MINTERM AND IMPLICANT SELECTION CRITERIA FOR DIRECT COVER ALGORITHMS AND OFF-BASED MINTERM ORDERING

Hakan AKAR[1*] and Fatih BAŞÇİFTÇİ [2]

[1,2]Selçuk University, Technology Faculty, 42003, Selçuklu, Konya, Turkey

Email: maviakar@gmail.com[1] (corresponding author), basciftci@selcuk.edu.tr[2]

DOI: https://doi.org/10.22452/mjcs.vol33no4.2

ABSTRACT

This paper presents a new method for minimization of logic functions. We summarized minterm and implicant selection rules for direct cover algorithms and introduced OFF based ordering method. As other minterm and implicant selection algorithms use ON minterms and runs in minimization phase, they may not find real isolated minterms for insignificant functions. The proposed method uses OFF minterms for ordering minterms. Thus insignificant functions including Don't Care minterms, could be handled easily with this approach. Results revealed that although proposed algorithm use running time in reordering process, it saves much time in simplification process. In addition, output functions are more simplified with OFF based ordering method.

Keywords*: Boolean function, digital electronics, switching circuit simplification, minimization, direct cover, OFF based ordering, isolated minterms.*

1.0 INTRODUCTION

Logic functions depend on binary system, which developed by Leipniz in early 1700 [1]. But it was George Boole, who revealed the connection between arithmetic and logic, thus introduced Boolean algebra including AND, OR, NOT, NOR operators in 1854 [2]. In 1881, "Logical Machine" which was a full functional mechanical logic processor is constructed [3]. First modern electronic AND gate is represented in 1924 and electromechanical logic gates are first used in computers in 1938 [4]. Finally, Boolean algebra is used for analyse and design of switching circuits by Shannon. He also introduced algebraic minimization of logical circuits in 1936 [5].

Logic functions are widely used in various areas such as hardware synthesis, data mining, cryptography, biology, etc. [6-9]. Therefore efficient minimization of logic function is important as a common tool for many disciplines [10, 11]. Several techniques have been introduced to perform minimization function such as ESPRESSO, Quine-McCluskey, Binary Decision Diagrams, KMOA, BOOM, etc. [12-17]. Some functions may include huge input variables, huge OFF minterm sets, difficult to cover ON minterms or insignificant input values. As contents of functions may differ, research findings revealed that almost none of simplification techniques is superior to others in all function classes [18].

Logic functions consist of ON and OFF minterms, which drive output of function to 1 and 0, respectively. Also, there could be some insignificant inputs named as Don't Care (DC). Finding the right minterm to begin simplification is very crucial [19]. Simplification of ordered minterm functions is not only time and memory efficient, but also results a more simplified logic function.

Second section of this paper gives some brief information about minterm minimization and isolated minterms. Third section summarizes previous work about minterm and implicants selection algorithms. Section four introduces a new OFF based ordering method for logic minimization. Section 5 explains how to implement presented method with examples, and section 6 displays the results showing how useful the proposed method. In section 7, paper ends with conclusions.

2.0 BACKGROUND

2.1 Minterm Minimization

Logic functions consist of input variables which results either logic high (1), or logic low (0) or DC state [12]. State of the input variables that results desired output such as "1" is called minterms. Figure 1 is a truth table for a sample function of 4 variables. This function can also be specified as {1,3,4,5,8,9,13,15}. Instead of using a function consist of 8 minterms like in Fig. 1. we could use a more simplified function consist of less implicants.

This is called logic function simplification/minimization. There are many algorithms for simplification of logic functions [20-22]. Direct cover algorithms selects a minterm and then finds a prime implicant covering the minterm [23]. Basic steps of direct cover algorithms are as follows [19, 24, 25]:

```
Begin

[01]    Prepare input function by putting them into groups of ON-SET, OFF-SET, DC-SET.
[02]    Pick a minterm m from ON-SET.
[03]    Generate implicants covering m.
[04]    Add one of implicants to I-SET.
[05]    Remove m from ON-SET.
[06]    If ON-SET is not empty goto step 2.
[07]    Return I-SET.
End
```

The most time and memory consuming step of this algorithm is step 3, finding an implicant covering the selected minterm. But step 2, minterm selection procedure directly affects step 3. Some minterms are difficult to cover, some of them has many implicant possibilities.

