NON-BINARY EVOLUTIONARY SYNTHESIS TECHNIQUES – A COMPARISON

Mostafa Abd-El-Barr
Department of Information Science
College of Computer Science and Engineering
Kuwait University
Safat 13060, Kuwait
mostafa.abdelbarr@gmail.com

ABSTRACT

Binary signaling is facing on-chip interconnection and functional density limitations. Beyond binary digital systems is one possible solution to such problems. Non-Binary circuits are currently used side-by-side with binary circuits in the realization of general and special purpose processors. A major challenge facing MVL is the enormous size of the functional search space. The fundamental Direct Cover (DC) algorithm for synthesis of MVL functions iteratively selects the next minterm to be covered and the appropriate implicant to cover it using a set of heuristically adopted criteria. Two evolutionary techniques have been used for synthesis of MVL functions. These are the Genetic algorithms (GAs) and Ant Colony Optimization (ACO). In this paper, we explain and compare these two evolutionary-based techniques. The algorithms used are simulated and tested using a benchmark consisting of 50000 randomly generated 2-variable 4-valued functions and a benchmark consisting of 50000 2-variable 5-valued randomly generated functions. The simulation results obtained showed that the average number of product terms (PTs) needed in the synthesis of a given MVL function using evolutionary techniques outperforms those obtained using conventional DC heuristics. Among the two techniques it is shown that the technique based on the ACO achieves the best results.

KEYWORDS

MVL functional synthesis, Direct-Cover algorithms (DC), Ant Colony Optimization (ACO), Genetic algorithms (GAs), non-binary digital signal processing (DSP).

1. INTRODUCTION

As the complexity of integrated circuits (ICs) increases, circuit implementations using binary logic show certain limitations, such as large layout area for on-chip interconnections and increased power consumption. Signal processing using multiple-valued logic (MVL) is carried out using more than two discrete logic levels. Recent advances in Very Large Scale Integration (VLSI) technology made it possible to fabricate more efficient MVL circuits using binary CMOS (Complementary Metal Oxide Semiconductor) circuits as well as nano-scale technologies [10][13][16][19] [47-48][53-54]. There have been successful hardware realizations of MVL circuits over the last three decades. These include a 200 MHz 54×54 Pipelined multiplier using 1.5 V-Supply Multiple-Valued MOS Current-Mode Circuit design using 0.8 μm achieving about 1.4 times faster than its corresponding binary multiplier [55], fundamental arithmetic operations such as addition, subtraction and multiplication using 4-valued current-mode VLSI technology [56][64], four-state Intel ROM chip achieving nearly 50% area reduction over its binary equivalent [46], a rule-based programmable 4-valued matching VLSI processor for high speed pattern matching, an associative 4-valued VLSI processor for database systems, and an RNS MVL image VLSI processor, an encryption Radix-4 VLSI processor, and robotic and machine learning [57][58].

The MVL functional synthesis problem is however more complex when compared with its binary counterpart. There are $r^{(r^n)} = 4^{(4^2)} = 4^{16}$ 2-variable 4-valued functions. The search space for finding optimal MVL function synthesis is enormous. Exact minimization of

MVL functions is prohibitively expensive. A number of heuristics for near minimal synthesis of MVL functions have been introduced [33][41][61][63]. These algorithms can be classified into five categories: iterative based [31], decomposition-based [1][4][5][11][27][40], decision diagram-based [21][23][29][30][34][42], algebraic-based [12][18][28][32], and direct cover-based [7][9][14]. In addition, there has been several attempts to synthesize MVL functions using evolutionary algorithms, such as Genetic algorithms (GAs) [44][45][46][49], Ant Colony (ACO) [50][51], and Simulated Annealing (SA) [59][60]. Fuzzy-based synthesis of MVL functions has also been reported in the literature [62]

Definition 1: An n-variable r-valued function $f(X)$ is defined as $f: R^n \rightarrow R$, where $R = \{0,1,\cdots,r-1\}$ is a set of r logic values with $r \geq 2$ and $X = \{x_1, x_2, \cdots, x_n\}$ is a set of r-valued n variables. □

Table 1 shows a number of MVL logic operators.

