On the Use of Mutations in Boolean Minimization

Petr Fišer, Jan Hlavička
Czech Technical University, Karlovo nám. 13, 121 35 Prague 2

Contact author: Jan Hlavicka
e-mail: hlavicka@fel.cvut.cz
Fax: +4202 2492 3325

Keywords: Boolean minimization, logic design, implicant generation, covering problem, mutations

On the Use of Mutations in Boolean Minimization

Abstract

The paper presents a new method of Boolean function minimization based on an original approach to implicant generation. The selection of these newly included literals, as well as the subsequent rejection of some others to obtain prime implicants, is based on heuristics working with the frequency of literal occurrence. Instead of using this data directly, some mutations are used on different places of the algorithm. The technique of mutations and their influence on the quality of the result obtained is evaluated in this article.

The BOOM system implementing the proposed method is efficient especially for functions with several hundreds of input variables, whose values are defined only for a small part of their range. It has been tested both on standard benchmarks and on problems of a much larger dimension, generated randomly. These experiments proved that the new algorithm is very fast and that for large circuits it delivers better results than the state-of-the-art ESPRESSO.

Introduction

Modern design methods use heuristics in many places, where a deterministic control of the decision algorithm would be too involved. Although the quality of the result is in such cases not guaranteed, the average result is mostly satisfactory. To improve the quality of the solution, some slight modifications of the heuristics may be occasionally undertaken. These are called mutations.

Among the classical problems of logic design, the problem of two-level minimization of Boolean functions surely belongs to the oldest. Even the new implementation technologies, like, e.g., multi-level custom design, FPGAs, and above all the PLA require in some phase this minimization, often referred to as PLA minimization. They are encountered in modern application areas like design of control systems, design of built-in self-test equipment, during solution of problems in the area of artificial intelligence, software engineering, etc. The modern design problems are mostly characterized by a large number of input variables and a limited number of input states for which the output value is determined (care states). The number of don't care states reaches then astronomical values, and the quality of a minimization method is then determined by its ability to take advantage of their existence without enumerating them. Similarly, it must be able to cope with the existence of large number of prime implicants (PIs), most of which are not needed for the minimal solution.

The Boolean minimization methods started with the papers by Quine [11] and McCluskey [8], which formed a basis for many follow-up methods. These mostly copied the structure of the original method, implementing the basic two phases known as PI generation and covering problem (CP) solution. Some more modern methods, including the well-known ESPRESSO [15], [6] with its later improvements ESPRESSO-EXACT and ESPRESSO SIGNATURE [9] combine these two phases, reducing the number of implicants to be processed.

A sort of combination of PI generation with solution of the CP, leading to a reduction in the total number of prime implicants generated, is also used in the BOOM (BOOlean Minimization) approach proposed here. However, the principal improvement consists in speeding up PI generation by applying the top-down approach instead of the commonly used bottom-up approach. The basic principles of the proposed method and the properties of the BOOM system were published in some previous reports [4], [5], [7]. The present paper concentrates on one of specific features of this system, namely on the role of mutations. BOOM was programmed in Borland C++ Builder and tested under MS Windows NT.

The paper has the following structure. After a formal problem statement in Section 2, the role of mutations in the proposed method is presented in Section 3. Section 4 describes the investigation of the influence of different forms of mutations on the results obtained. The results of experimental verification of the BOOM system are evaluated and commented in Section 5.

2. Problem Statement

We will assume the standard problem of Boolean minimization. A function of n input variables is defined by a truth table describing the **on-set** F(x_1, x_2, … x_n) and **off-set** R(x_1, x_2, … x_n). The terms not represented in the input field of the truth table are implicitly assigned don't care values of the corresponding output function, i.e., they represent the **don't care set** $D_i(x_1, x_2, \dots x_n)$. Listing the two care sets instead of an on-set and a don't care set, which is usual, e.g., in MCNC benchmarks, is more practical for problems where n is of the order of hundreds, because there the size of the don't care set usually largely exceeds all other sets.

Our task is to formulate a synthesis algorithm, which will produce a sum-of-products expression $G = g_1+g_2+\dots+g_t$, where $F \subseteq G \subseteq F+D$ and t is minimal. In case of a set of m functions we should minimize the total number of implicants while some of them can be used for more output functions.

