Encyclopedia of Algorithms
DOI 10.1007/978-3-642-27848-8_446-2

Optimal Two-Level Boolean Minimization

Robert P. Dick*
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

Keywords Logic minimization • Tabular method • Quine–McCluskey algorithm

Years and Authors of Summarized Original Work

1956; McCluskey
1955; Quine
1952; Quine

Problem Definition

Find a minimal sum-of-products expression for a Boolean function. Consider a Boolean algebra with elements *False* and *True*. A Boolean function $f(y_1, y_2, \ldots, y_n)$ of n Boolean input variables specifies, for each combination of input variable values, the function's value. It is possible to represent the same function with various expressions. For example, the first and last expressions in Fig. 1 correspond to the same function. Assuming access to complemented input variables, straightforward implementations of these expressions would require two *AND* gates and an *OR* gate for $\left(\overline{a} \wedge \overline{b}\right) \vee (\overline{a} \wedge b)$ and only a wire for $\overline{a}$. Although the implementation efficiency depends on target technology, in general terser expressions enable greater efficiency. Boolean minimization is the task of deriving the tersest expression for a function. Elegant and optimal algorithms exist for solving the variant of this problem in which the expression is limited to two levels, i.e., a layer of *AND* gates followed by a single *OR* gate or a layer of *OR* gates followed by a single *AND* gate.

Key Results

This survey will start by introducing the Karnaugh Map visualization technique, which will be used to assist in the subsequent explanation of the Quine–McCluskey algorithm for two-level Boolean minimization. This algorithm is optimal for its constrained problem variant. It is one of the fundamental algorithms in the field of computer-aided design and forms the basis or inspiration for many solutions to more general variants of the Boolean minimization problem.

*E-mail: dickrp@umich.edu

Encyclopedia of Algorithms
DOI 10.1007/978-3-642-27848-8_446-2

Expression	Meaning in English	Boolean Logic Identity
$\overline{a} \wedge \overline{b} \vee \overline{a} \wedge b$	not a and not b or not a and b	Distributivity
$\overline{a} \wedge (\overline{b} \vee b)$	not a and either not b or b	Complements
$\overline{a} \wedge \mathit{True}$	not a and True	Boundedness
$\overline{a}$	not a	

Fig. 1 Equivalent representations with different implementation complexities

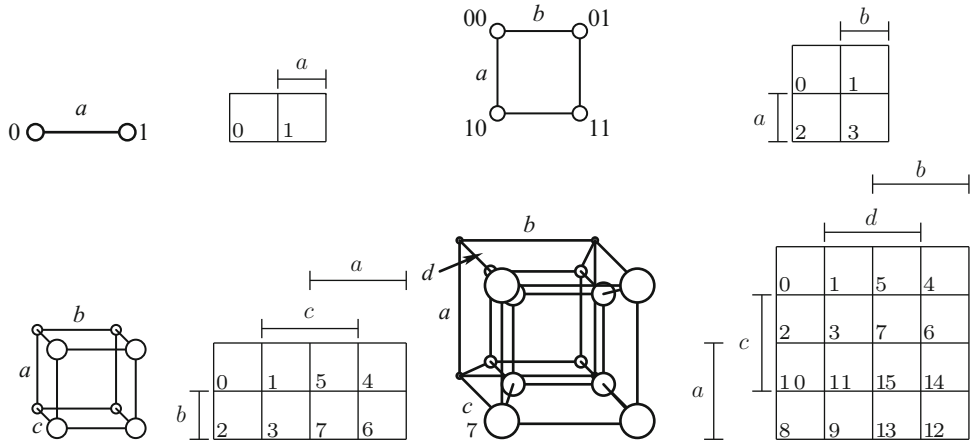

Fig. 2 Boolean function spaces from one to four dimensions and their corresponding Karnaugh Maps

Karnaugh Maps

Karnaugh Maps [4] provide a method of visualizing adjacency in Boolean space. A Karnaugh Map is a projection of an n-dimensional hypercube onto a two-dimensional surface such that adjacent points in the hypercube remain adjacent in the two-dimensional projection. Figure 2 illustrates Karnaugh Maps of 1, 2, 3, and 4 variables: a, b, c, and d.

