Int. J. Appl. Math. Comput. Sci., 2003, Vol. 13, No. 4, 577–583

WWW-BASED BOOLEAN FUNCTION MINIMIZATION

SEBASTIAN P. TOMASZEWSKI*, ILGAZ U. CELIK**
GEORGE E. ANTONIOU ***

* BAE SYSTEMS Controls
600 Main Street, Johnston City, NY 13790, USA
e-mail: sebastian.tomaszewski@baesystems.com

** 121 Communicatins Inc., 857 Avenue of the Americas
Suite 1800, New York, NY 10001, USA
e-mail: ucelik@web121.com

*** Image Processing and Systems Laboratory
Department of Computer Science, Montclair State University
Upper Montclair NJ 07043, USA
e-mail: george.antoniou@montclair.edu

In this paper a Boolean minimization algorithm is considered and implemented as an applet in Java. The application is based on the Quine-McCluskey simplification technique with some modifications. The given application can be accessed on line since it is posted on the World Wide Web (WWW), with up to four variables, at the URL http://www.csam.montclair.edu/∼antoniou/bs. After extensive testing, the performance of the algorithm has been found to be excellent. The proposed application is a useful aid for students and professors in the fields of electrical and computer engineering and computer science as well as a valuable tool for digital designers.

Keywords: digital logic, logic design, Boolean functions, Boolean minimization, Quine-McCluskey metod

1. Introduction

The modified Quine-McCluskey (M Q-M) method is a very simple and systematic technique for minimizing Boolean functions. Why do we want to minimize a Boolean expression? By simplifying the logic function we can reduce the original number of digital components (gates) required to implement digital circuits. Therefore by reducing the number of gates, the chip size and the cost will be reduced, and the speed will be increased.

Logic minimization uses a variety of techniques to obtain the simplest gate-level implementation of a logic function. The heart of digital logic design is the Boolean algebra (Boole, 1954). A few dacades later C.E. Shannon showed how the Boolean algebra can be used in the design of digital circuits (Shannon, 1938). Using Boolean laws it is possible to minimize digital logic circuits (Huntington, 1904). Since minimization with the use of Boolean laws is not systematic nor suitable for computer implementation, a number of algorithms were proposed in order to overcome the implementation issue. Karnaugh proposed a technique for simplifying Boolean expressions using an elegant visual technique, which is actually a modified truth table intended to allow minimal sum-of-products (SOP) and product-of-sums (POS) expressions to be obtained (Karnaugh, 1953). The Karnaugh Map (K-Map) based technique breaks down beyond six variables. Quine and McCluskey proposed an algorithmic-based technique for simplifying Boolean logic functions (McCluskey, 1956; Quine, 1952). The Quine-McCluskey (Q-M) method is a computer-based technique for simplification and has mainly two advantages over the K-Map method. Firstly, it is systematic for producing a minimal function that is less dependent on visual patterns. Secondly, it is a viable scheme for handling a large number of variables. A number of methods have been developed that can generate optimal solutions directly at the expense of additional computation time. Another algorithm was reported by Petrick (1959). This algorithm uses an algebraic approach to generate all possible covers of a function. A popular tool for simplifying Boolean expressions is the Espresso, but it is not guaranteed to find the best two-level expression (Katz, 1994).

In this paper an Internet based implementation is proposed for simplifying two to four-variable Boolean functions, using a Modified Quine-McCluskey (M Q-M) method. The M Q-M technique is implemented as an applet in Java,

and can be accessed on line, up to four variables, since it is posted on the World Wide Web. Due to the algorithmic nature of the technique, the proposed method and its implementation can easily be expanded to cover more than four variables.

2. Modified Quine-McCluskey Algorithm

The main difference between the proposed algorithm and the Q-M method starts when the Q-M method groups the elements according to the number of ones in each element, but in the proposed algorithm grouping is not required. In the following steps the M Q-M follows Q-M up to the first step of the prime implicant table, which is identifying the essential prime implicants. For the next step, the Q-M uses several different techniques to eliminate the implicants efficiently. The M Q-M method simulates the elimination process of minterms and finally, when the most efficient combination is reached, it is taken out from the table.