2.2 Isolated Minterms

It is important to compute similarities between minterms. Some minterms are very similar to each other which means difference between them is $2i$, where $i = 0, 1, 2, \ldots$, n. These minterms can be handled together for finding prime implicants [26]. It is easy to find implicants and cover these minterms. And there will be equal or more than one implicant covering these minterms. On the other hand, some minterms are more than $2i$ different from other minterms. There is only one or no implicant covering these minterms. It takes less time and memory to find implicants covering them. These minterms are called isolated minterms. Finding implicants for isolated minterms results in more computable remaining function.

For example, function *G* is given in Disjunctive Canonical Form (DCF) as follows:

$$G(a, b, c, d) = \sum m(1, 3, 4, 5, 8, 9, 13, 15) \tag{1}$$

As seen on Fig. 1, isolated minterms {3, 4, 8, 15} has only one similar minterms, which results in only one implicant covering each of these minterms. As there is only one potential implicant, there is no need to selection procedure. In addition, finding implicants for remaining minterms will be easier. On the other hand, minterms {1, 5, 9, 13} has many similar minterms and there are more than one potential implicants. Implicant selection procedure could be time and memory consuming.

	00	01	11	10
00		1	1	
01	1	1		
11		1	1	
10	1	1		

G(x)	**a**	**b**	**c**	**d**	**output**
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Fig. 1: Karnaugh map and truth table for function *G*

Finding prime implicants for covering minterms is the most time and memory consuming stage of direct cover algorithms. Many heuristic techniques are presented for this stage [18-20]. However, giving priority to isolated minterms makes this stage much easier.

In most direct cover algorithms, output function consists of prime implicants given by,

$$\mathrm{G}_{PI}(x) = \{PI(x)\}_{i=1}^{K} \tag{2}$$

Where prime implicant selection procedure is given by,

$$PI_k(x) > PI_i(x) \leftrightarrow |G_{ON} \cap PI_k(x)| > |G_{ON} \cap PI_i(x)| \quad \text{for all } i \neq k \tag{3}$$

This approach selects most minterm covering *PI(x)*. But, (3) cannot find the best solutions in all cases. To demonstrate the defiance in (3), consider simplifying *G(x)* given in (1) [27].

$G(a, b, c, d) = \sum m(1, 3, 4, 5, 8, 9, 13, 15)$ can be stated as

G_{ON} = {0001, 0011, 0100, 0101, 1000, 1001, 1101, 1111}. Off minterms for this function is G_{OFF} = {0, 2, 6, 7, 10, 11, 12, 14}.

If we start the simplification procedure with minterm 0001 then following prime implicants PI_1 (0001) = xx01 and PI_2 (0001) = 00x1 will be obtained. Therefore,

C(xx01) = $G_{ON} \cap$ xx01 = {0001, 0011, 0100, 0101, 1000, 1001, 1101, 1111} $\cap$ xx01
= {0001, 0101, 1101, 1001}
C(00x1) = $G_{ON} \cap$ 00x1 = {0001, 0011, 0100, 0101, 1000, 1001, 1101, 1111} $\cap$ 00x1
= {0001, 0011}

Since $|$C(00x1)$|$ = 2 and $|$C(xx01)$|$ = 4, rule(3) leads us to select C(xx01) as
PI_1(0001) = xx01.

Therefore: G_{PI} = {xx01};
$G_{ON} = G_{ON}$ # xx01 = {0011, 0100, 1000, 1111}.

Thus, the updated *ON* minterm series to be simplified is G_{ON} = {0100, 0111, 1001, 1010} and already covered minterms are G_{DC} = {1100, 1101, 1111, 1110}. Fig. 2 represents the current state of the function *f*. Covered minterms are stated as *DC*.

	00	01	11	10
00		DC	1	
01	1	DC		
11		DC	1	
10	1	DC		

Fig. 2: Karnaugh map for current state of function *G*

Fig. 2 reveals that, uncovered 4 remaining minterms are not close to each other. In addition, there is only one prime implicant covering each. Remaining prime implicants are as follows:

PI_2(0011) = 00x1 , PI_3(0100) = 010x , PI_4(1111) = 11x1 , PI_5(1000) = 100x.

As a result,

$G_{ON} = G_{ON} \# \{PI_2, PI_3, PI_4, PI_5\}$
$= \{0100, 1001, 0111, 1010\} \# \{00x1, 010x, 11x1, 100x\} = \varnothing$
$G_{PI} = \{xx01, 00x1, 010x, 11x1, 100x\}$

Some of the minterms are both covered by PI_1 and other *PI*s. In other words, PI_1(xx01) is redundant, as all minterms are covered by other *PI*s. Starting minimization for relatively central minterms such as "0101, 1101, 1001" will not change the result. On the other hand, if we start minimization from isolated minterms, simplification process would be shorter and output function would be more simplified.

