

mation and Management (AAIM), Portland, OR, USA, Lecture Notes in Computer Science, vol. 4508, pp. 399–410. Springer (2007)

12. Jiang, M.: A PTAS for the weighted 2-interval pattern problem over the preceding-and-crossing model. In: Y.X. A.W.M. Dress, B. Zhu (eds.) Proc. 1st Annual International Conference on Combinatorial Optimization and Applications (COCO), Xi'an, China, Lecture Notes in Computer Science, vol. 4616, pp. 378–387. Springer (2007)
13. Joseph, D., Meidanis, J., Tiwari, P.: Determining DNA sequence similarity using maximum independent set algorithms for interval graphs. In: Proc. 3rd Scandinavian Workshop on Algorithm Theory (SWAT). Lecture Notes in Computer Science, pp. 326–337. Springer, Berlin (1992)
14. Lyngsø, R., Pedersen, C.: RNA pseudoknot prediction in energy-based models. *J. Comput. Biol.* **7**, 409–427 (2000)
15. Micali, S., Vazirani, V.: An  $O(\sqrt{|V||E|})$  algorithm for finding maximum matching in general graphs. In: Proc. 21st Annual Symposium on Foundation of Computer Science (FOCS), IEEE, 1980, pp. 17–27
16. Nussinov, R., Pieczenik, G., Griggs, J., Kleitman, D.: Algorithms for loop matchings. *SIAM J. Appl. Math.* **35**, 68–82 (1978)
17. Ren, J., Rastegart, B., Condon, A., Hoos, H.: HotKnots: Heuristic prediction of rna secondary structure including pseudoknots. *RNA* **11**, 1194–1504 (2005)
18. Rivas, E., Eddy, S.: A dynamic programming algorithm for RNA structure prediction including pseudoknots. *J. Mol. Biol.* **285**, 2053–2068 (1999)
19. Ruan, J., Stormo, G., Zhang, W.: An iterated loop matching approach to the prediction of RNA secondary structures with pseudoknots. *Bioinformatics* **20**, 58–66 (2004)
20. Vialette, S.: On the computational complexity of 2-interval pattern matching. *Theor. Comput. Sci.* **312**, 223–249 (2004)
21. Zhao, J., Malmberg, R., Cai, L.: Rapid ab initio rna folding including pseudoknots via graph tree decomposition. In: Proc. Workshop on Algorithms in Bioinformatics. Lecture Notes in Computer Science, vol. 4175, pp. 262–273. Springer, Berlin (2006)

## Two-Level Boolean Minimization

1956; McCluskey

ROBERT DICK  
Department Electrical Engineering and Computer  
Systems, Northwestern University, Evanston, IL, USA

### Keywords and Synonyms

Logic minimization; Quine–McCluskey algorithm; Tabular method

### Problem Definition

#### Summary

Find a minimal sum-of-products expression for a Boolean function.

### Two-Level Boolean Minimization, Table 1

Equivalent representations with different implementation complexities

Expression	Meaning in english	Boolean logic identity
$\bar{a} \wedge \bar{b} \vee \bar{a} \wedge b$	not $a$ and not $b$ or not $a$ and $b$	Distributivity Complements Boundedness
$\bar{a} \wedge (\bar{b} \vee b)$	not $a$ and either not $b$ or $b$	
$\bar{a} \wedge True$	not $a$ and $True$	
$\bar{a}$	not $a$	