The M Q-M algorithm is presented using the following step-by-step approach:

I. Input:
- 1.1 Enter the input of a Boolean expression either into the K-map, Truth Table, or as a Boolean expression.
- 1.2 Obtain the binary representation of each term from the input data.

II. Calculations:
- 2.1 Compare each of the terms one with another in order to find the terms that are logically adjacent. The following rules have to be followed when combining the terms:
  - (a) Combine two terms only if they differ by only one bit.
  - (b) Once there are two terms that differ by one bit, create the new term with the same exact bits or characters, except replacing the bit that is different from the '–' symbol in both of those terms.
  - (c) Once the new term is created, mark both of the old terms, indicating that both of the terms are combined.
- 2.2 Swap all of the combined terms (new terms) and terms that were not combined at all.
- 2.3 Repeat Steps 2.1 and 2.2 until it is impossible to combine the terms.

III. Table:
- 3.1 Make sure that there is only one term alike. That is, get rid of a term if it is a duplicate of another term in the content.
- 3.2 Create a prime implicant chart.
- 3.3 Identify the essential prime implicants and consider them as the first terms which will make up the result. After each implicant is put into the result term area, the implicant chart should be updated.
- 3.4 If there are any more minterms left over, proceed as follows:
  - (a) Look into the prime implicant chart for the implicants which have exactly the same minterms and eliminate the one that is least efficient.
  - (b) Try selecting out one of the terms and see whether or not the term cancels out all of the implicants.
  - (c) If it cancels out all of the implicants, put the term back into the result term area.
  - (d) If it does not cancel out all of the implicants, repeat Step (b), with a higher combination of the terms to be taken out.

IV. Display:
Display the values out of the result term area.

3. Implementation

The algorithm was first implemented using simple Java. Then it was converted into the object oriented language. To make sure that the conversion was correct, the program was retested with the same method like previously. Currently the structure of this program is shown in Fig. 1.

In the following section, two step-by-step examples are given illustrating the proposed technique.

4. Examples

Example 1. Simplify the following Boolean function:

$$F = abcD + aBcD + aBCD + aBCd + ABcd + ABcD + ABCD + AbCD. \quad (1)$$

Applying the algorithm, we have the following results:

I. Input:
- 1.1 Input the expression in either way as shown in Fig. 2.
- 1.2 In order to obtain a binary representation of the terms, you will have to know that lower case letters such as a, b, c, and d are negative variables. Thus a is (not A) or a binary zero.
  In our example, the binary representation of the terms is as follows:

0001 (1)	1100 (C)
0101 (5)	1101 (D)
0111 (7)	1111 (F)
0110 (6)	1011 (B)