3.0 RELATED WORK

This section presents a survey of other minterm and implicant selection procedures that direct cover algorithms use. There are many heuristic algorithms for simplification of logic functions. Differences of these direct cover algorithms lie in the decision making of minterm and implicant selection process. These selection procedures directly affects algorithm performance. We should keep in mind that some decision process could be time and memory consuming.[28] Some algorithms decide randomly for minterm and/or PI while others decided by some calculations.

In Pomper and Armstrong algorithm, there is no minterm selection heuristic. Minterms are minimized as given in the input file. But this algorithm finds the PI that drives largest number of minterms reduced to zero (LRZ). All the PIs are found and biggest one is selected according to their DC driving capability [29].

Minterm and PI selection is done according to some rules in Besslich Algorithm. Besslich algorithm compares every minterm with each other and assign them an isolation weight (IW) as given in (4). Similar minterms are heavier than others and relatively they are not isolated. Simplification process starts from lightweight isolated minterms. After all PIs are determined for a minterm, effectiveness factor for each PI is calculated. Effectiveness factor is equal to number of covered minterms divided by cost of PI. To sum, Besslich algorithm gives priority to lightweight minterms and selects the most effective PI [30].

$$Minterm\,Weight = \sum_{m=1}^{n} \beta_m - \gamma_m \qquad (4)$$

Dueck and Miller algorithm selects the minterms by their isolation factor (IF). Isolation factor is inversely proportional with number of expandable adjacencies and number of directions of expandable adjacencies (DEA_α) given in (5). Minterms having high isolation factor has priority for simplification. This algorithm creates all PIs for selected minterm and calculates Break Count Reduction (BCR) for each. BCR reveals how much simplified the function is. Then a PI which simplifies remaining function as simple as possible is selected [27, 31].

$$DEA_\alpha = \sum_{i=1}^{n} \gamma_i \qquad (5)$$

Yang & Wang algorithm initially computes clustering factors of all minterms (CFN). Then minimization algorithm selects a minterm that has the smallest clustering factor. Considering implicant selection phase, Yang & Wang algorithm tries to select most isolated implicants. The algorithm computes Neighborhood Relative Count (NRC) for all implicants. NRC is the implicants' coupling strength with their neighbors. Selecting an implicant means breaking the coupling between this implicant and its neighbors. Therefore this algorithm selects the most isolated implicant which means lowest NRC [23, 32].

BOOM algorithm finds all possible PIs covering all minterms and creates a virtual PI pool. Thus there is no importance of minterm selection procedure, it selects minterms randomly. Skipping minterm selection procedure and handle all PIs together helps to save time and memory [33, 34]. PI selection criterias can be summarized in following order:
1. Select a PI covering the most minterms.
2. Select a PI covering difficult minterms.
3. Select a PI involving less number of literals.
4. Select a PI randomly.

Weighted direct cover and Ordered direct cover algorithms combine all selection rules to create a selection criterion. These algorithms compute all selection rules and combine them with some factors. Weighted direct cover algorithm assigns a weight to different decision rules and computes a factor using all these selection rules. Minterm and implicant selection is done according to these factors. Ordered direct cover algorithm gives some priorities to decision rules. This algorithm proposes that minterm selection order shall be Smallest IW, smallest CFN, Smallest IF and implicant selection order shall be smallest RBC, largest LRZ, and smallest NRC. These algorithms do not present a novel decision rule, try to use the efficiency of other rules [35, 36].

Researchers revealed that none of the algorithms are superior to each other in all benchmarks [31]. Some benchmarks do not need minterm reordering thus random selection may save time and memory. Some benchmarks may include difficulty to cover minterms while some include many DC that should be taken into consideration. Table 1 summarizes the selection criterias of algorithms.