This formulation of the minimization process uses the number of product terms (implicants) as a universal quality criterion. This is mostly justified, but it should be kept in mind that the measure of minimality should correspond to the needs of the intended application. ESPRESSO uses the sum of the number of literals and the output cost, i.e., the number of inputs into all output OR gates.

3 Main Components of the BOOM System

In addition to the usual two major phases found in most minimization, namely **generation of prime implicants** and **solution of the covering problem**, BOOM uses several additional steps which improve the quality of solution. The generation of PIs consists of two steps: **Coverage-Directed Search (CD-Search)**, which produces a set of implicants needed for covering the input function, and **Implicant Expansion (IE)**, which converts these implicants into PIs. Then the CP is solved for all obtained primes using the heuristic method suggested in [13], [3]. For multi-output functions there is on top of that an **Implicant Reduction** phase, which derives group implicants from PIs and finally the **Output Reduction**, which eliminates redundant implicants of individual outputs (corresponding to the ESPRESSO's MAKE_SPARSE procedure [6]).

3.1 Mutations

The heuristics used to implement individual steps of this procedure are based on the study of statistical properties of the given Boolean functions. The results of these tests often give results whose interpretation is not straightforward and thus several different decisions may be taken. In such cases the mutations, implemented as a random choice used in place of deterministic decision, may be of help. These mutations may be used on several places of the procedure. Section 4 will investigate their usefulness, i.e., the quality of the solution obtained and the time needed to find the solution.

First let us introduce a formal definition of mutation. Let us have a set $\boldsymbol{X}$ of elements x_i with weights $w(x_i)$ assigned. The selection function $S(\boldsymbol{Y})$, $\boldsymbol{Y} \subset \boldsymbol{X}$, is a mapping into $\boldsymbol{X}$ such that the function S returns the element $x_j \in \boldsymbol{Y}$ with the highest value of w. As a **mutation** of the function $S(\boldsymbol{Y})$ we will call a random selection of an element from $\boldsymbol{Y}$, i.e., a selection that is not influenced by the weights. The selection function affected by the mutations will be denoted as $S_m(\boldsymbol{Y}, k)$, where $k \in <0,1>$ is the mutation rate. S_m returns then a random element from $\boldsymbol{Y}$ with the probability k and the element $S(\boldsymbol{Y})$ with the probability $1-k$.

3.2 Iterative Minimization

As mentioned above, in the BOOM system the result of minimization depends to a certain extent on random events. Thus when there are several equal possibilities to choose from, the decision is made randomly, which increases the chance that the repeated application of the same procedure to the same problem will yield different solutions. The iterative minimization concept takes advantage of the fact that each iteration produces a new set of implicants satisfactory for covering all minterms of all output functions. These implicants are recorded and the set of implicants gradually grows until a maximum reachable set is obtained. Finally, the covering problem for all generated primes is solved.

3.3 Coverage-Directed Search

The PI generation consists of several phases is denoted as CD-search, because it consists in a directed search for the most suitable literals that should be added to some previously constructed term. Thus instead of increasing the dimension of an implicant starting from a 1-minterm, we use a top-down approach by gradually decreasing the dimension of an n-dimensional hypercube by adding literals to its term, until it becomes an implicant of F. This happens at the moment when there is no intersection with a 0-term.

It is well known that any product term describing a hypercube that lies within another hypercube must contain all literals present in the term of the larger hypercube. Thus it is advantageous to start the implicant generation by selecting the most frequent input literal from the given on-set. The selected literal describes an *n-1* dimensional hypercube, which may be an implicant, if there is no intersection with a 0-term. If there is a 0-minterm covered, we add one literal and verify whether the new term already corresponds to an implicant by comparing it with all 0-terms. We continue adding literals with the maximum frequency of occurrence in terms that can be yet covered by this implicant, until an implicant is generated, then we record it, remove all on-set terms covered by this implicant and start searching for other implicants.

A certain drawback of the CD-search algorithm is that it is greedy and the implicants need not be prime. Thus they should be expanded into PIs (see Subsection 4.2).