### Extended Definition

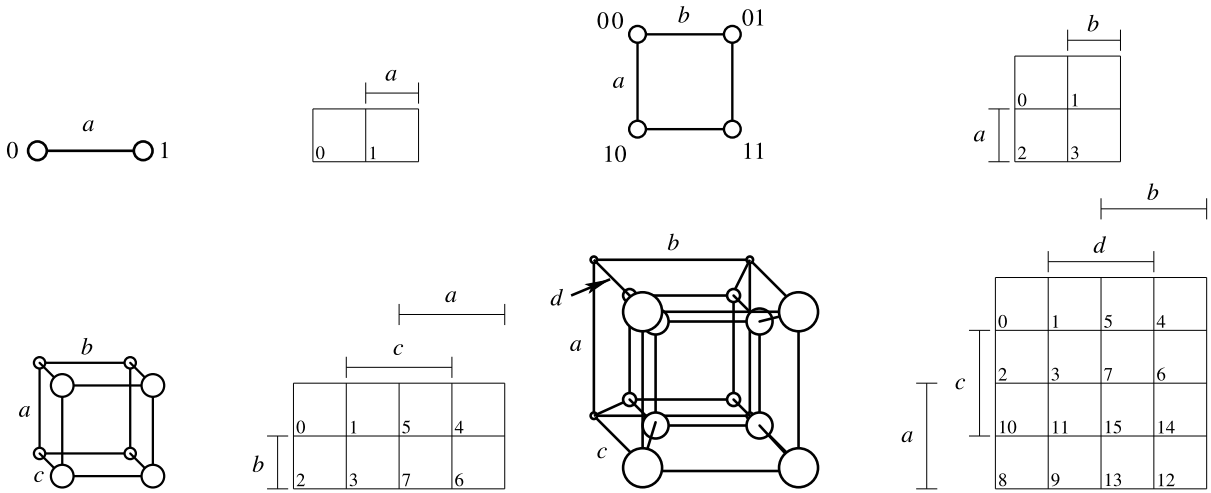
Consider a Boolean algebra with two elements: *False* or *True*. A Boolean function  $f(y_1, y_2, \dots, y_n)$  of  $n$  Boolean input variables specifies an output value for each combination of input variable values. It is possible to represent the same function with a number of different expressions. For example, the first and last expressions in Table 1 correspond to this function. Assuming access to complemented input variables, straight-forward implementations of these expressions would require two *and* gates and an *or* gate for  $\bar{a} \wedge \bar{b} \vee \bar{a} \wedge b$  and only a wire for  $\bar{a}$ . 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

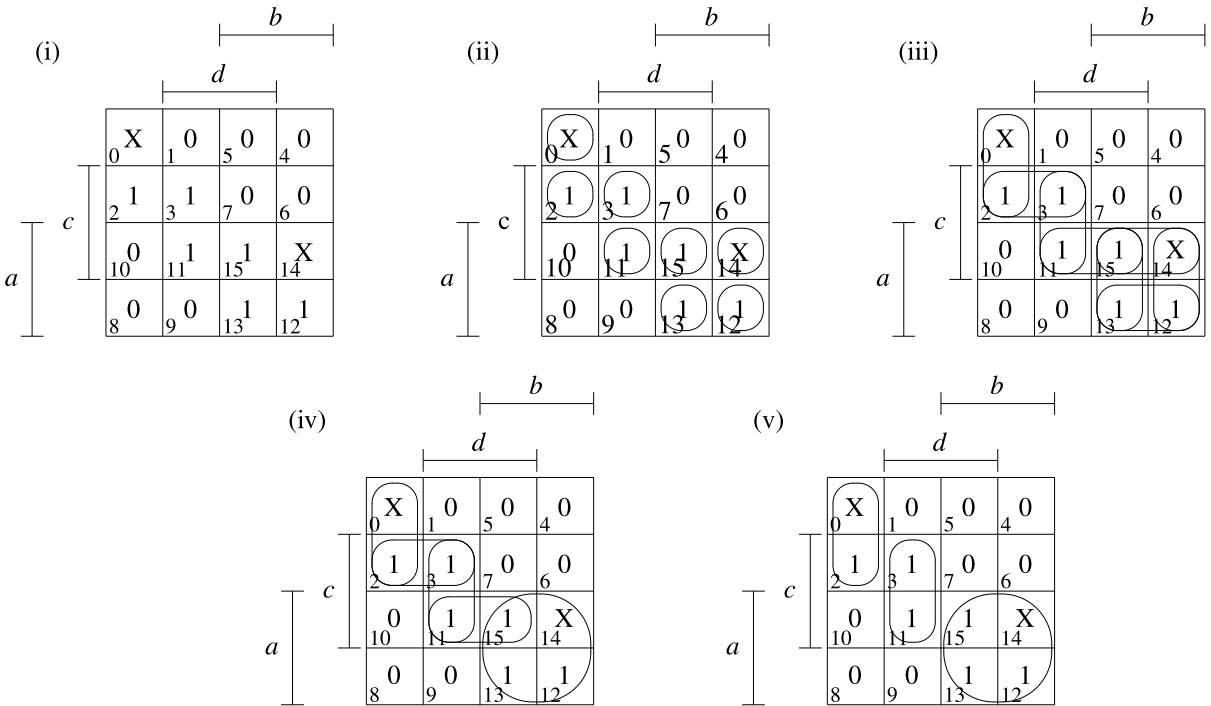
This survey will start by introducing the Karnaugh Map visualization technique, which will be used to assist in the subsequent explanation of the Quine–McCluskey algorithm for two-level Boolean minimization. This algorithm is optimal for its constrained problem variant. It is one of the fundamental algorithms 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.

