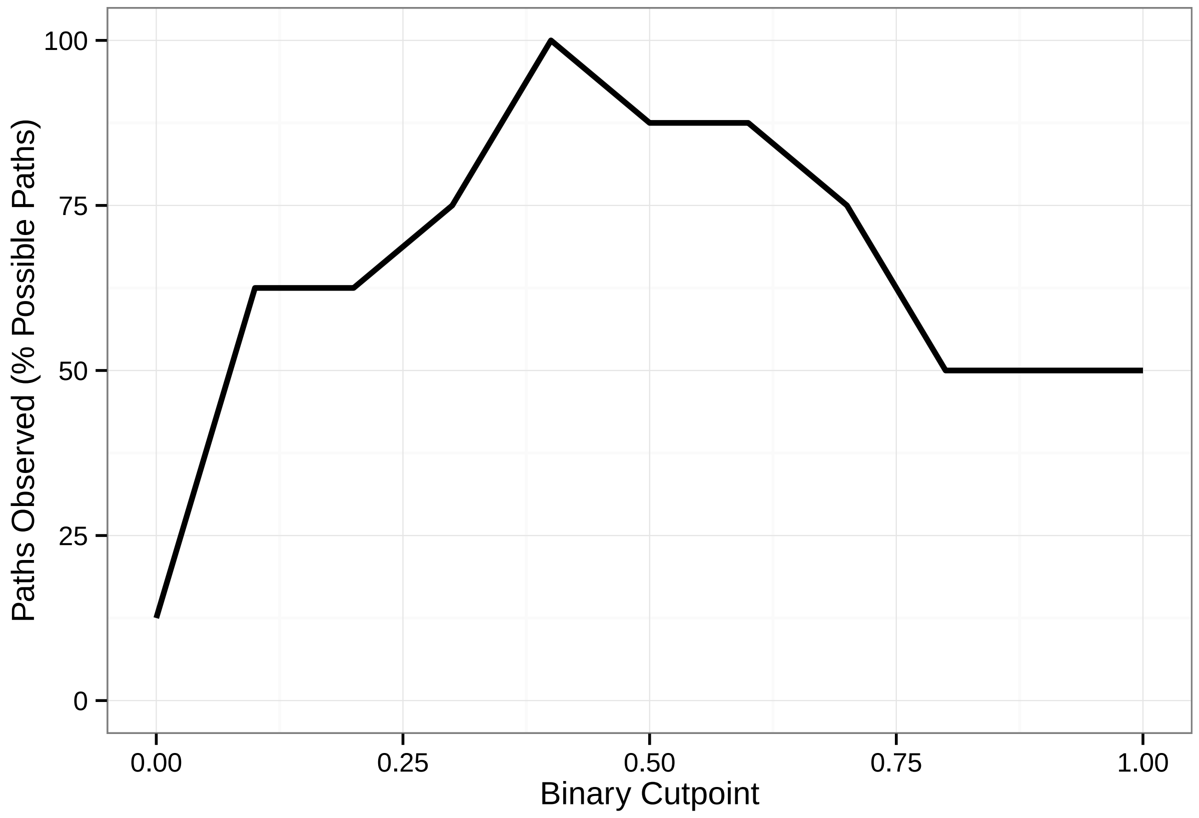
Aside from a high expectation of result sensitivity based upon the low proportion of observed paths in the data, one can also say something about what types of cases or which variables would be most consequential in altering the layout of observed paths. `paths_summary()` contains an object that measures the diversity of values observed in a dataset for a particular value. This variable diversity measure ranges from 0, where only one of the binary set membership scores is observed, to 1, where both of the possible set membership scores have an equal probability of being observed. Upon inspecting the variable diversity object, one finds that variable CG has especially homogenous values, with only 2 of 15 observations equal to 1. This fact is also shown clearly in the paths plot, where the paths following the branches where CG is equal to zero are relatively well filled in, while the branches for CG equal to one are very sparsely filled. Were a researcher looking to increase the reliability of QCA results in this case, one might consider adding addition cases where CG was equal to one.

```
# Get variable diversity
out$var.diversity
  condition diversity
1        PS    0.9333
2        DU    0.9333
3        WM    0.9333
4        LG    0.8000
5        CG    0.2667

# Inspect values of variable CG
table(hicks_20$CG)

 0  1
13  2
```