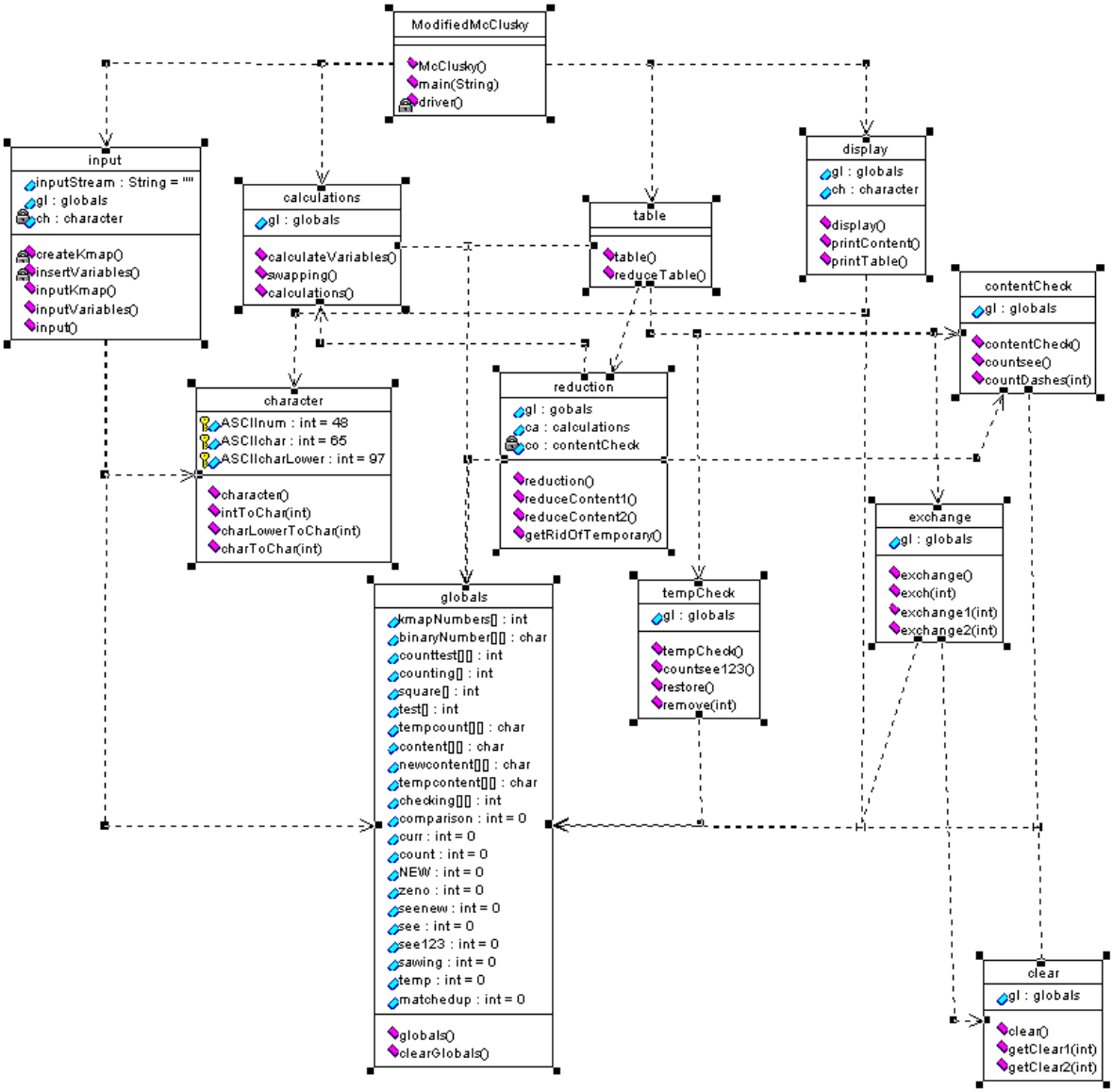

Fig. 1. Structure of the program.

II. Calculations:

2.1 and 2.2. According to the rules of combining the terms, the result of combining is as follows:

Old Terms	New Terms	Swapped Terms
X 0001 (1)	0–01 (1,5)	
X 0101 (5)	01–1 (5,7)	
X 0111 (7)	–101 (5,D)	
X 0110 (6)	011– (7,6)	
X 1100 (C)	–111 (7,F)	
X 1101 (D)	110– (C,D)	
X 1111 (F)	11–1 (D,F)	
X 1011 (B)	1–11 (F,B)	

As it is seen, the result of combining the terms creates only eight new terms and there are no swapped terms since the old terms were combined.