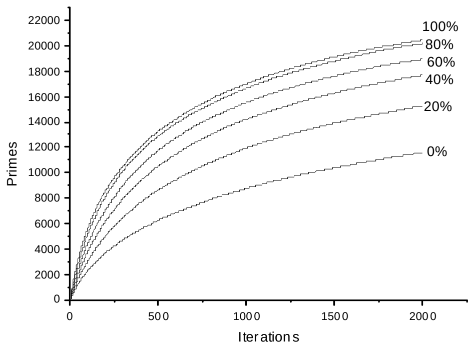
4 Use of Mutations

4.1 CD-Search Mutations

As it was stated above, during the CD-search only literals with maximum frequency of occurrence are candidates for selection. In some cases this selection criterion may prevent reaching the minimum solution. In other words, there may exist implicants that are unreachable by a strict CD-search, although they are necessary for obtaining the minimal solution.

This can be solved by the introduction of **mutations**. In this case it means the selection of a random literal that has a non-zero frequency of occurrence in the currently reduced on-set. The extent of mutations is controlled by the mutation rate introduced in Subsection 3.1. In Figs. 1 and 2 we can observe that the more mutations are implanted, the faster is the growth of the number of primes during iterative minimization. This is caused by the variety of implicants that are being produced. The mutation rate (portion of literals selected at random) was changed as a parameter from 20 to 100 %. The problem solved was the minimization of a single-output function of 20 input variables with 300 defined minterms.

Although the number of PIs grows faster for higher mutation rates, the CD-search is slowed down. This is because implicants that cover less 1-terms are produced and thus more of them must be generated to cover all the on-set. The time needed for one pass of the CD-search as a function of the mutation rate is shown in Fig. 3.



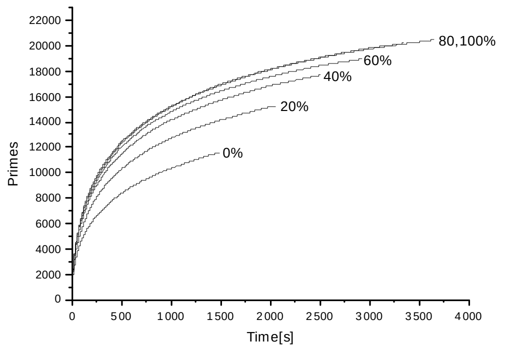


Fig. 1, 2. *PI set growth for various mutation rates*

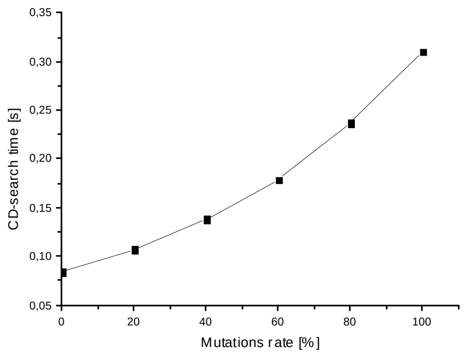


Fig. 3. *CD-search run-time growth with increasing mutation rate*

The effects documented above can be summarized as: the mutations slow down the whole minimization process and make it less effective, hence selecting literals with maximum frequency of occurrence is the best method of literal selection. The necessary set of implicants needed for covering the on-set is then reached in the shortest time, and any deviation from this rule will slow down the algorithm. However, experiments show that 2-5% of mutations can improve the result by producing some originally unreachable implicants.

4.2 Implicant Expansion

As mentioned above, the implicants constructed during the CD search need not be prime. To increase the chance that fewer implicants will be needed to cover all 1-terms of the given function, we have to increase their size by IE, which means by removing literals (variables) from their terms. When no literal can be removed from the term any more, we get a prime implicant. There exist many IE methods differing in the exhaustiveness of expansion and the order in which literals are tried for removal. Some of them used in BOOM are listed in [5].

4.3. Implicant Reduction (IR)

To obtain the minimum solution we may need implicants of more than one output function that are not primes of any. Here, the next part of minimization – **Implicant Reduction** - takes place.

