# `minBool` manual
# Minimization of Boolean function by the Quine-McCluskey method

Andrey Popov

February 2007

## 1 Why Boolean function minimization?

Here we will not go into the theory of Boolean functions, automata or finite state machines. The reader is expected to have some background knowledge in them or at the very least understanding what logical conjunction (**and**), disjunction (**or**) and negation (**not**) are.

The following short notations will be used throughout this document:

$$\begin{aligned} x \text{ and } y &\leftrightarrow xy \\ x \text{ or } y &\leftrightarrow x + y \\ \text{not}(x) &\leftrightarrow \bar{x} \end{aligned}$$

Let us start with a motivating example. Assume you have a logical function $f$ depending on 4 logical variables $a$, $b$, $c$ and $d$, as follows:

$$f = \bar{b} + \bar{a}cd + b\bar{c}d + c\bar{d} + a\bar{d} + a\bar{b}$$

This could be an activation function $f$ for some automata, that depends on state of the 4 signals. This could also be some logical expression in a programming language, that depends on the combination of 4 logical conditions. For example in C this could be

```
if (!b || (!a && c && d) || (b && c && d) || (c && !d) || (a && !d) || (a && !b))
```

The question that we want to answer not is **Can we simplify this expression?**

This can be motivated by the fact that we want to reduce the number of logical elements involved in out activation function (respectively the size and cost of the final product) or that we want to make our program faster and hopefully understandable. In the rest we will abstract ourselves from the underlying problem and will deal only with the problem posed above.

## 2 How to solve Boolean minimization problems?

There are 2 main algorithms: Karnaugh map (also known as Veitch diagram) and Quine-McCluskey method. While the former is an relative easy and intuitive it becomes difficult to use with more than 5 Boolean variables and is difficult to implement in a program. On other hand the Quine-McCluskey method is more complicated, but easier for computer implementation. Therefore `minBool.m` is an implementation of this method [1]

Coming back to the above example, there are altogether 16 combinations that can occur between them. They are listed in Table 1. In the last column with + are marked the combinations, that will set $f$ to 1 for our example. For example the condition $\bar{b}$ will be true for combinations $0 \ldots 3$ and $8 \ldots 11$.

Table 1: Activation combinations

| | $a$ | $b$ | $c$ | $d$ | used |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | + |
| 1 | 0 | 0 | 0 | 1 | + |
| 2 | 0 | 0 | 1 | 0 | + |
| 3 | 0 | 0 | 1 | 1 | + |
| 4 | 0 | 1 | 0 | 0 | |
| 5 | 0 | 1 | 0 | 1 | |
| 6 | 0 | 1 | 1 | 0 | + |
| 7 | 0 | 1 | 1 | 1 | + |
| 8 | 1 | 0 | 0 | 0 | + |
| 9 | 1 | 0 | 0 | 1 | + |
| 10 | 1 | 0 | 1 | 0 | + |
| 11 | 1 | 0 | 1 | 1 | + |
| 12 | 1 | 1 | 0 | 0 | |
| 13 | 1 | 1 | 0 | 1 | |
| 14 | 1 | 1 | 1 | 0 | + |
| 15 | 1 | 1 | 1 | 1 | + |

At the end we obtain, that $f$ will be 1 for combinations $0, 1, 2, 3, 6, 7, 8, 9, 10, 11, 14$ and $15$.

To find the minimal combination simply call `minBool.m` with the list of activating combinations:

```
>> R = minBool([0, 1, 2, 3, 6, 7, 8, 9, 10, 11, 14, 15])
R =
     1     2     3     4
     0    -1     0     0
     0     0     1     0
```

What the result is telling us is that we can exchange the initial activation for $f$ by a disjunction of 2 conjunctions. The first line of `R` is giving only the numbers of the incoming variables we have used. Each line that follows it is describing one of the conjunctions. The first line is showing us that only the 2nd signal, i.e. $b$, is involved and it has to be negated (-1 shows negation, 1 shows that the signal is involved directly). The second line is stating that the second conjunction is only of signal 3, i.e. $c$. Of the whole Boolean condition can be exchanged with

$$f = \bar{b} + c$$

or if it is a program `if (!b || c)`

Looks better, doesn't it!

## 3 What if there are "don't care" combinations?

So called "don't care" combinations can occur for example by the activation function by J-K and R-S flip-flops. The reason for their occurrence is that the flip-flop can change state depending only on 1 of the inputs and what is the signal on the other input is of no importance (see Table 2) and furthermore there are combinations that (by normal operation) will never be reached. Thus this gives freedom to chose a signal for this "free" input, that allows simplification of the activation function.