2.3 Repeat Steps 2.1 and 2.2:

Old Terms	New Terms	Swapped Terms
0–01 (1,5)	–1–1 (5,7,D,F)	–1–1 (5,7,D,F)
X 01–1 (5,7)	–1–1 (5,D,7,F)	–1–1 (5,D,7,F)
X –101 (5,D)		0–01 (1,5)
011– (7,6)		011– (7,6)
X –111 (7,F)		110– (C,D)
110– (C,D)		1–11 (F,B)
X 11–1 (D,F)		
1–11 (F,B)		

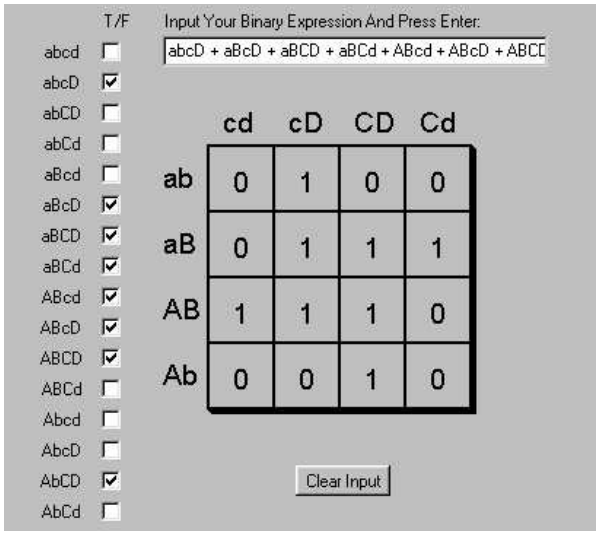

Fig. 2. Boolean function input.

The result of combining the terms creates two new terms, but after swapping there are six terms altogether. The reason behind this is that there were four terms among the old ones that were not combined with any of the other terms. That is the reason why 'X' does not mark all of the terms.

Since the swapped terms cannot be combined any further, Steps 2.1 and 2.2 are not repeated again.

III. Table:

3.1 By combining the terms and swapping, the following terms are present:

–1–1 (5,7,D,F)
–1–1 (5,D,7,F)
0–01 (1,5)
011– (7,6)
110– (C,D)
1–11 (F,B)

After getting rid of duplicates, the following terms are present:

–1–1 (5,D,7,F)
0–01 (1,5)
011– (7,6)
110– (C,D)
1–11 (F,B)

3.2 The prime implicant chart (cf. Table 1) is created to indicate what terms given at the beginning were combined to create the resulting terms or the minterms. For example, the term –1–1 was combined with four terms, which explains why there is an 'X' under (5), (7), (D), and (F).

Table 1. Prime implicant chart.

	1	5	7	6	C	D	F	B
–1–1		X	X			X	X	
0–01	X	X						
011–			X	X				
110–					X	X		
1–11							X	X

3.3 The next step is to find the essential prime implicants. These are prime implicants that cover the minterms which are not covered by other prime implicants. In our example, the essential prime implicants are 0–01, 011–, 110–, and 1–11.

3.4 After the previous step, there are no minterms that are not covered.

IV. Display:

Initially, the given Boolean expression had the form

$$0001+0101+0111+0110+1100+1101+1111+1011,$$

which is equivalent to (1).

After the application of the procedure, the following simplified expression is derived:

$$0\text{–}01 \;+\; 011\text{–} \;+\; 110\text{–} \;+\; 1\text{–}11,$$

or

$$F' = acD + aBC + ABc + ACD. \qquad (2)$$

The expression (2) is the optimal solution for the given Boolean function (1).

It is noted that the same result (2) is given in Fig. 3, using our on-line implementation. ♦

Example 2. Simplify the following Boolean function:

$$F = abcd + abcD + abCD + aBcd + aBCD + ABcd + ABcD + ABCD \qquad (3)$$

Applying the algorithm, we have

I. Input:

1.3 Input the expression in either way as shown in Fig. 4.

1.4 In our example the binary representation of the terms is as follows:

0000 (0)
0001 (1)
0011 (3)
0100 (4)
0111 (7)
1100 (C)
1101 (D)
1111 (F)