All obtained primes are tried for reduction by adding literals in order to become implicants of more output functions. The method of implicant reduction is similar to a CD-search. Literals that prevent intersecting given term with most off-set terms are repetitively added until there is no chance that the implicant will be used for more functions. When no further reduction yields any possible improvement, the reduction is stopped and the implicant is recorded. If a term no longer intersects with the off-set of any output function, it becomes its implicant. All implicants that were ever found are stored and output functions are assigned to them. After that simple dominance checks are preformed in order to eliminate implicants that are dominated by another implicant in all functions. This causes that the total number of non-primes surprisingly decreases during the iterations, because newly generated primes often "beat" some group implicants. Fig. 4 shows the typical growth of the number of group implicants (non-primes) as a function of the number of iterations. Here the function of 13 input variables, 13 output variables and 200 defined terms was used for demonstration.

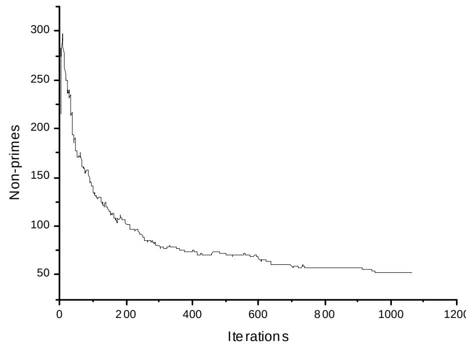


Fig. 4: *Number of non-primes during the iterations*

4.4 Implicant Reduction Mutations

As well as the CD-search the implicant reduction can be enhanced by mutations. Here the mutation is a random selection of a literal that prevents intersecting given term with at least one zero term. The number of group implicants (non-primes) grows with increasing mutation rate very fast – see Fig. 5. However, the experimental results show that the final result of minimization (the quality of solution) depends on the rate of IR mutations only little. Mostly group implicants that arise from the mutations are not necessary for reaching the minimum solution. However, there may exist functions that may require the use of mutations on order to be the minimum solution found.

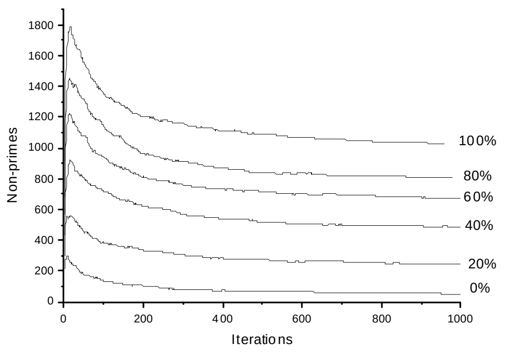


Fig. 5: *Non-primes growth for different IR mutation rates*

5 Experimental Results

Extensive experimental work was done to evaluate the efficiency of the proposed algorithm, especially for problems of large dimensions. Both runtime in seconds and result quality were evaluated. The processor used was a Celeron 433 MHz with 160 MB RAM. The quality of the results was measured by three parameters: total number of literals, output cost and number of product terms (implicants).

5.1. Solution of MCNC Benchmark Problems

First a group of MCNC benchmark problems was solved by ESPRESSO 2.3 and by BOOM [14]. The results of the comparison are shown in Tab. 1. The column *n*/*m*/*p* contains the parameters of the problem, namely the number of inputs, outputs and care terms. The benchmarks appearing in the table were solved by BOOM in one pass, hence the runtimes are very short (the 0.01 sec. value in most cases indicates a non-measurable runtime). Tab. 1 shows that problems with a large number of defined terms (*p*) were often solved by ESPRESSO in shorter time. This is due to the quadratic dependence of runtime on the number of terms in BOOM [7]. In all these examples the quality of solutions was equal, in one case BOOM gave even better result than ESPRESSO.

