Finding Isolated Minterms in Simplification of Logic Functions

Fatih Başçiftçi, and Hakan Akar

***Abstract*—**The researches in different areas were carried out about logic expressions for years. Today, the researches related with logic expressions continue on such areas as health, computer architecture, cryptography, coding systems, quantum circuits. The simplification of logic expressions have great importance to use less material, to obtain a more concise statements or to use a more efficient memory. To be find out the most simplified version of logic expressions, it is important to begin to simplifying from which minterm. In this study, the researches explicated to determine the isolated minterms in logic minimization, among these researches, Besslich and Dueck & Miller algorithms were coded in C#. Algorithms were tested in random 10 functions. Program using Besslich algorithm yielded successful results in 7 functions, where program using Dueck & Miller algorithm yielded 9. Programs using Besslich and Dueck & Miller algorithms calculated functions in 4.5 and 5.1 seconds respectively using approximately 26 Megabyte memories. At the end of the research, recommendations were made in order to obtain more accurate results.

***Keywords*—**Algorithms, Heuristic Minimization, Isolated Minterms, Logic minimization, Weighted minterms.

I. Introduction

THE basis of digital electronics is logic expressions. Logic expressions can be single-valued (*0-1*), or it may be multiple valued. Both logic expressions are widely in use today. During computer technology evolves, the logic expressions get more complicated and it means increase of spent time, used memory and implementing cost. Accelerating of logic expressions' complexity so simplification of the expressions is becoming even more important.

Logic expressions can be expressed in two ways as the sum of products and product of sums [1]. If different values in parenthesis which are before summed then the parentheses with each other multiplying so this expression was written in the form of *Product of Sums* (*POS*). If the logic expressions by multiplying each other, then these multiplies are summed with each other's so this is called as *Sum of Products* (*SOP*) [2].

Over the decades several studies are carried out about logic expressions. Today, the logic synthesis has an important role in different areas, such as cryptology, health, video compression. The simplification of logic expressions has a great importance to use less material, to obtain more concise statements or to use a more efficient memory. The wide range of algorithms and programs were developed to simplify logic expressions [3]–[5]. Four basic methods can be used to simplify logic expressions. These are *Karnaugh map* (*Kmap*), mathematical simplification, the table method and tree method.

On the second part of this paper, the isolated minterms have been identified, previous studies have been summarized. On the third part, weighted isolated minterm application and neighborhood related application, which are developed as parallel to Besslich and Dueck & Miller algorithms, have been described. The results of the applications in terms of running time and memory usage were given in the third part. On the fourth part, the results and recommendations are given.

Fatih Başçiftçi is with the Faculty of Technology, Selcuk University, Konya, 42003, TURKEY.

Hakan Akar, is with the Department of Informatics, Akdeniz University, Antalya, 07059, TURKEY.

This work is supported by the Coordinatorship of Selcuk University's and Akdeniz University's Scientific Research Projects.

II. Isolated Minterms

A. What is Isolated Minterm?

In which minterm to start simplification of logic expressions has great importance, which method is used for simplification. If to start the simplification from the relatively centered minterm, the result cannot be correct, it means the simplest result cannot be obtained. However, if relatively farthest minterms have been simplified first, than the centered minterms could be included so it cannot be needed to simplification of these. Thus the most simplified result can be obtained. There is an example of on-set consisting of eight minterms in set (1) and *Kmaps* which belongs to it in Fig. 1. The dots on the map show the on minterms, *prime implicants* (*PI*) which involves the minterms within ovals.