A *literal* is a single appearance of a complemented or uncomplemented input variable in a Boolean expression. A product term or *implicant* is the Boolean product, or *AND*, of one or more literals. Every implicant corresponds to the repeated balanced bisection of Boolean space, or of the corresponding Karnaugh Map, i.e., an implicant is a rectangle in a Karnaugh Map with width m and height n where $m = 2^j$ and $n = 2^k$ for arbitrary nonnegative integers j and k, e.g., the ovals in Fig. 3(ii–v). An *elementary implicant* is an implicant in which, for each variable of the corresponding function, the variable or its complement appears, e.g., the circles in Fig. 3(ii). Implicant A *covers* implicant B if every elementary implicant in B is also in A.

Prime implicants are implicants that are not covered by any other implicants, e.g., the ovals and circle in Fig. 3(iv). It is unnecessary to consider anything but prime implicants when seeking a minimal function representation because, if non-prime implicants could be used to cover some set of elementary implicants, there is guaranteed to exist a prime implicant that covers those elementary implicants and contains fewer literals. One can draw the largest implicants covering each elementary implicant and covering no positions for which the function is *False*, thereby using Karnaugh Maps to identify prime implicants. One can then manually seek a compact subset of prime implicants covering all elementary implicants in the function.

This Karnaugh Map-based approach is effective for functions with few inputs, i.e., those with low dimensionality. However, representing and manipulating Karnaugh Maps for functions of

DOI 10.1007/978-3-642-27848-8_446-2

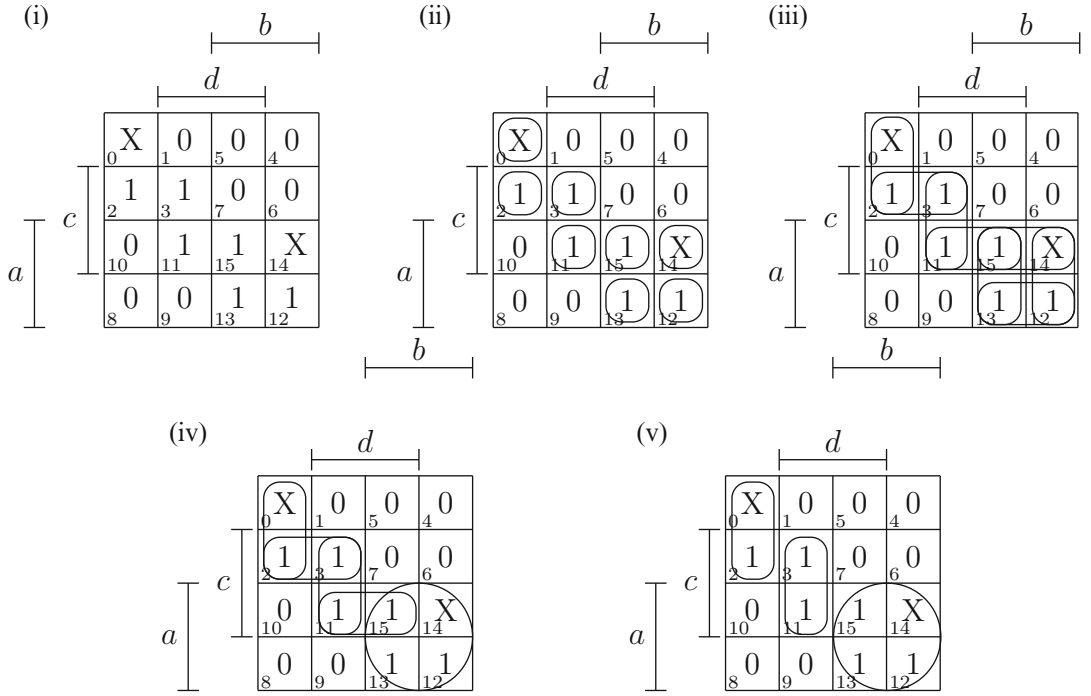

Fig. 3 (**i**) Karnaugh Map of function $f(a, b, c, d)$, (**ii**) elementary implicants, (**iii**) second-order implicants, (**iv**) prime implicants, and (**v**) a minimal cover

many variables is challenging. Moreover, the Karnaugh Map method provides no clear set of rules to follow when selecting a minimal subset of prime implicants to implement a function.