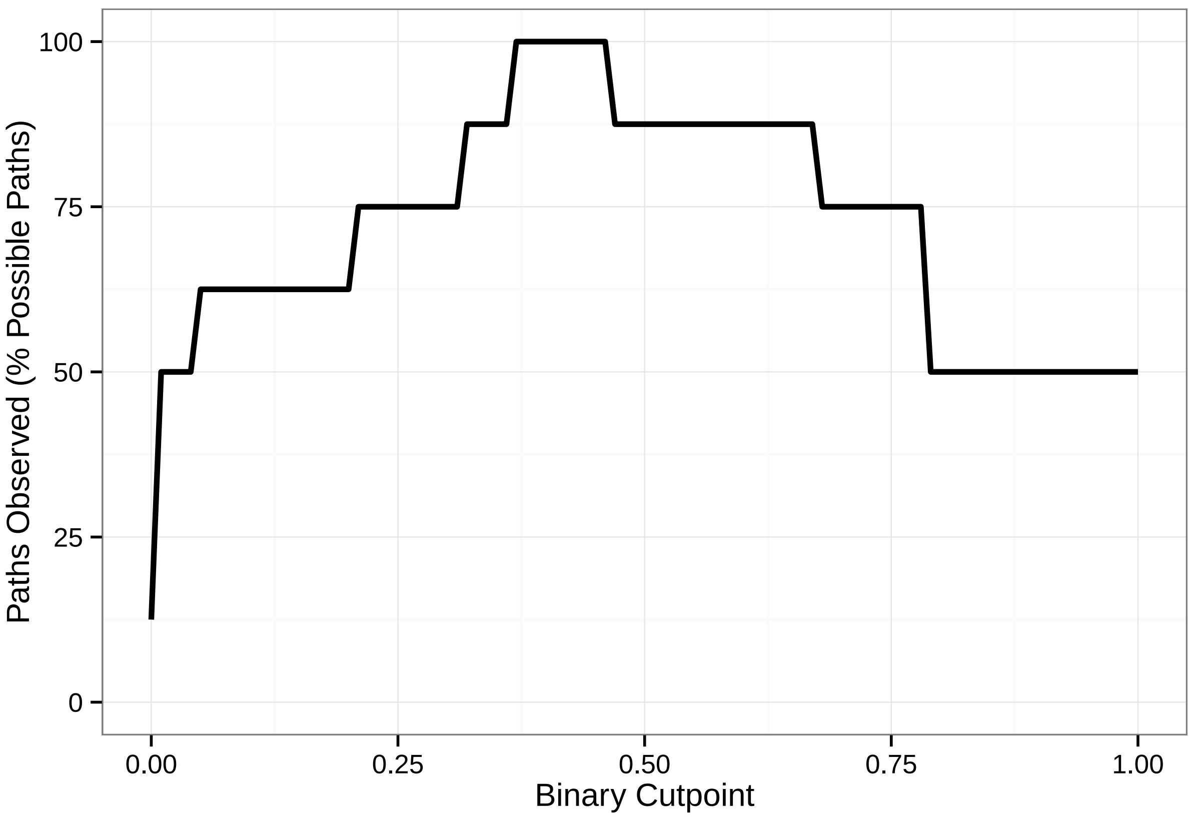
Unfortunately, carrying out such an analysis of the observed and unobserved paths to the outcome is only possible when the set membership scores in the data are binary. Where the data are fuzzy, the number of potential paths to the outcome is infinite. The only way to conduct path analysis with fuzzy set data is to include some sort of dichotomization that reduces to the number of possible causal configurations to its crisp-set maximum. This includes picking sum threshold between 0 and 1 above which all fuzzy set values will be converted to 1, and below which all values will be converted to zero.

The `paths_summary_fuzzy()` function takes a dataset of fuzzy-set membership scores and calculates the percentage of possible paths to the outcome that are observed when different dichotomizing values are used to convert fuzzy set values to crisp set values. A standard application to the `hh` dataset is shown below. Consider these case where three causal factors are included in the analysis. The number of significant digits for the dichotomizing values (set via the `sig.digits` argument) governs the number and specificity of the dichotomizing values used. For instance, setting `sig.digits=1` means that the function will dichotomize the data for the values 0.0, 0.1, 0.2 and onward up to 1.0. Similarly, setting `sig.digits=2` will cause the function to dichotomize over 0.00, 0.01, and 0.02 through 1.00.

```
# Get fuzzy paths summary
out<-paths_summary_fuzzy(data = hh, outcome = "SUCCESS", conditions = c("GDPPC", "GINI"

# Inspect fuzzy paths plot
out$fuzzy.paths.plot
```



```
# Get fuzzy paths summary, with 2 significant digits
out<-paths_summary_fuzzy(data = hh, outcome = "SUCCESS", conditions = c("GDPPC", "GINI"

# Inspect fuzzy paths plot
out$fuzzy.paths.plot
```



As the plots show clearly, changing the dicotomizing values can greatly alter the percentage of possible paths to the outcome that are empirically observed. Setting the value to between 0.37 and 0.46 means that every possible path to the outcome would be empirically observed. Setting the dichotomizing value at 0.79 or above, however, reduces number of paths obsered to just half of the total number of possible paths. With this information in hand, the researcher can re-examine his or her fuzzy set scorings to ensure that they are appropriate relative to the conceptualization of each set, and that small changes in these values will not result in large alterations to the percentage of possible paths observed.