Tab. 1 MCNC Benchmark problems

		ESPRESSO		BOOM	
Bench	*n*/*m*/*p*	Time	lit/out/impl	time	lit/out/impl
9sym	9/1/158	0.12	516/86/86	0.05	same
al2	16/47/139	0.15	324/103/66	0.66	same
Alu1	12/8/39	0.10	41/19/19	0.01	same
Alu2	10/8/241	0.20	268/79/68	0.04	same
Alu4	14/8/1184	1.63	4443/644/575	3.52	same
b9	16/5/292	0.18	754/119/119	0.25	same
br1	12/8/107	0.12	206/48/19	0.02	same
br2	12/8/83	0.11	134/38/13	0.01	same
Clpl	11/5/40	0.12	55/20/20	0.01	same
Con1	7/2/18	0.10	23/9/9	0.01	same
dc1	4/7/25	0.12	27/27/9	0.01	same
dc2	8/7/101	0.13	207/52/39	0.01	206/51/39
dk27	9/9/24	0.10	31/15/10	0.01	same
dk48	15/17/64	0.24	115/28/22	0.02	same
ex7	16/5/292	0.19	754/119/119	0.22	same
in7	26/10/142	0.14	337/90/54	0.13	same
Max46	9/1/155	0.14	395/46/46	0.03	same
Misex1	8/7/41	0.12	51/45/12	0.01	same
Newpla	12/10/60	0.14	74/28/17	0.02	same
Newpla1	17/2/25	0.15	64/12/10	0.01	same
Newpla2	10/4/26	0.18	42/7/7	0.01	same
Newbyte	5/8/16	0.17	40/8/8	0.01	same
Newcond	11/2/72	0.16	208/31/31	0.01	same

		ESPRESSO		BOOM	
Bench	*n/m/p*	Time	lit/out/impl	time	lit/out/impl
Newcwp	4/5/24	0.18	31/19/11	0.01	same
Newill	8/1/18	0.13	42/8/8	0.01	same
Newtag	8/1/12	0.16	18/8/8	0.01	same
Newtpla	15/5/63	0.15	176/23/23	0.01	same
Newtpla1	10/2/15	0.16	33/4/4	0.01	same
Newtpla2	10/4/26	0.19	54/15/9	0.01	same
p82	5/14/74	0.17	93/56/21	0.02	same
rd53	5/3/67	0.09	140/35/31	0.01	same
rd73	7/3/274	0.14	756/147/127	0.08	same
rd84	8/4/511	0.35	1774/296/255	0.37	same
sao2	10/4/137	0.11	421/75/58	0.04	same
sqrt8	8/4/66	0.11	144/44/38	0.01	same
squar5	5/8/65	0.12	87/32/25	0.01	same
vg2	25/8/304	0.15	804/110/110	0.47	same
xor5	5/1/32	0.08	80/16/16	0.01	same
z9sym	9/1/72	0.09	226/34/34	0.01	same

5.2. Test Problems with *n*>50

The MCNC benchmarks have relatively few input terms and few input variables (*n* never exceeds 128) and also have a small number of don’t care terms. In order to compare the performance and result quality achieved by the minimization programs on larger problems, a set of problems with up to 300 input variables and up to 300 minterms were solved. The truth tables were generated by a random number generator, for which only the number of input variables, number of care terms and number of don’t cares in the input portion of the truth table were specified. The number of outputs was set equal to 5. The on-set and off-set of each function were kept approximately of the same size. First, the problem was solved by ESPRESSO and then by BOOM, which ran until the solution of the same or better quality was reached. The quality criterion selected was the sum of the number of literals and the output cost. For all samples the same or better solution was found by BOOM in much shorter time than by ESPRESSO.

Tab. 2 Solution of problems with n>50

p/n	50	100	150	200	250	300
50	111/0.06 (1) 132/5.71	92/0.08 (1) 92/7.15	83/0.12 (1) 84/20.00	77/0.59 (4) 88/42.77	77/0.39 (2) 77/51.29	75/8.69 (35) 76/110.74
100	219/2.36 (9) 220/7.38	190/2.57 (7) 190/27.95	174/4.19 (9) 176/104.38	163/31.05 (35) 165/114.65	155/14.74 (19) 158/184.31	154/1.40 (2) 154/317.39
150	330/2.34 (4) 334/21.42	287/9.44 (10) 287/79.47	289/1.11 (1) 289/129.20	249/31.23 (20) 253/367.19	231/57.38 (29) 233/396.01	247/44.66 (19) 248/569.44
200	338/20.26 (11) 447/55.24	401/37.79 (15) 404/209.27	349/91.96 (25) 350/297.20	344/63.23 (20) 347/557.54	331/2.27 (1) 334/794.97	321/2.89 (1) 328/857.19
250	576/32.38 (9) 576/80.27	460/242.27 (36) 463/323.27	443/142.71 (23) 450/404.09	409/481.63 (50) 445/934.13	423/196.56 (27) 425/1607.45	385/507.23 (52) 389/2354.24
300	594/83.35 (13) 597/105.20	580/203.06 (22) 588/333.90	505/446.42 (38) 508/798.84	506/416.01 (34) 512/847.05	500/470.90 (38) 500/1822.01	465/205.76 (32) 466/3012.90