$$S_{ON}=\{0001,0101,0111,0110,1100,1101,1111,1011\} \quad (1)$$

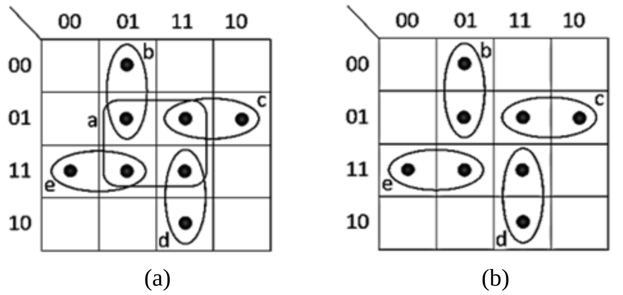

Fig. 1 (a) The worst and, (b) The best SOP statements

If we start from the center to cover minterms as on the Fig. 1 (a), then *a*, *b*, *c*, *d*, *e* *PI*s have been occurred. *S*={x1x1+0x01+011x+1x11+110x} is the worst *SOP* expression for the above map. If we start from the outer minterms(isolated minterms) to cover as on the Fig. 1 (b), then *b*, *c*, *d*, *e* *PI*s have been occurred. And there is no need to use a *PI*. *S*={0x01+011x+1x11+110x} is the best *SOP* expression for the above map.

As can be seen from Fig. 1, it's important to start covering with right minterm to achieve the simplest result. The minterm which have no neighbors around also called isolated minterm.

B. Finding Isolated Minterm

All kind of direct cover algorithms consist of two stages. Firstly, a minterm is selected, and then a *PI*, that covers the minterm, is found [6]. Direct cover algorithm is given in Algorithm 1. *M* represents function set consist of *ON* minterms, whereas *S* represent simplified output function.

```
Program Direct Cover
Begin
{ S ← ∅
  While (M ≠ ∅) do
  { Pick minterm α from the set M
    Find imlicant $I_\alpha$ covers the minterm α
    S ← $I_\alpha$ + S
    Remove α from the set M
  }
}
```

Algorithm 1. Direct Cover Pseudo Code

In simplification of logic expressions, it is important to select the appropriate minterms and *PI* to get the simplest expression. While some algorithms are choosing either minterm or *PI* randomly but some choose it according to some criteria. In the Pomper and Armstrong algorithm; the minterm has been selected randomly but from among the *PI*s which covers this minterm, the one is selected which is covering the maximum quantities of minterms. In this algorithm, all of the PIs which covers the minterm have been determined and the one is selected which creates maximum quantities of do not care minterms [7]. Pomper and Armstrong algorithm is given in Algorithm 2, maximum minterm covering implicant (max_Implicant) algorithm is given in Algorithm 3.

In Besslich Algorithm, each minterm has a weight. The weights of the central minterms are much more but the weight of the outside isolated minterms are less. In this algorithm, the covering process of the minterms starts from the isolated ones. All *PI*s which covers the minterm are determined in the first phase. Then effectiveness factor is calculated for each *PI* by dividing the cost of each number of minterm covered by PI. The covering process is completed by selecting the most effective *PI* [8]. The code of direct cover method given in algorithm 1 is valid in Besslich algorithm. But, covering the minterms is not a random process in this case, covering process starts from the most isolated minterm calculated in Algorithm 4.

```
Program Pomper and Armstrong
Begin
{ F=input function
  S=Solution Set=Ø
  While ( F ) do
  { α=Random minterm from the set F;
    I=max_Implicant(α);
    S=S+I;
    F=F-α;
  }
}
```

Algorithm 2. Pomper and Armstrong Pseudo Code

```
Function P_A_Max_Imp (α)
Begin
{ Imp_size = -∞;
  Covered_Minterm= -∞;
  For (every I covering α) do
  { I_covered_min=Number of Minterm drived 0 or DC;
    I_size=Implikant size;
    If(I_covered_min>Covered_Minterm)
    { maximal_Implicant(α)=I;
      Imp_size=I_size;
      Covered_Minterm=I_covered_min;
    }
  } return maximal_Implicant
}
```

Algorithm 3. Maximal Implicant Determination Pseudo Code

```
Function Besslich_isolated (F)
Begin
{ lowest_wt = ∞ ;   α = 00…00;
  for (every minterm(β) except for 0 or DC)
  { wt(β)=$\sum_{1\leq m\leq n}(\beta_m-\gamma_m)$
    If ( wt(β) <lowest_wt )
    { α=β;
      lowest_wt=wt(β)
    }
  } return lowest_wt
}
```