Interfacing with New Set-Theoretic Comparative Methods

While the vast majority of set-theoretic work involves the use of QCA and its variants, a number of new approaches have been recently developed that rely on machine learning algorithms for classification. For instance, Krogslund and Michel (2014b) propose Random Forest Comparative Analysis (RFCA), which used random forests and k-means clustering of variable importance scores to recover QCA-style solutions. This method is implemented in the function `rfca()`.

As shown below, several parameters used in the method can be directly adjusted using the function's arguments, including the number of trees (`ntree`), the number of variables sampled for splitting at each node (`mtry`), along with many parameters utilized by `kmeans()`. The function can accommodate datasets with both crisp and fuzzy set membership scores, and allows the user to specify a function via which set membership values for conjunctions can be calculated. And as with `eqmcc()`, the `rfca()` function is accompanied by an options function that allow the user to run `rfca` across a variety of different values for the functional arguments.

```
# Run rfca on hicks_20
rfca(data = hicks_20, outcome = "CON", case.ids = "Case")
[1] "PS"    "WM"    "PS*WM"

# Run rfca on hh
rfca(data = hh, outcome = "SUCCESS")
 [1] "GDPPC"                          "UNEMP"
 [3] "URBAN"                          "MOBILE"
 [5] "INTERNET"                       "GDPPC*GINI"
 [7] "GDPPC*URBAN"                    "GDPPC*INTERNET"
 [9] "GDPPC*FUEL"                     "GINI*UNEMP"
[11] "GINI*YOUTH"                     "URBAN*INTERNET"
[13] "YOUTH*INTERNET"                 "MOBILE*FUEL"
[15] "FUEL*POL"                       "GDPPC*URBAN*INTERNET"
[17] "GDPPC*MOBILE*FUEL"              "GINI*YOUTH*INTERNET"
[19] "GINI*YOUTH*FUEL"                "UNEMP*YOUTH*INTERNET"
[21] "UNEMP*FUEL*POL"                 "URBAN*YOUTH*INTERNET"
[23] "YOUTH*MOBILE*INTERNET"          "YOUTH*FUEL*POL"
[25] "GDPPC*GINI*YOUTH*FUEL"          "GDPPC*GINI*YOUTH*POL"
[27] "GDPPC*GINI*FUEL*POL"            "GDPPC*YOUTH*MOBILE*INTERNET"
[29] "GDPPC*YOUTH*FUEL*POL"           "GINI*YOUTH*FUEL*POL"
[31] "UNEMP*INTERNET*FUEL*POL"        "URBAN*YOUTH*MOBILE*INTERNET"
[33] "GDPPC*GINI*YOUTH*FUEL*POL"      "GDPPC*GINI*INTERNET*FUEL*POL"
```

```
[35] "GDPPC*URBAN*YOUTH*MOBILE*INTERNET" "GINI*UNEMP*INTERNET*FUEL*POL"

# Run rfca on hh, using mean for interacted values
rfca(data = hh, outcome = "SUCCESS", inter.func = "mean")
 [1] "MOBILE*POL"                        "URBAN*YOUTH*MOBILE"
 [3] "YOUTH*MOBILE*INTERNET"             "GDPPC*UNEMP*URBAN*YOUTH"
 [5] "UNEMP*YOUTH*MOBILE*INTERNET"       "YOUTH*MOBILE*INTERNET*POL"
 [7] "YOUTH*INTERNET*FUEL*POL"           "GDPPC*URBAN*YOUTH*MOBILE*POL"
 [9] "UNEMP*URBAN*YOUTH*MOBILE*INTERNET" "URBAN*YOUTH*MOBILE*INTERNET*POL"

# Get rfca() options list
arglist<-rfca_args()

# Set ntree to 100, 500, 2000
arglist$ntree<-c(100,500,2000)

# Set mtry to 5, 10, 20
arglist$mtry<-c(5,10,20)

# Set outcome
arglist$outcome<-"CON"

# Set case.id
arglist$case.id<-"Case"

# Run rfca_options()
out<-rfca_options(data = hicks_20, rfca.options.list = arglist, verbose = F)

# Inspect options list
out$opts
  outcome ntree mtry case.id list.id
1     CON   100    5    Case       1
2     CON   500    5    Case       2
3     CON  2000    5    Case       3
4     CON   100   10    Case       4
5     CON   500   10    Case       5
6     CON  2000   10    Case       6
7     CON   100   20    Case       7
8     CON   500   20    Case       8
9     CON  2000   20    Case       9

# Inspect some results
out$results[[3]]
[1] "PS"    "WM"    "PS*WM"
```

```
out$results[[6]]
[1] "PS"    "WM"    "PS*WM"
out$results[[9]]
[1] "PS"    "WM"    "PS*WM"
```