The Quine–McCluskey Algorithm

The Quine–McCluskey algorithm provides a formal, optimal way of solving the two-level Boolean minimization problem. W. V. Quine laid the essential theoretical groundwork for optimal two-level logic minimization [7, 8]. However, E. J. McCluskey first proposed a precise algorithm to fully automate the process [6]. Both are built upon the ideas of M. Karnaugh [4].

The Quine–McCluskey method has two phases: (1) produce all prime implicants and (2) select a minimal subset of prime implicants covering the function. In the first phase, the elementary implicants of a function are iteratively combined to produce implicants with fewer literals. Eventually, all prime implicants are thus produced. In the second phase, a minimal subset of prime implicants covering the on-set elementary implicants is selected using unate covering [5].

The Quine–McCluskey method may be illustrated using an example. Consider the function indicated by the Karnaugh Map in Fig. 3(i) and the truth table in Fig. 4. For each combination of Boolean input variable values, the function $f(a, b, c, d)$ is required to output a 0 (*False*), a 1 (*True*), or has no requirements. The lack of requirements is indicated with an X, or don't-care symbol.

Expanding implicants as much as possible will ultimately produce the prime implicants. To do this, combine on-set and don't-care elementary implicants using the reduction theorem ($\overline{a}b \vee ab = b$) shown in Fig. 1. The elementary implicants are circled in Fig. 3(ii) and listed in the second column of Fig. 5. In this figure, 0s indicate complemented variables, and 1s indicate

DOI 10.1007/978-3-642-27848-8_446-2

Elementary implicant (a, b, c, d)	Function value (a, b, c, d)	Elementary implicant	Function value
0000	X	1000	0
0001	0	1001	0
0010	1	1010	0
0011	1	1011	1
0100	0	1100	1
0101	0	1101	1
0110	0	1110	X
0111	0	1111	0

Fig. 4 Truth table of function $f(a, b, c, d)$

Number of ones	Elementary implicant (a, b, c, d)	Second-order implicant	Third-order implicant
0	0000 ✓	00X0	
1	0010 ✓	001X	
2	0011 ✓	X011	11XX
	1100 ✓	110X ✓	
		11X0 ✓	
3	1011 ✓	1X11	
	1101 ✓	11X1 ✓	
	1110 ✓	111X ✓	
4	1111 ✓		

Fig. 5 Identifying prime implicants

uncomplemented variables, e.g., 0010 corresponds to $\overline{a}\overline{b}c\overline{d}$. It is necessary to determine all possible combinations of implicants. It is impossible to combine nonadjacent implicants, i.e., those that differ in more than one variable. Therefore, it is not necessary to consider combining any pair of implicants with a number of uncomplemented variables differing by any value other than 1. This fact can be exploited by organizing the implicants based on the number of ones they contain, as indicated by the first column in Fig. 5. All possible combinations of implicants in adjacent subsets are considered. For example, consider combining 0010 with 0011, which results in 001X or $\overline{a}\overline{b}c$, and also consider combining 0010 with 1100, which is impossible due to differences in more than one variable. Whenever an implicant is successfully merged, it is marked. These marked implicants are clearly not prime implicants because the implicants they produced cover them and contain fewer literals. Note that marked implicants should still be used for subsequent combinations. The merged implicants in the third column of Fig. 5 correspond to those depicted in Fig. 3(iii).

After all combinations of elementary implicants have been considered, and successful combinations listed in the third column, this process is repeated on the second-order merged implicants in the third column, producing the implicants in the fourth column. Implicants that contain don't-care marks in different locations may not be combined. This process is repeated until a column yielding no combinations is arrived at. The unmarked implicants in Fig. 5 are the prime implicants, which correspond to the implicants depicted in Fig. 3(iv).