Algorithm 4. Besslich Algorithm, Minterm Selection Pseudo Code

In Dueck, and Miller algorithm; the isolation factor is determined for each minterm. The isolation factor is inversely proportion with the addition of number of neighbors of a minterm and the numbers of direction of the neighbors located. The covering process starts with high isolation factored minterms which has no neighbors and goes on with the minterms which have the lower isolation factor coefficient. Minterm selection code is given in algorithm 5. And then all *PI*s which cover all of these minterms are created. Relative Break Count (*RBC*) is calculated for each *PI*. This calculation helps to find out how much the *PI* simplifies the expression. As given in the algorithm 6, the most simplifying *PI* is selected among remaining *PI*s [9].

```
Function Dueck_and_Miller (F)
Begin
{ α = 00…00;  CFmin = ∞;
  While (F function) do
  { CF(β) = Number of Neighbors(β)+Direction of
            Neighbors(β)
    If (CFmin>CF(β))
    { CFmin = CF(β);
      α = β;
    }
    F = F - β;
  } return α
}
```

Algorithm 5. Dueck & Miller, Minterm Selection Pseudo Code

```
Function Dueck_Miller_Imp_Sel (F)
Begin
{ rbc(I) = 0;
  for(every minterm α covered by implicant I) do
```

```
    { if(implicant I covers minterm β near α)
      rbc(I) = rbc(I) - 1;
      if(implicant I does not cover minterm β
         before or after minterm α)
      rbc(I) = rbc(I) + 1;
    }
    cur_rbc = ∞;
    for(every implicant I covering minterm α)
    do
    { if ( rbc(I) <cur_rbc )
      { best = I;
        cur_rbc = rbc(I);
      }
    } return best
}
```

Algorithm 6. Dueck & Miller, Implicant Selection Pseudo Code

According to the study by Tirumalai and Butler's; none of the algorithms cannot have a fully superiority against others for all of the functions. The Pomper and Armstrong Algorithm provide better results than the random selection. Each of Dueck & Miller and Besslich algorithms provides better results than Pomper and Armstrong Algorithms [9]. Summary of the algorithms is given in Table I.

TABLE I
ALGORITHM SUMMARIES

Algorithm	Minterm Selection	Implicant Selection
Pomper and Armstrong	Random	Drives most minterm *DC*
Besslich	Lightest (most isolated)	Drives most minterm *DC*
Dueck and Miller	Biggest isolation factor	Most simplifies remaining function
Boom	Random	Drives most minterm *DC*

According to the Boom Algorithm, all of the *PI*s covering minterms are determined and it's accumulated in the *PI* pool. There is no matter of starting from which minterm because all PIs are accumulated in the pool. Thus a duplicate *PI* does not occur. The accumulation of *PI* provides the program run faster and uses less memory. Coverage problem is solved in the following order at four stages [10]:

1) Select *PI* which covers maximum numbers of minterms.
2) Select *PI* which covers minterms, covering is difficult.
3) Select *PI* has lowest cost (involved less literals).
4) If there is still much possibility than select randomly.

III. Weighted And Neighborhood Related Isolated Minterm Application

In this study, finding the isolated minterm according to Besslich and Dueck & Miller algorithms is performed in C# .Net program. Programs which based on Besslich and Dueck & Miller algorithms are called "Weighted Isolated Minterm Application" and "Neighborhood Related Isolated Minterm Application" respectively. In the developed program, the user can determine the number of variables. In the program, there is no limit for the number of variables theoretically. User clicks "Create" button to generate *Karnaugh Map*, then clicks checkboxes which are allocated for minterms.

When user click calculate button, program calculates the weights of all of the minterms and marked the most isolated minterms. By clicking the "Reset" button, all of the checkboxes can be cleaned up and, re-entry can be made.

A. Weighted Isolated Minterm Algorithm

According to algorithm of weighted isolated minterm; all of the minterms are effecting each other. The effect of two minterms to the each other is inversely proportional with the distance between them. When the minterms are farther from each other, they have less effect on each other. In a matrix with the variable of *n*; the coordinates of the minterms *A* and *B* are A_x, A_y and B_x, B_y. The distance on axis x between the minterms of *A* and *B* is calculated with the formula (2).