Table 2: J-K and R-S flip-flops

| State | | Flip-Flop inputs | | | |
|---|---|---|---|---|---|
| $Q(k)$ | $Q(k+1)$ | $R$ | $S$ | $J$ | $K$ |
| 0 | 0 | x | 0 | 0 | x |
| 0 | 1 | 0 | 1 | 1 | x |
| 1 | 0 | 1 | 0 | x | 1 |
| 1 | 1 | 0 | x | x | 0 |

Again for a small-scale problems Karnaugh maps can be used to chose appropriate values for those "don't care" situations, but as the number of activation signals increases so does the complexity. In such cases one can use the method proposed [2]. It uses genetic algorithm to propose values for those "free" states and than evaluates which combination is better depending on the total number of conjunctions, signals involved in the conjunctions and/or number of negations needed. The method is illustrated with the `GAmin` toolbox (see [3]), but can be used with other genetic/evolutionary algorithms as well.

As an example consider that we want to implement the second J-K flip-flop ($Q_2$) from [2] and in particular the activation function for its $J$ input (i.e. $J_2$). The state of this signal (0 or 1) is no importance by combinations $2, 3, 6, 7, 10, 11, 14$ and $15$. Thus there are total of 8 decision variables for a genetic algorithm to chose.

The genetic algorithm used in [2] is a multiobjective one, since it gives the freedom to chose the "best" solution after the algorithm has finished, instead of deciding what is more important (the number of disjunction, the number of negation, etc) before the optimization algorithm. This way also the problem of deciding of weights that allow combining several objectives to a single one is avoided.

The three objective functions used here are

$F_1$ number of conjunctions in the disjunctive function generating the $J$ or $K$ (respectively $S$ and $R$) signals

$F_2$ total number of signals involved in the activation function

$F_3$ total number of inversion

For an activation function like $J = \bar{a}bc + bd$ the values are $F_1 = 2$, $F_2 = 5$, $F_3 = 1$.

If one has different requirements or plans to use other activation logic (e.g. NOT-OR) one can design other cost functions.

The first step of the evaluation program is to substitute the decision variables into the activation function and see which combinations are used. Then those combinations are passed to `minBool.m` which evaluates the minimal function representation. Finally the values of the objective functions are computed and returned to the evolutionary algorithm.

The following Matlab code realizes the above algorithm. It is taken from the demonstration functions for [2].

```
function costFun = evalBool_J2( B )
% B is a vector with the 8 decision variables

% substitute the variables in the correct positions
J2 = [0 1 B(1:2) 0 1 B(3:4) 0 0 B(5:6) 0 1 B(7:8)];

% take only the combinations for which J2 is 1
G = find(J2)-1;                 % -1, because the counting starts at 0

% Perform boolean minimization
R = minBool(G);
```

```
R = R(2:end,:);                 % remove the heading row

% evaluate the cost functions
costFun(1) = n = size(R,1);     % number of desjunctions
costFun(2) = sum(sum(R~=0));    % number of involved signals
costFun(3) = sum(sum(R==-1));   % number of inverted signals
```

Now to run the optimization just define the number of variables (8), the fact that they are integers, taking only values of 0 and 1 and the function, computing the cost function `evalBool_J2`.

```
% Define optimization options
opt = GAopt(-5);
opt.MaxIter = 100;   opt.Graphics = 'off';

% lower and upper boundary and variable type
BNDS = [zeros(8,1) ones(8,1) -ones(8,1)];

%% Run the optimization
[rGens, rFit] = GAMOminSC('evalBool_J2', BNDS, opt);
```

The only remaining thing is interpreting the results and if there are several results obtained from the multiobjective algorithm choosing the most appropriate one.

```
for i = 1:size(rGens,1)
    B = rGens(i,:);
    fprintf('\n Proposed solution number %d\n',i);

    J2 = [0 1 B(1:2) 0 1 B(3:4) 0 0 B(5:6) 0 1 B(7:8)];
    R = minBool(find(J2));

    fprintf('\t x    Q1    Q2    Q3\n');
    disp(R(2:end,:))
end
```

Executing this gives exactly 1 activation function: $J_2 = \bar{x}Q_3 + Q_1Q_3$.

Enjoy!

# References


[1] St. Mihailov, A. Popov, Kr. Filipova, N. Kasev, "Comparative Analysis of Boolean Functions Minimization in Terms of Simplifying the Synthesis", *First International Congress of Mechanical and Electrical Engineering and Technologies*, ISBN 954-20-0215-7, MARIND 2002, 6-11 Oct., Varna, pp.273-276

[2] A. Popov, Kr. Filipova, "Genetic Algorithms synthesis of finite state machines", *27th International Spring Seminar on Electronics Technology*. IEEE Proc., Catalog N 04EX830, ISBN 0-7803-8422-9, pp.388-392, 13-16 may, 2004, Sofia, in Annual School Lectures

[3] A. Popov, "Genetic Algorithms for Optimization - Application in the regulator synthesis task", Bachelors thesis at Technical University - Sofia, faculty "Automatics", department "Systems and Control"