Table 1: Minterm and Implicant Selection Criterias of Algorithms

	Minterm Selection	**Implicant Selection**
Pomper & Armstrong	Random	Drives most DC
Besslich	Lightest minterm	Drives most DC
Dueck and Miller	Biggest isolation factor	Most simplifies function
Yang & Wang	Smallest clustering factor	Lowest NRC
Boom	Random	Drives most DC
Weighted Direct Cover	Weighted factor of 3 rules	Weighted factor of 3 rules
Ordered Direct Cover	Give priority order to 3 rules	Give priority order to 3 rules

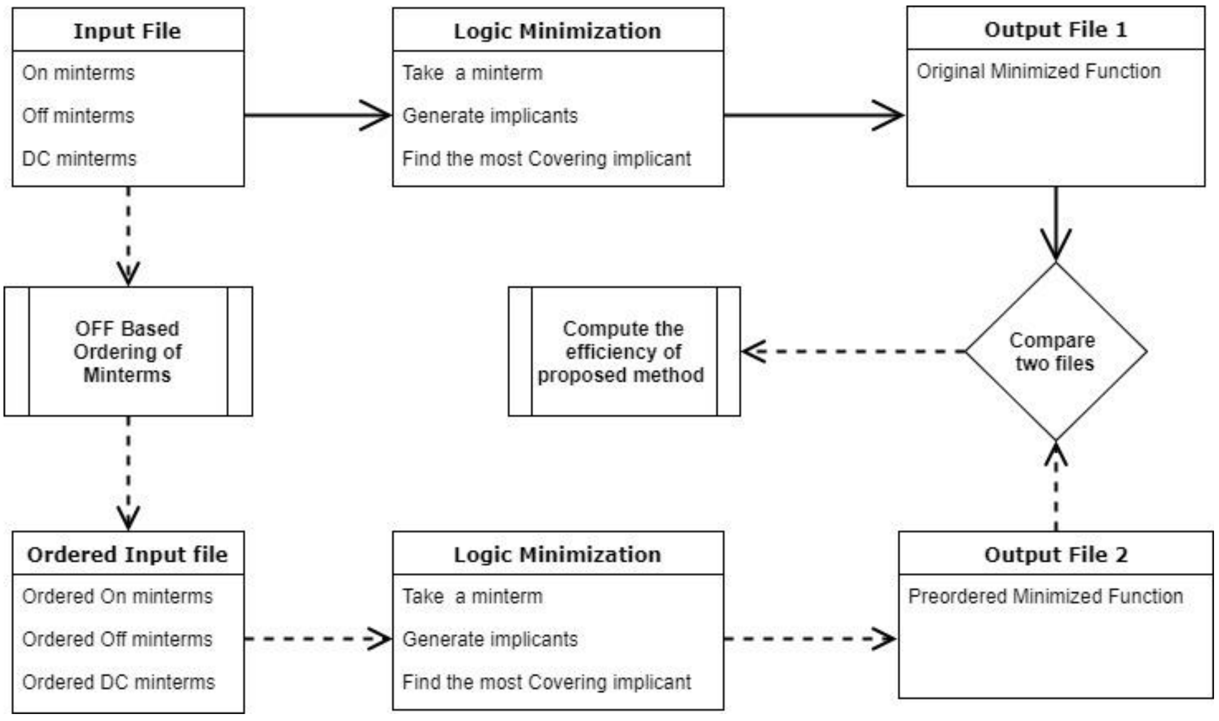


Fig. 3: OFF based ordering method

4.0 OFF BASED ORDERING OF MINTERMS

This section describes a new approach, preprocessing input file before minimization. Input function files consist of ON, OFF and DC minterms. Our algorithm does preparation process on input file. Therefore minimization algorithms work better.

Most of the algorithms use ON minterms for finding isolated minterms. Some algorithms select minterms which has fewest covering options. Some algorithms look for minterms which has most difficult covering options. Due to ON minterms that may be surrounded by DC minterms, and DC minterms can be included for obtaining cheapest PIs, these algorithms may not find real isolated minterms in large DC set functions.

In this study, we developed an ordering algorithm for ON, OFF and DC minterms. Our algorithm uses OFF minterms to reorder ON minterms. This heuristic enables to consider DC minterms as a potential PI element. The proposed method reorders the input file before minimization as seen on Fig. 3. Thus minimization process is easier and faster. Also result quality (minimization ratio of output file) goes higher.

```
Input: Function File
Output: Ordered Function File

Create ON-set from ON-minterms;
Create OFF-set from OFF-minterms;
Create DC-set from DC-minterms;
FOR all minterm "M" in input file DO
    If "M" is ON-minterm DO
        T=0;
        FOR all F in OFF-set DO
            X, S=0;
            X=$M \oplus F$; //EXOR operation of two minterms
            S= Count number of "1" in X; // compute the difference
            T=T+S;
    Link value of T to M;
    Else if "M" is OFF-minterm DO
        T=0;
        FOR all N in ON-set DO
            X, S=0;
            X=$M \oplus N$; //EXOR operation of two minterms
            S= Count number of "1" in X; // compute the difference
            T=T+S;
    Link value of T to M;
    Else // "M" is a DC minterm
        T=0;
        FOR all F in OFF-set DO
            X, S=0;
            X=$M \oplus F$; //EXOR operation of two minterms
            S= Count number of "1" in X; // compute the difference
            T=T+S;
    Link value of T to M;
End FOR
Output = Sort ON-set by value of T;
Output = Output + Sort DC-set by value of T;
Output = Output + Sort OFF-set by value of T;
Return Output;
```