$$S_x = |A_x - B_x| \tag{2}$$

Equation (2) calculates the distance between two minterms on the axis of *x*. If the distance between two minterms on the axis of *x* (S_x) is less than the half of variable number (*n*/2) so it does not change but if its more; then the weightiness is increased as much as the more part. Two minterms can be far away from each other as much as the half of number of variables. For example; on a 9x9 *Kmap*, two minterms can be far away from each other mostly 4 units on the *x* axis. If the difference between is more than 4; it means the minterms are closer to each other from the other end of map. This situation has been mathematically described in (3).

$$S_x = \begin{cases} |A_x - B_x| & |A_x - B_x| \le \frac{n}{2} \\ n - |A_x - B_x| & |A_x - B_x| > \frac{n}{2} \end{cases} \tag{3}$$

The same calculation which has been done for axis *x* also done for axis *y*. Absolute distance is calculated by summing the distance of the axis *x* and axis *y*.

$$S = S_x + S_y \tag{4}$$

To save the weightiness of each minterm; a 2 dimensional array as long as *n* is used. The minterms are coded according to their location on the axis *x* and axis *y*. Each minterm has a unit weight as *n* to itself. To find out the weightiness of the minterms (*E*) to the other minterms; the distance between them (*S*) is subtracting from the number of variable (*n*) (5).

$$E = n - S \tag{5}$$

Four nested loops were used to find the effecting weightiness of self minterm and all other minterms. The first two loops of *for* creates the coordinates of x and y of minterm (A_{min}) whereas second two loops of *for* creates the axes of x and y of minterm (B_{min}). Effecting weightiness of minterms are kept in a two dimensional array.

B. Neighborhood Related Isolated Minterm Algorithm

There are two main calculations that should be done in Neighborhood related isolated minterm algorithm. One is size of neighborhood (KG_A), the other is directions of neighbors (KY_A). In order to calculate isolation factor (IF_A) of minterm *A*, both calculation results should be summed for this minterm.

$$IF_A = KG_A + KY_A \tag{6}$$

A recursive function is used to find the size of a minterm neighborhood. Thus, the program needs less memory and computes in a shorter time [11]. The pseudo code for this algorithm is given in Algorithm 7. *xPos* is the position of minterm *A* in the axes x, where *yPos* is the position of minterm *A* in the axes *y*.

```
Komsu1=0
Function Calculate(xPos,yPos)
Begin
{ Komsu1++
  if(variable are in range & minterm is not 0)
  { Calculate(xPos+1, yPos)
    Calculate(xPos, yPos+1)
    Calculate(xPos-1, yPos)
    Calculate(xPos, yPos-1)
  }
  return true
}
```

Algorithm 7. Computing Neighborhood Algorithm

The function which pseudo code is given in Algorithm 8, determines the directions of neighborhoods.

```
Function Neighbours(xPos, yPos)
Begin
{ Komsu2=0
  if (xPos+1, yPos) komsu2++
  if (xPos-1, yPos) komsu2++
  if (xPos, yPos+1) komsu2++
  if (xPos, yPos-1) komsu2++
  return komsu2
}
```

Algorithm 8. Computing Neighborhood Directions

Komsu1 variable calculated in Algorithm 7, browses through all neighborhood minterms recursively, increase the Komsu1 variable in each occurrence. *Komsu1* variable keeps the size of whole neighborhood of a minterm.

Komsu2 variable used in algorithm 8, keeps whether a minterm has neighbor minterms on top, left, right or down. *Komsu2* variable is saving data about how many different directions do the neighbors are located.

C. Strengths and Weaknesses of Weighted and Neighborhood Related Isolated Minterm Applications

Weighted isolated minterm application makes the calculation according to the proximity of the minterms to others. This program can measure the distance of minterms which looks like far away to each other on the map but they are close to each other on the other side of the map. There is an "*if* statement" in the algorithm that checks this possibility. X_A represents minterm *A*'s position on the axes *x*, *n* represents variable number in the function, *L* represents the distance between two minterms on axes *x*, in the Algorithm 9.

```
Function Compute_Distance($X_A$, $X_B$)
Begin
{If (($X_A$ - $X_B$)>(n/2))
   Then L=$X_A$-$X_B$-(n/2)
   Else L=$X_A$-$X_B$
 Return L
}
```

Algorithm 9. Computing Distance Algorithm

The application is tested for 10 randomly created function using 100 variables (100x100 matrix) on a Win 8 computer which is capable of i5 processor, 4 gigabyte memory. Microsoft Process Explorer program is used to calculate memory usage and running time [12].