### 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 two-dimensional surface such that adjacent points in the hypercube remain adjacent in the two-dimensional projection. Figure 1 illustrates Karnaugh Maps of 1, 2, 3, and 4 variables:  $a$ ,  $b$ ,  $c$ , and  $d$ .



**Two-Level Boolean Minimization, Figure 1**  
 Boolean function spaces from one to four dimensions and their corresponding Karnaugh Maps



**Two-Level Boolean Minimization, Figure 2**  
 (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

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 a 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 non-negative integers  $j$  and  $k$ , e.g. the ovals in Fig. 2(ii–v). An *elementary implicant* is an implicant in which, for each variable of the cor-

responding function, the variable or its complement appears, e. g., the circles in Fig. 2(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. 2(iv). It is unnecessary to consider anything but prime implicants when seeking a minimal function representation because, if a 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 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].

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.

The Quine–McCluskey method may be illustrated using an example. Consider the function indicated by the Karnaugh Map in Fig. 2(i) and the truth table in Table 2. 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 requirement. The lack of a requirement 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 re-

Two-Level Boolean Minimization, Table 2

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

Elementary implicant ( <i>a, b, c, d</i> )	Function value ( <i>a, b, c, d</i> )	Elementary implicant ( <i>a, b, c, d</i> )	Function value ( <i>a, b, c, d</i> )
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	1

Two-Level Boolean Minimization, Table 3

Identifying prime implicants

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

duction theorem ( $\bar{a}b \vee ab = b$ ) shown in Table 1. The elementary implicants are circled in Fig. 2(ii) and listed in the second column of Table 3. In this table, 0s indicate complemented variables and 1s indicate uncomplemented variables, e. g., 0010 corresponds to  $\bar{a}\bar{b}c\bar{d}$ . It is necessary to determine all possible combinations of implicants. It is impossible to combine non-adjacent 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 Table 3. All possible combinations of implicants in adjacent subsets are considered. For example, consider combining 0010 with 0011, which results in 001X or  $\bar{a}\bar{b}c$ , 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

**Two-Level Boolean Minimization, Table 4**  
Solving unate covering problem to select minimal cover

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

the third column of Table 3 correspond to those depicted in Fig. 2(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 Table 3 are the prime implicants, which correspond to the implicants depicted in Fig. 2(iv).

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 Table 4, 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 marks. In the worst case, the best existing algorithms are required to make tentative decisions, determine the consequences, 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 multi-level 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].

## 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

- ▶ [Local Approximation of Covering and Packing Problems](#)

## Recommended Reading

1. Brayton, R.K., Hachtel, G.D., Sangiovanni-Vincentelli, A.L.: Multi-level logic synthesis. Proc. IEEE **78**(2), 264–300 (1990)
2. Coudert, O., Madre, J.C.: New ideas for solving covering problems. In: Proc. Design Automation Conf., 1995, pp. 641–646

3. Gimpel, J.F.: A reduction technique for prime implicant tables. *IEEE Trans. Electron. Comput.* **14**(4), 535–541 (1965)
4. Karnaugh, M.: The map method for synthesis of combinational logic circuits. *Trans. AIEE, Commun. Electron.* **72**, 593–599 (1953)
5. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) *Complexity of Computer Computations*, pp. 85–103. Plenum Press, New York (1972)
6. McCluskey, E.J.: Minimization of Boolean functions. *Bell Syst. Tech. J.* **35**(6), 1417–1444 (1956)
7. Quine, W.V.: The problem of simplifying truth functions. *Am. Math. Mon.* **59**(8), 521–531 (1952)
8. Quine, W.V.: A way to simplify truth functions. *Am. Math. Mon.* **62**(9), 627–631 (1955)
9. Umans, C., Villa, T., Sangiovanni-Vincentelli, A.L.: Complexity of two-level logic minimization. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **25**(7), 1230–1246 (2006)

---

## Two-Person Game

- ▶ Complexity of Bimatrix Nash Equilibria

---

## Two-Player Game

- ▶ Complexity of Bimatrix Nash Equilibria

---

## Two-Player Nash

- ▶ Complexity of Bimatrix Nash Equilibria