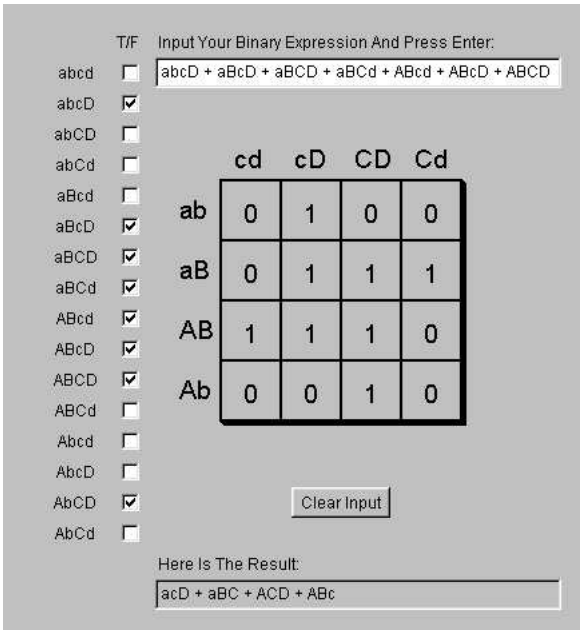

Fig. 3. Boolean function result.

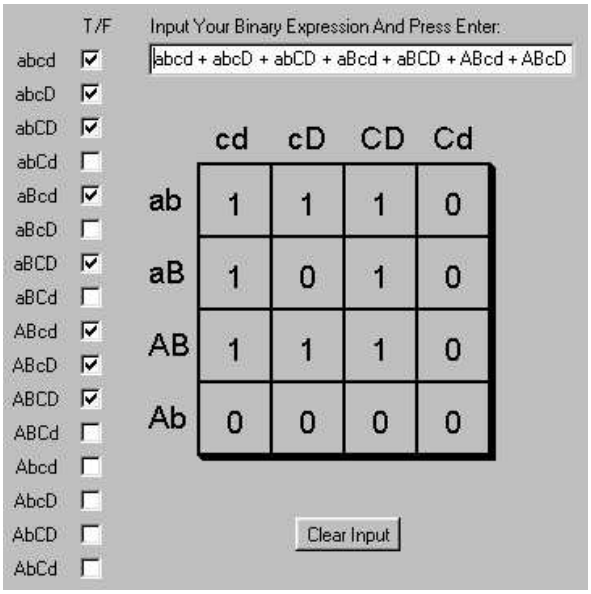

Fig. 4. Boolean function input.

II. Calculations:

2.1 and 2.2

Old Terms	New Terms	Swapped Terms
X 0000 (0)	000– (0,1)	
X 0001 (1)	0–00 (0,4)	
X 0011 (3)	00–1 (1,3)	
X 0100 (4)	0–11 (3,7)	
X 0111 (7)	–100 (4,C)	
X 1100 (C)	–111 (7,F)	
X 1101 (D)	110– (C,D)	
X 1111 (F)	11–1 (D,F)	

As can be seen, the result of combining the terms creates only eight new terms and there are no swapped terms since the old terms were combined.

2.3 Since the swapped terms cannot be combined any further, Steps 2.1 and 2.2 are not repeated.

III. Table:

3.1 As a result of combining the terms, the following terms are present:

000– (0,1)
0–00 (0,4)
00–1 (1,3)
0–11 (3,7)
–100 (4,C)
–111 (7,F)
110– (C,D)
11–1 (D,F)

Since there is no other repeating term, we can skip this part of the method.

3.2 The respective prime implicant chart is given in Table 2.

Table 2. Prime implicant chart.

	0	1	3	4	7	C	D	F
000–	X	X						
0–00	X			X				
00–1		X	X					
0–11			X		X			
–100				X		X		
–111					X			X
110–						X	X	
11–1							X	X

3.3 The next step is to find the essential prime implicants. In our example there are no essential prime implicants, since by looking at Table 2 there are no prime implicants that cover the minterms which are not covered by other prime implicants.