Isolated minterms are calculated on 10 functions whose minterm numbers are between 7 and 49. Off minterms' weights are not calculated with the help of control statement (*if* statement) which lies inside the loops. Recalculating the checked minterms does not spoil the result, but it consumes much memory and CPU time [13]. Weighted Isolated minterm application detected 7 over 10 functions successfully whereas Neighborhood Related Isolated Minterm application detected 9 over 10 functions successfully. Results are given at Table II.

The average number of minterms is 23 in tested 10 functions. Weighted isolated minterm application consumed 25.909 MB memory, found the results in 4.543 seconds in average. Neighborhood related isolated minterm application consumed 25.878 MB memory and found the results in 5.117 seconds in average. Weighted isolated minterm application's running times are between 1.4–8.3 seconds whereas neighborhood related isolated minterm application's running times are between 1.75–8.875 seconds. Weighted isolated minterm application's memory usage is between 22.336–32.312 MB whereas neighborhood related isolated minterm application's memory usages are between 23.096–28.324 MB.

Because of the application is making the weight calculation according to their positions relative to each other, it can be seen as non-isolated minterm which has no neighbors but has a central location. To find out the isolated minterms; it's important where the minterms located and whether they have or not neighbors and if they have, then in which directions these neighbors. The weighted isolated minterm application does not consider the neighborhood relations of minterms.

Neighborhood related isolated minterm application detects the isolated minterm successfully in most times (90% in our sample tests). But it is not successful to distinguish isolated ones among remaining minterms. Minterms in the same group have very close isolation factors. Thus neighborhood related isolated minterm application detects the most isolated minterm successfully, but this application could not rank the remaining minterms isolation factor successfully.

TABLE II
ALGORITHM SUMMARIES

Function No	Minterm	Running Time (Sec.)		Memory Usage (MB)		Results	
	Number	Weighted	N.Related	Weighted	N.Related	Weighted	N.Related
Function 1	10	1.781	2.750	22.472	24.868	Successful	Successful
Function 2	15	4.359	5.531	28.916	26.436	Successful	Successful
Function 3	14	4.562	2.390	24.440	23.096	Successful	Successful
Function 4	22	1.859	2.531	22.372	25.124	Unsuccessful	Successful
Function 5	46	8.265	7.578	32.312	28.324	Successful	Unsuccessful
Function 6	26	6.812	6.828	30.028	25.268	Successful	Successful
Function 7	21	6.156	6.281	23.864	25.544	Unsuccessful	Successful
Function 8	21	4.953	8.875	24.632	28.264	Successful	Successful
Function 9	7	1.359	1.750	22.336	23.848	Unsuccessful	Successful
Function 10	49	5.328	6.656	27.720	28.024	Successful	Successful
AVERAGE	23.1	4.543	5.117	25.909	25.878	7 SUCCESSFUL	9 SUCCESSFUL

IV. CONCLUSION

In this study, the importance of isolated minterms for simplification of logic expressions and previously developed algorithms have been explained and a similar of Besslich and Dueck & Miller algorithms which have successful results, was coded in *C#* program. The developed algorithms are explained and its strengths and weaknesses are described. Randomly generated 10 functions are tested in the both programs. Weighted Isolated minterm application detected 7 over 10 functions successfully whereas Neighborhood Related Isolated Minterm application detected 9 over 10 functions successfully. The average statistics for memory usage and running time of applications are 26 Megabytes and 4.5 seconds respectively. It's observed that the developed program can easily calculate the proximity distance of the minterms to each other's but it cannot detect the neighborhood relations of the minterms. For this reason the program cannot find the correct result in situations where the neighborhood relations are important then the location. Neighborhood related isolated minterm application detects the most isolated minterm successfully, but it cannot detect the rest minterms successfully.

In the future researches; hybrid systems can be improved which uses both weighted and neighborhood related minterm application. Furthermore, *fuzzy* systems can be used for finding isolated minterm and imlicants [14].