Fig. 4: Pseudo code for OFF based minterm ordering algorithm

Our algorithm depends on adjacency logic. Adjacent two minterms must be same but only one different bit. If we do bitwise "EXOR" operator to these minterms, outcome number should include only one "1" bit. Otherwise these minterms cannot create a PI. Same as checking ON minterms for adjacency, we can check these ON minterm for how isolated they are. An isolated minterm should be adjacent for one or more OFF minterms. Isolated minterm detection algorithm is presented in Fig. 4.

This algorithm compares all ON minterms with each of OFF minterms. Complexity of this algorithm is $O(n.m) \approx O(n^2)$, where n represents number of ON-minterms, and m represents number of OFF-minterms. Polynomial time complexity is the main drawback of this algorithm. Minimization algorithms' time complexity is far much higher. This algorithm works on easy to handle integers. In addition, our algorithm is appropriate for fast bitwise operators. Organizing the input file by ordering minterms, accelerates minimization process.

5.0 IMPLEMENTATION

The proposed method changes the input file. Same minimization process is done to both original input file and ordered input file. Then output files are compared with each other, in terms of running time and result quality. High quality result means more simplified functions which include less number of implicants.

Let us use our algorithm for simplification of a basic function f given in (1). On minterms for function f are {1, 3, 4, 5, 8, 9, 13, 15} and Off minterms are {0, 2, 6, 7, 10, 11, 12, 14}.

The proposed algorithm picks an ON minterm, lets say "0001", then compares with Off minterms by XOR operation. Number of "1"s in output numbers are isolation level of the minterm. This computation is given in Table 2. Isolation levels of all ON minterms are given in Table 3.

Table 2: Computation Table for Minterm "0001"

ON Minterm	OFF Minterms	ON $\oplus$ OFF	# of "1"s
0001	0000	0001	1
0001	0010	0011	2
0001	0110	0111	3
0001	0111	0110	2
0001	1010	1011	3
0001	1011	1010	2
0001	1100	1101	3
0001	1110	1111	4
		Isolation Level:	20

Table 3: Isolation Levels of On Minterms.

On Minterms of f		Isolation Level
1	0001	20
3	0011	16
4	0100	16
5	0101	20
8	1000	16
9	1001	20
13	1101	20
15	1111	16

As seen from the Table 3, the proposed algorithm found isolated minterms as S_i = {0011, 0100, 1000, 1111} which have lowest isolation level score.

Let us start simplification with minterm 0011$\in S_i$. Covering implicant for this minterm is PI_1(0011) = 00x1. Consequentially, remaining minterms are:

$G_{ON} = G_{ON}$ # PI_1= {0001, 0011, 0100, 0101, 1000, 1001, 1101, 1111} # PI_1
= {0100, 0101, 1000, 1001, 1101, 1111}.
We can proceed simplification by minterm 0100$\in S_i$. Covering implicant for this minterm is PI_2(0100) = 010x. Consequentially, remaining minterms are:

$G_{ON} = G_{ON}$ # PI_2= {1000, 1001, 1101, 1111}.

Initial isolated minterms are 1111, 1000$\in$ S_i. Covering implicants for these minterms are $PI_3(1111) = 11x1$ and $PI_4(1000) = 100x$. Consequentially,

$G_{ON} = G_{ON}$ # $\{PI_3, PI_4\}$ = {1000, 1001, 1101, 1111} # {11x1, 100x} = $\varnothing$.

As there is no uncovered minterms in G_{ON}, simplification process is completed. Simplified output function is
G_{PI} = {00x1, 010x, 11x1, 100x}.

The example above reveals that minterm selection plays a very crucial role in simplification process. Relatively centered minterms are easy to cover but it is difficult to find real *PI*. Finding PIs for isolated minterms not only helps to find real PI, but also make the remaining function easier. By ordering minterms in advance, simplification process worked 4 times instead of 5. This means simplification time is decreased by 20%. In addition, PI decision process is not needed.