3.4

(a) Looking at the prime implicant chart, we deduce that there are no implicants which have the same exact minterms, and therefore none of the terms will be eliminated. The next step is to try selecting the terms which will cancel out all of the terms, since trying to select one, two or three term(s) will not cancel out all of the terms. Therefore Steps (b), (c), and (d) of the algorithm

will be repeated until it reaches the four terms that will cancel out all of the terms.

IV. Display:
Initially, the given Boolean expression had the form

$$0000+0001+0011+0100+0111+1100+1101+1111$$

which is equivalent to (3).

After the application of the procedure, the following simplified expression is derived:

$$000- \ + \ 0\text{–}11 \ + \ \text{–}100 \ + \ 11\text{–}1,$$

or

$$F' = abc + aCD + Bcd + ABD. \qquad (4)$$

The above expression (4) is the optimal solution for the given Boolean function (3).

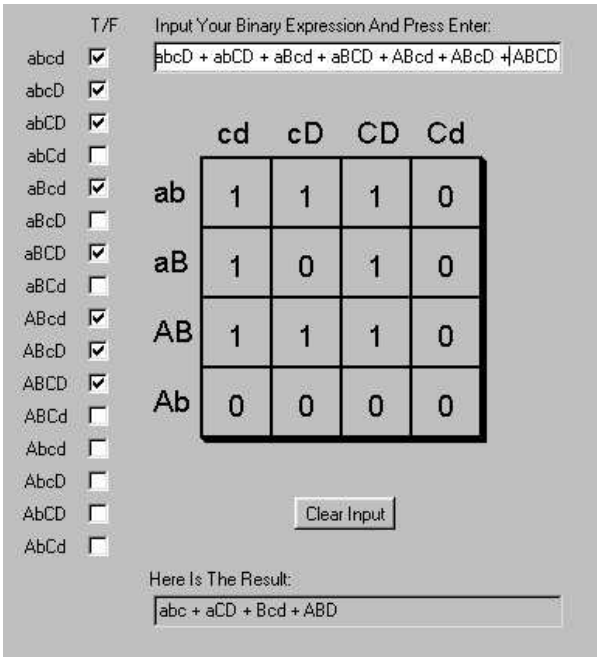

Fig. 5. Boolean function result.

Note that the same result (4) is given in Fig. 5 using our on-line implementation.

5. Algorithm Performance

The computational complexity for the Quine-McCluskey method is $O(N^{\log_2^3 \log_2 N})$, with input length $N = 2^n$ (Wegener, 1987). Mileto and Putzolu investigated the average running time of the algorithm for randomly chosen Boolean functions (Mileto and Putzolu, 1964; 1965). The running time according to variables and the number of ones present is illustrated in Fig. 6.

The performance was measured in milliseconds for all of 65 535 possibilities using Java Object Oriented programming. In simulations a SUN Ultra Enterprise 450 Server with dual modular UltraSPARC II CPU's was used, with Java v. 1.3.0.

It should be emphasized that the algorithm was tested correctly for all 65 535 possibilities, using four-variable combinations in conjunction with the K-map based minimization algorithm for Palm-based personal assistant devices (Bitincka and Antoniou, 2001).

6. Conclusion

In this paper, a new modified Quine-McCluskey algorithm for minimizing Boolean expressions has been proposed and implemented. The application which was implemented in Java using up to four variables, can be accessed on the Internet. The results of this paper can easily be extended to cover more than four variables. This application can be a great aid for students and professors in digital logic design courses and a valuable tool for the digital logic designers. The proposed application can be accessed on-line at `http://www.csam.montclair.edu/~antoniou/bs`.

Acknowledgements

The authors would like to thank Professor Carl Bredlau of the Department of Computer Science, Montclair State University, for his valuable advice on JAVA programming. Also we would like to thank the anonymous reviewers for their valuable comments.

References

Bitincka L. and Antoniou G.E. (2001): *Boolean function simplification on a Palm-based environment*. — Proc. Int. Conf. *Computing and Information Technologies*, Montclair, NJ, pp. 221–228.