REFERENCES

[1] Y. Crama, and P. L. Hammer, *Boolean Functions; Theory, Algorithms, and Applications.* Cambridge University Press, 2012.

[2] M. Altun, and M. D. Riedel, "Logic Synthesis for Switching Lattices," *Transactions on Computers*, vol. 61, no. 11, pp. 1588–1600, Nov. 2012. http://dx.doi.org/10.1109/TC.2011.170

[3] M. G. A. Martins, R. P. Ribas, and A. I. Reis, "Functional Composition: A New Paradigm for Performing Logic Synthesis," in *Proc. 13th Int. Symposium on Quality Electronic Design*, California, 2012, pp. 236–242.

[4] F. Basçiftçi, and Ş. Kahramanlı, "Fast computation of determination of the prime implicants by a novel near minimum minimization method," *Turk Journal of Electrical Engineering & Computer Science*, vol. 18, no. 6, pp. 1041–1052, 2010.

[5] P. Fiˇser, and J. Hlaviˇcka, "Boom — A Heuristic Boolean Minimizer," *Computing and Informatics*, vol. 22, no. 1, pp. 19–51, 2003.

[6] Y. M. Wang, "Truncated Sum MVL Minimization Using the Neighborhood Decoupling Algorithm," Unpublished Master Thesis, Naval Postgraduate School, California, 1989.

[7] G. Pomper, and J. R. Armstrong, "Representation of Multivalued Functions Using the Direct Cover Method," *IEEE Transactions On Computers*, vol. C-30, no. 9, pp. 674–679, Sept. 1981. http://dx.doi.org/10.1109/TC.1981.1675867

[8] P. W. Besslich, "Heuristic Minimization of MVL Functions: A Direct Cover Approach," *IEEE Transactions on Computers*, vol. C-35, no. 2, pp. 134–144, Feb. 1986. http://dx.doi.org/10.1109/TC.1986.1676731

[9] P. Tirumalai, and J. T. Butler, "Minimization Algorithms for Multiple Valued Programmable Logic Arrays," *IEEE Transactions on Computers*, vol. 40, no. 2, pp. 167–177, Feb. 1991. http://dx.doi.org/10.1109/12.73587

[10] A. Bernasconi, V. Ciriani, P. Fišer, and G. Trucco, "Weighted Don't Cares," in *Proc. 10th International Workshop on Boolean Problems*, Freiberg, 2012, pp. 123–130.

[11] M. Burgin, "Super-Recursive Algorithms as a Tool for High Performance Computing," in *Proc. High Performance Computing Symposium*, San Diego, 1999, pp. 224–228.

[12] M. Russinovich, "TNProcess Explorer," *http://technet.microsoft.com/tr-tr/sysinternals/ bb89 6653.aspx*.

[13] G. W. Dueck, "Direct Cover MVL Minimization with Cost Tables," in *Proc. 22nd International Symposium on Multiple-Valued Logic*, Sendai, May 1992, pp. 58–65.

[14] M. Abd-El-Barr, "Hybrid Fuzzy Direct Cover Algorithm for Synthesis of Multiple-Valued Logic Functions," *IJCSI International Journal of Computer Science Issues*, vol. 8, issue 3, no. 2, pp. 158–166, May 2011.

[15] K. Datta, and I. Sengupta, "Applications of Reversible Logic in Cryptography and Coding Theory," in *Proc. 26th International Conference on VLSI Design*, India, 2013, pp. 66–67.

[16] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A Meet-in-the-Middle Algorithm for Fast Synthesis of Depth-Optimal Quantum Circuits," *Transactions on Computer – Aided Design of Integrated Circuits and Systems*, vol. 32, no. 6, pp. 818–830, June 2013.

[17] V. Rawat, R. P. Singh, M. Pawar, and R. K. Mishra, "Lossless Gray Image Compression Using Logic Minimization," *Recent Research in Science and Technology*, vol. 4, no. 1, pp. 14–18, Jan. 2012.

[18] F. Başçiftçi, "Anahtarlama Fonksiyonları İçin Yerel Basitleştirme Algoritmaları," Unpublished PhD. Thesis, Selcuk University, Konya, 2006.