6.0 EXPERIMENTAL RESULTS

This section presents experimental results. Minterm selection and function simplification algorithms are implemented in C#. All experiments were done on a PC having i5 1.7 GHz processor, 8 GB RAM and running Windows 8.

Random minterm selection has desired (*n*) time complexity given in (6), where n is the number of ON minterms. As the proposed method compared by multiplying two minterm groups, it has (*m.n*) time complexity where m is the number of OFF minterms given in (7). If we assume that number of ON and OFF minterms are roughly equal in average, complexity of the proposed method is quadratic.

$T(n)_{rdm} = O(n)$ (6)
$T(n)_{iso} = O(m.n) \approx O(n^2)$ (7)

Absolutely, random minterm selection is faster. But OFF based ordering of minterms makes simplification process easier. Minimum covering in logic function simplification problem is NP-hard [37]. Number of possible prime implicants for a function *f* is exponential as given in (8), where *m* is the number of minterms. Worst case complexity of an exact solution procedure is double exponential [37-41].

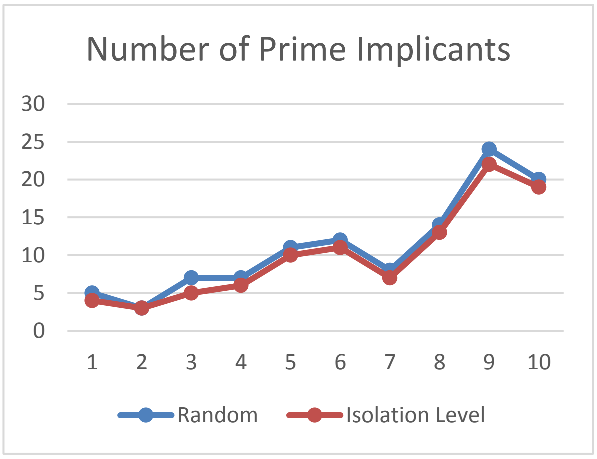
$p(f) \leq 2^{m(f)} - 1$ (8)

Table 4: Analysis of Simplification Results for Random and OFF based ordered files

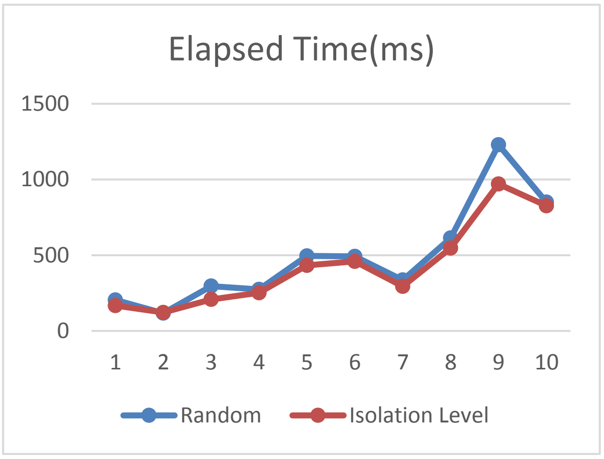
Input File	# of Input Variables	# of ON minterm	# of OFF minterm	# of P.I.		Elapsed Time (ms)	
				Random	OFF based	Random	OFF based
1	4	8	6	5	4	204	167
2		6	6	3	3	117	122
3	5	12	13	7	5	296	208
4		11	21	7	6	274	252
5	6	22	31	11	10	495	433
6		21	31	12	11	492	459
7	7	17	43	8	7	336	294
8		28	49	14	13	613	547
9	8	49	96	24	22	1229	970
10		45	92	20	19	850	826
Total		**219**	**388**	**111**	**100**	**4906**	**4278**

OFF based ordering of minterms spends time to organize but it saves much time in simplification process. 10 different functions consist of 219 ON minterms and 388 OFF minterms, are simplified by both random selection and isolation level selection methods. Comparison of results is given in Table 4.

Results revealed that even though proposed method spends time, it helps simplification process both saving time and finding less number of implicants. The proposed method finds more simplified solutions including less number of implicants than random minterm selection. Both methods find the same results for only one function which consists of 4 variable. As seen on Fig. 5(a), the proposed method simplifies 219 minterms to 100 implicants, where random minterm selection finds 111 implicants. The proposed method finds 11% better results in average.