DOI 10.1007/978-3-642-27848-8_446-2

Requirements (elementary implicants)	Resources (prime implicants)				
	00X0	001X	X011	1X11	11XX
0010	✓	✓			
0011		✓	✓		
1011			✓	✓	
1100					✓
1101					✓
1111				✓	✓

Fig. 6 Solving unate covering problem to select minimal cover

After a function's prime implicants have been identified, it is necessary to select a minimal subset that covers the function. The problem can be formulated as unate covering. As shown in Fig. 6, label each column of a table with a prime implicant; these are resources that may be used to fulfill the requirements of the function. Label each row with an elementary implicant from the on-set; these rows correspond to requirements. Do not add rows for don't cares. Don't cares impose no requirements, although they were useful in simplifying prime implicants. Mark each row–column intersection for which the elementary implicant corresponding to the row is covered by the prime implicant corresponding to the column. If a column is selected, all the rows for which the column contains marks are *covered*, i.e., those requirements are satisfied. The goal is to cover all rows with a minimal-cost subset of columns. McCluskey defined minimal cost as having a minimal number of prime implicants, with ties broken by selecting the prime implicants containing the fewest literals. The most appropriate cost function depends on the implementation technology. One can also use a similar formulation with other cost functions, e.g., minimize the total number of literals by labeling each column with a cost corresponding to the number of literals in the corresponding prime implicant.

One can use a number of heuristics to accelerate solution of the unate covering problem, e.g., neglect rows that have a superset of the marks of any other row, for they will be implicitly covered and neglect columns that have a subset of the marks of any other column if their costs are as high, for the other column is at least as useful. One can easily select columns as long as there exists a row with only one mark because the marked column is required for a valid solution. However, there exist problem instances in which each row contains multiple two marks. In the worst case, the best existing algorithms are required to make tentative decisions, determine the consequences, and then backtrack and evaluate alternative decisions.

The unate covering problem appears in many applications. It is $\mathcal{NP}$-complete [5], even for the instances arising during two-level minimization [9]. Its use in the Quine–McCluskey method predates its categorization as an $\mathcal{NP}$-complete problem by 16 years. A detailed treatment of this problem would go well beyond the scope of this entry. However, Gimpel [3] as well as Coudert and Madre [2] provide good starting points for further reading.

Some families of logic functions have optimal two-level representations that grow in size exponentially in the number of inputs, but have more compact multilevel implementations. These families are frequently encountered in arithmetic, e.g., a function indicating whether the number of on inputs is odd. Efficient implementation of such functions requires manual design or multilevel minimization [1].

DOI 10.1007/978-3-642-27848-8_446-2

Applications

Digital computers are composed of precisely two things: (1) implementations of Boolean logic functions and (2) memory elements. The Quine–McCluskey method is used to permit efficient implementation of Boolean logic functions in a wide range of digital logic devices, including computers. The Quine–McCluskey method served as a starting point or inspiration for most currently used logic minimization algorithms. Its direct use is contradicted when functions are not amenable to efficient two-level implementation, e.g., many arithmetic functions.

Cross-References

▸ Greedy Set-Cover Algorithm

Recommended Reading

1. Brayton RK, Hachtel GD, Sangiovanni-Vincentelli AL (1990) Multilevel logic synthesis. Proc IEEE 78(2):264–300
2. Coudert O, Madre JC (1995) New ideas for solving covering problems. In: Proceedings of the design automation conference, San Francisco, pp 641–646
3. Gimpel JF (1965) A reduction technique for prime implicant tables. IEEE Trans Electron Comput 14(4):535–541
4. Karnaugh M (1953) The map method for synthesis of combinational logic circuits. Trans AIEE Commun Electron 72:593–599
5. Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (eds) Complexity of computer computations. Plenum Press, New York, pp 85–103
6. McCluskey EJ (1956) Minimization of Boolean functions. Bell Syst Tech J 35(6):1417–1444
7. Quine WV (1952) The problem of simplyfying truth functions. Am Math Mon 59(8):521–531
8. Quine WV (1955) A way to simplify truth functions. Am Math Mon 62(9):627–631
9. Umans C, Villa T, Sangiovanni-Vincentelli AL (2006) Complexity of two-level logic minimization. IEEE Trans Comput-Aided Des Integr Circuits Syst 25(7):1230–1246