Boole G. (1954): *An Investigation of the Laws of Thought*. — New York: Dover Publications.

Huntington E.V. (1904): *Sets of independence postulates for the algebra of logic*. — Trans. Amer. Math. Soc., Vol. 5, No. 3, pp. 288–309.

Karnaugh M. (1953): *The map method for synthesis of combinatorial logic circuits*. — Trans. AIEE Comm. Electron., Vol. 72, No. 4, pp. 593–598.

Katz R.H. (1994): *Contemporary Logic Design*. — Redwood City, CA: Benjamin/Cummings.

McCluskey E.J. (1956): *Minimization of Boolean functions*. — Bell System Tech. J., Vol. 35, No. 5, pp. 1417–1444.

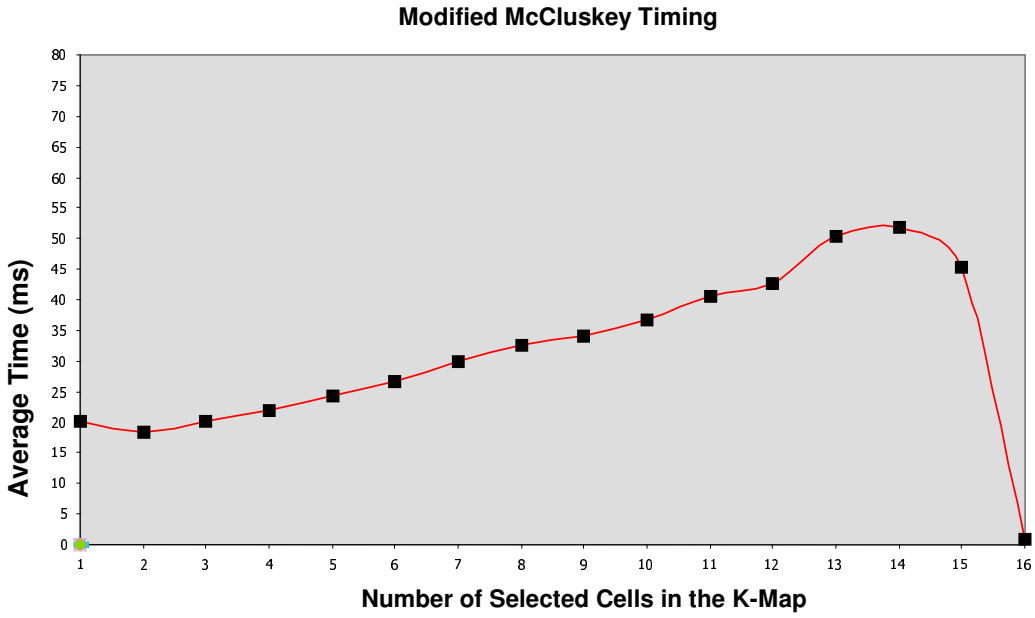

Fig. 6. Running time.

Mileto T. and Putzolu G. (1964): *Average values of quantities appearing in Boolean function minimization*. — IEEE Trans. El. Comp., Vol. 13, No. 2, pp. 87–92.

Mileto T. and Putzolu G. (1965): *Statistical complexity of algorithms for Boolean function minimization*. — J. ACM, Vol. 12, pp. 364–375.

Petrick S.K. (1959): *On the minimization of Boolean functions*. — Proc. Int. Conf. *Information Processing*, Paris: Unesco, pp. 422–423.

Quine W.V. (1952): *The problem of simplifying truth tables*. — Amer. Math. Month., Vol. 59, No. 8, pp. 521–531.

Shannon C.E. (1938): *A symbolic analysis of relay and switching circuits*. — Trans. AIEE, Vol. 57, No. 6, pp. 713–723.

Wegener I. (1987): *The Complexity of Boolean Functions*. — New York: Wiley.

Received: 9 April 2002
Revised: 5 March 2003