(a) Number of Prime Implicants



(b) Elapsed Time in Millisecond

Fig. 5: Random vs. Isolation Level Minimization Results

Elapsed time of simplification process for each function can be seen from Fig. 5(b). The proposed method simplifies 9 of the functions faster than random minterm selection rule. Especially, function 3 and function 9 is simplified 42.3% and 26.7% faster respectively. Because of the time complexity of simplification algorithm is directly proportional to number of implicants and these functions are more simplified compared to others. The proposed method simplified all functions 14.67% faster in average. These results are in parallel with complexity analysis of algorithms above. Although the proposed method spends time in quadratic complexity, it also saves time in simplification process which has exponential complexity.

Function 2 is a 4 variable function which has only 6 minterms. Reordering of 6 minterms does not help much for simplification of this function. As both methods output 3 implicants, proposed method could not save time in simplification method. On the contrary, it slightly increases solution time about %5.

7.0 CONCLUSION

The aim of this article is to introduce a new approach for helping function simplification process. In contrast to previous works, this approach uses OFF minterms for ordering minterms and finding isolated minterms. One of the benefits of engaging OFF minterms is taking DC minterms into consideration for simplification process, which leads to efficiently synthesis functions with "don't cares". Thus, we could find real isolated minterms resulting simpler and faster outputs. The proposed algorithm is tested on various benchmarks. The results revealed that although proposed algorithm spends some time in reordering process, it saves much time in simplification process. As part of future work, we would like to advance parallel algorithms for logic simplification that could efficiently work in today's multicore computers.

Acknowledgement: This study is supported by TUBITAK #1059B141500323 and the Coordinatorship of Selcuk University's Scientific Research Projects.

REFERENCES

[1] Ryan, J. A. (1996). Leibniz' binary system and Shao Yong's "Yijing". *Philosophy East and West, 46* (1), 59-90.

[2] Boole, G. (1847). *The mathematical analysis of logic.* Philosophical Library.

[3] Marquand, A. (1885). A new logical machine. *Proceedings of the American Academy of Arts and Sciences, John Wilson and Son,* 303-307.

[4] Jess, J. A. G. (2000). Designing electronic engines with electronic engines: 40 years of bootstrapping of a technology upon itself. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19*(12), 1404-1427.

[5] Shannon, C. E. (1938). A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers, 57*(12), 713-723.

[6] Roy, S., Bhunia, C. T. (2015). On synthesis of combinational logic circuits. *International Journal of Computer Applications, 127*(1).

[7] Su, G., Wei, D., Varshney, K. R., & Malioutov, D. M. (2015). Interpretable two-level boolean rule learning for classification. *arXiv preprint arXiv:1511.07361*.

[8] Yavuz, E., Yazıcı, R., Kasapbaşı, M. C., & Yamaç, E. (2015). A chaos-based image encryption algorithm with simple logical functions. *Computers and Electrical Engineering 1–13.*

[9] Macia, J., Manzoni, R., Conde, N., Urrios, A., Nadal, E., Solé, R., & Posas, F. (2016). Implementation of complex biological logic circuits using spatially distributed multicellular consortia, *PLOS Computational Biology*.

[10] Hacıbeyoğlu, M., Başçiftçi, F., & Kahramanlı, Ş. (2011). A logic method for efficient reduction of the space complexity of the attribute reduction problem. *Turkish Journal of Electrical Engineering & Computer Sciences, 19*(4), 643-656.

[11] Başçiftçi, F., Eldem, A. (2013), Using reduced-rule base with expert system for the diagnosis of disease in hypertension. *Medical & Biological Engineering & Computing, 51*(12), 1287-1293.

[12] Karnaugh, M. (1953). The map method for synthesis of combinational logic circuits. *Transactions of the Communication and Electronics, American Institute of Electrical Engineers, 72* (5), 593-599.

[13] Roy, S., Bhunia, C. T. (2014). Constraints analysis for minimization of multiple inputs logic programming. *Elsevier Proc. of International Conference on Signal and Speech Processing (ICSSP-14),* 61-64, Kollam, India.

[14] Nowick, S. M., Dill, D. L. (1995). Exact two-level minimization of hazard-free logic with multiple-input changes. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, 14* (8), 986-997.

[15] Veitch, E. W. (1952). A chart method for simplifying truth functions. *Proceedings of the 1952 ACM National Meeting,* 127-133, Pittsburgh.

[16] Quine, W. V. (1955). A way to simplify truth functions. *American Mathematical Monthly*, pp. 627-631.

[17] McCluskey, E. J. (1956). Minimization of boolean functions. *Bell System Technical Journal, 35*(6), 1417-1444.

[18] Başçiftçi, F. (2006). *Anahtarlama fonksiyonları için yerel basitleştirme algoritmaları*, SELÇUK GRADUATE SCHOOL OF NATURAL SCIENCES, KONYA TURKEY.