Entry format: BOOM: #of literals+output cost / time in seconds (# of iterations)

ESPRESSO: #of literals+output cost / time in seconds

5.3 Solution of Very Large Problems

The main strength of BOOM lies in its capability of minimizing functions of a large number of input variables, which could be seen from Tab. 2. For functions with more than 300 input variables ESPRESSO could not be used at all because of its extremely long runtimes. However, BOOM is beyond this border usable without any problem. This is due to the fact that, unlike other Boolean minimization methods, the time complexity as a function of input variables is almost linear [7]. Just for demonstration: for problems with 1000 input variables, 10 output variables and 1000 defined minterms one iteration of BOOM takes less than 20 minutes on a common PC.

6 Conclusions

A new Boolean function minimization method has been proposed and implemented as the BOOM minimization tool. Its application is advantageous above all for problems with large dimensions and a large number of don't care states where it beats ESPRESSO both in minimality of the result and in runtime. The PI generation method is very fast, hence it can easily be used in an iterative manner. The notion of mutations in the iterative process was introduced and their possible positive influence on the final solution.

The BOOM minimizer has been placed on a web page [14], from where it can be downloaded by anybody who wants to use it.

Acknowledgment

This research was in part supported by the grant of the Czech Grant Agency GACR 102/99/1017.

References

[1] Brayton, R.K. et al.: Logic minimization algorithms for VLSI synthesis. Boston, MA, Kluwer Academic Publishers, 1984
[2] Coudert, O. - Madre, J.C.: Implicit and incremental computation of primes and essential primes of Boolean functions, In Proc. of the Design Automation Conf. (Anaheim, CA, June 1992), pp. 36-39
[3] Coudert, O.: Two-level Logic Minimization: an Overview, Integration, the VLSI journal, 17-2, pp. 97-140, Oct. 1994.
[4] Fišer, P. – Hlavička, J.: Efficient Minimization Method for Incompletely Defined Boolean Functions, Proc. 4th Int. Workshop on Boolean Problems, Freiberg, (Germany), Sept. 21-22, 2000, pp. 91-98
[5] Fišer, P. - Hlavička, J.: Implicant Expansion Method used in the BOOM Minimizer. Proc. IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS’01), Gyor (Hungary), 18-20.4.2001 (in print)
[6] Hachtel, G.D. - Somenzi, F.: Logic synthesis and verification algorithms. Boston, MA, Kluwer Academic Publishers, 1996, 564 pp.
[7] Hlavička, J. - Fišer, P.: Algorithm for Minimization of Partial Boolean Functions, Proc. IEEE Design and Diagnostics of Electronic Circuits and Systems (DDECS00) Workshop, Smolenice, (Slovakia) 5-7.4.200. ISBN 80-968320-3-4, pp.130-133
[8] McCluskey, E.J.: Minimization of Boolean functions. The Bell System Technical Journal, 35, No. 5, Nov. 1956, pp. 1417-1444
[9] McGeer, P. et al.: ESPRESSO-SIGNATURE: A new exact minimizer for logic functions. Proc. DAC’93
[10] Nguyen, L. – Perkowski, M. – Goldstein, N.: Palmini – fast Boolean minimizer for personal computers. In Proc. DAC’87, pp.615-621
[11] Quine, W.V.: The problem of simplifying truth functions, Amer. Math. Monthly, 59, No. 8, 1952, pp. 521-531.
[12] Rudell, R.L. – Sangiovanni-Vincentelli, A.L.: Multiple-valued minimization for PLA optimization. IEEE Trans. on CAD, 6(5): 725-750, Sept.1987
[13] Rudell, R.L.: Logic Synthesis for VLSI Design, PhD Thesis, UCB/ERL M89/49, 1989.
[14] http://cs.felk.cvut.cz/~fiserp/boom/
[15] http://eda.seodu.co.kr/~chang/ download/espresso/
[16] ftp://eecs.berkeley.org