Should one prefer to apply a different classification algorithm aside from random forests before using k-means clustering, the function `caret_infer()` provides a direct interface with the *caret* package, which is itself a standardized interface that interacts with a wide range of classification and decision tree packages written in R (Kuhn, 2008). For example, the code below shows how one can use a linear support vector machine or naive Bayes classifer to recover QCA-style solutions. For details on how to pass arguments to the `train()` functions, please refer to the *caret* documentation.

```
# Support vector machine
out<-caret_infer(data = hicks_20, case.ids = "Case", outcome = "CON", type = "c", methoc

# Inspect solutions
out$solutions
[1] "PS"    "WM"    "PS*WM"

# Naive Bayes
out<-caret_infer(data = hicks_20, case.ids = "Case", outcome = "CON", type = "c", methoc

# Inspect solutions
out$solutions
[1] "PS"    "WM"    "PS*WM"
```

References

Breiman, L. (2001). Random forests. *Machine learning 45*(1), 5–32.

Breiman, L., J. Friedman, C. J. Stone, and R. A. Olshen (1984). *Classification and regression trees*. CRC press.

Collier, D. (2014). Problematic Tools: Introduction to Symposium on Set Theory in Social Science. *Qualitative & Multi-Method Research 12*(1), 2–9.

Goertz, G. and J. Mahoney (2012). *A tale of two cultures: Qualitative and quantitative research in the social sciences*. Princeton University Press.

Hartigan, J. A. and M. A. Wong (1979). Algorithm as 136: A k-means clustering algorithm. *Applied statistics*, 100–108.

Hastie, T., R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani (2009). *The elements of statistical learning*, Volume 2. Springer.

Hicks, A., J. Misra, and T. N. Ng (1995). The programmatic emergence of the social security state. *American Sociological Review*, 329–349.

Hug, S. (2013). Qualitative Comparative Analysis: How Inductive Use and Measurement Error Lead to Problematic Inference. *Political Analysis 21*(2), 252–265.

Hussain, M. M. and P. N. Howard (2013). What best explains successful protest cascades? icts and the fuzzy causes of the arab spring. *International Studies Review 15*(1), 48–66.

Krogslund, C., D. D. Choi, and M. Poertner (2014). Fuzzy sets on shaky ground: Parameter sensitivity and confirmation bias in fsqca. *Political Analysis*.

Krogslund, C. and K. Michel (2014a). A Larger-N, Fewer Variables Problem? The Counterintuitive Sensitivity of QCA. *Qualitative & Multi-Method Research 12*(1), 25–33.

Krogslund, C. and K. E. Michel (2014b). Can QCA Uncover Causal Relationships? An Assessment and Proposed Alternative. Presented at the Annual Meeting of the American Political Science Association, Washington, D.C.

Kuhn, M. (2008, 11). Building predictive models in r using the caret package. *Journal of Statistical Software 28*(5), 1–26.

Lucas, S. R. and A. Szatrowski (2014). Qualitative Comparative Analysis in Critical Perspective. *Sociological Methodology 44*.

Mahoney, J. and G. Goertz (2006). A tale of two cultures: Contrasting quantitative and qualitative research. *Political Analysis 14*(3), 227–249.

Marx, A., B. Rihoux, and C. Ragin (2014). The origins, development, and application of qualitative comparative analysis: the first 25 years. *European Political Science Review 6*(01), 115–142.

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

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

Rihoux, B. and C. C. Ragin (Eds.) (2009). *Configurational Comparative Methods*. Thousand Oaks: Sage Publications.

Schneider, C. Q. and C. Wagemann (2012). *Set-Theoretic Methods for the Social Sciences: A Guide to Qualitative Comparative Analysis (QCA)*. Cambridge: Cambridge University Press.

Seawright, J. (2014). Limited Diversity and the Unreliability of QCA. *Sociological Methodology 44*.

Skaaning, S.-E. (2011). Assessing the Robustness of Crisp-set and Fuzzy-set QCA Results. *Social Methods Research 40*(2), 391–408.

Thiem, A. (2013). Membership Function Sensitivity of Descriptive Statistics in Fuzzy-Set Relations. *International Journal of Social Research Methodology*, 1–18.

Thiem, A. and A. Duşa (2013a). Boolean minimization in social science research: a review of current software for qualitative comparative analysis (qca). *Social Science Computer Review*.

Thiem, A. and A. Duşa (2013b). QCA: A Package for Qualitative Comparative Analysis. *The R Journal 5*(1), 87–97.