[19] Dueck, G. W. (1988). *Algorithms for the Minimization of Binary and Multiple-Valued Logic Functions.* GRADUATE SCHOOL OF MANITOBA.

[20] Slagle, J. R., Chang C. L., Lee, R. C. T. (1970). A new algorithm for generating prime implicants. *IEEE Transactions on Computers, 100*(4), 304-310.

[21] Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers, 100*(8), 677-691.

[22] Roth, J. P. (1958). Algebraic topological methods for the synthesis of switching systems. *Transactions of the American Mathematical Society, 88*(2), 301-326.

[23] Wang, Y. M. (1989). *Truncated sum MVL minimization using the Neighborhood Decoupling Algorithm.* NAVAL POSTGRADUATE SCHOOL MONTEREY CA.

[24] Tirumalai, P. P., Butler, J. T. (1991). Simplification algorithms for multiple-valued programmable logic arrays. *IEEE Transactions on Computers, 40*(2), 167-177.

[25] Giovanni, D. M. (1994). *Synthesis and optimization of digital circuits.* Mccraw-Hill Higher Education.

[26] Majumder, A., Chowdhury, B., Mondai, A. J., & Jain, K. (2015, January). Investigation on Quine McCluskey method: A decimal manipulation based novel approach for the minimization of Boolean function. In *Electronic Design, Computer Networks & Automated Verification (EDCAV), 2015 International Conference on* (pp. 18-22). IEEE.

[27] Dueck, G. W., Miller, D. M. (1987). A direct cover MVL simplification using the truncated sum. *Proc. 17th International Symposium on Multiple-Valued Logic*, 221-227.

[28] Başçiftçi F., Kahramanlı Ş. (2011). Fast computation of the prime implicants by exact direct-cover algorithm based on the new partial ordering operation rule. *Advances in Engineering Software, 42*, 316-321.

[29] Pomper G., Armstrong, J. R. (1981). Representation of multivalued functions using the direct cover method, *IEEE Transactions on Computers, 100*(9), 674-679.

[30] Besslich, P. W. (1986). Heuristic minimization of MVL functions: a direct cover approach. *IEEE Transactions On Computers, 100*(2), 134-144.

[31] Tirumalai, P., Butler, J. T. (1991). Analysis of minimization algorithms for multiple valued programmable logic arrays, *IEEE Transactions on Computers, 40*(2), 167-177.

[32] Yang, C., Wang, Y. M. (1990). A neighborhood decoupling algorithm for truncated sum minimization, *Proceedings of the Twentieth International Symposium on Multiple-Valued Logic*, Charlotte, NC, 153-160.

[33] Bernasconi, A., Ciriani, V., Fišer, P. Trucco, G., (2012). Weighted don't cares. In Proc. of 10th International Workshop on Boolean Problems, 123-130.

[34] Fiˇser, P., Hlaviˇcka, J. (2012). Boom — a heuristic boolean minimizer. *Computing and Informatics, 22*(1), 19–51.

[35] Abd-El-Barr, M., Sarif, B. A. B., (2007). Weighted and ordered direct cover algorithms for minimization of MVL functions, *37th International Symposium on Multiple-Valued Logic (ISMVL'07)*, Oslo, 48.

[36] Abd-El-Barr, M., Khan, E. A., (2013). Improved direct cover heuristic algorithms for synthesis of multiple-valued logic functions, *International Journal of Electronics, 101(2)*, 271-286.

[37] McMULLEN, C., SHEARER, J. (1986). Prime implicants, minimum covers, and the complexity of logic simplification, *IEEE Transactions on Computers, C-35*(8), 761-762.

[38] Akar H., Başçiftçi F. (2016). Parallelised algorithm of isolated minterm detection for logic function simplification, *4th International Symposium on Innovative Technologies in Engineering and Science*, 1006-1014.

[39] Akar H., Başçiftçi F., Uğuz H. (2013). Paralel ve sıralı brute force algoritmasının karşılaştırılması, *Akademik Bilişim Konferansı*, 923-927.

[40] Akar H., Başçiftçi F. (2020). Smart minterm ordering and accumulation approach for insignificant function minimization, *Ain Shams Engineering Journal*, https://doi.org/10.1016/j.asej.2020.04.003.

[41] Akar H. (2020). Finding isolated minterms in logic functions and developing an efficient simplification algorithm (Doctoral dissertation), *Retrieved from tez.yok.gov.tr* (630786).