Definition 2: A *product term* (PT), $P(x_1, x_2, ..., x_n)$, is defined as the minimum of a set of *window literals* such that $P(x_1, x_2, ..., x_n) = c \bullet {}^{a_1,b_1}x_1 \, {}^{a_2,b_2}x_2 \, ..., \, {}^{a_n,b_n}x_n = \min(c, {}^{a_1,b_1}x_1, {}^{a_2,b_2}x_2, ..., {}^{a_n,b_n}x_n)$ where $a_i, b_i \in R$, $a_i \leq b_i$ and $c \in \{1,2,\ldots, r\text{-}1\}$ is called the value of the PT. □

Definition 3: An assignment of values to variables such that $x_1 = a_1, x_2 = a_2 = ..., x_n = a_n$, where $a_i \in \{0, 1, \ldots, r\text{-}1\}$, in an MVL function $f(x_1, x_2, ..., x_n)$ is called a *minterm*, iff: $f(x_1, x_2, ..., x_n) \neq 0$. □

A *minterm* is a special case of a *PT* for which $a_1 = b_1, a_2 = b_2 = ..., a_n = b_n$. If the value of a *minterm* is *r*, then it is considered as *don't care* and is represented as *d*.

Definition 4: An *implicant* of a function $f(x_1, x_2, ..., x_n)$, is a PT, $I(x_1, x_2, ..., x_n)$, such that $f(x_1, x_2, ..., x_n) \geq I(x_1, x_2, ..., x_n)$ for all assignments of x_i's. □

Fig. 1 shows an example of a 2-varaible 4-valued function. In this figure $1\bullet{}^0X_1{}^0\bullet{}^3X_2{}^3$, $2\bullet{}^0X_1{}^0\bullet{}^1X_2{}^1$ and $3\bullet{}^0X_1{}^0\bullet{}^2X_2{}^2$ are examples for minterms while $2\bullet{}^0X_1{}^1\bullet{}^1X_2{}^1$and $2\bullet{}^0X_1{}^1\bullet{}^1X_2{}^2$ are examples for *implicants*.

Table 1: Sample MVL operators.

Operator	The logic equation
Literal	${}^a x^b = \begin{cases} (r-1) & if\ (a \leq x \leq b) \\ 0 & otherwise \end{cases}$ where $a, b \in R$ and $a \leq b$
Tsum	$tsum(a_i, a_j) = a_i \oplus a_j = \begin{cases} a_i + a_j & if\ a_i + a_j < r-1 \\ r-1 & otherwise \end{cases}$ $a_i \in R$ and $\oplus$ represents the truncated sum.
MAX	$MAX(x_1, x_2) = \begin{cases} x_1 & \text{if } x_1 \geq x_2 \\ x_2 & \text{otherwise} \end{cases}$
MIN	$MIN(x_1, x_2) = \begin{cases} x_1 & \text{if } x_1 \leq x_2 \\ x_2 & \text{otherwise} \end{cases}$

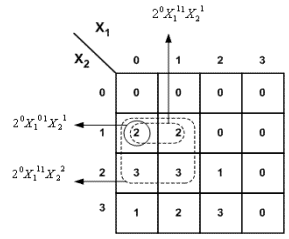


Fig. 1: A Tabular Representation of $f(X_1, X_2)$.

2. THE DIRECT COVER ALGORITHM

The Direct Cover (DC) techniques for synthesis of MVL functions consist of the following main steps:

1. Choose a minterm (see Definition 3),
2. Identify a suitable implicant (see Definition 4) that covers the chosen minterm,
3. Obtain a reduced function by removing the identified implicant, and
4. Repeat steps 1 to 3 until no more minterms remain uncovered.

The selection of appropriate minterms and the implicants covering them play an important role in obtaining less number of product terms to cover a given function. The Direct Cover approaches reported in the literature differ in the way appropriate minterms are chosen. They also differ in the way appropriate implicants are

identified. Different metrics to select minterms have been proposed in the literature. These are explained in the following subsections.

2.1. Minterm Selection

Table 2 summarizes the techniques used example for each.

Table 2: Minterm selection criteria.

Technique	The measure	Example
IW [3]	$w(\alpha,\beta) = 2^{[n(r-1)-D(\alpha,\beta)]}$. where D(α, β) = $\sum_{1\le m\le n} \mid i_m - j_m \mid$ $IW(\beta) = \sum_{\alpha\in Trans} Trans(\alpha)\times w(\alpha,\beta)$.	See Fig. 2.
CF [10]	$CF(\beta) = DEA_\beta * (r-1) + EA_\beta$	See Fig. 3.
CFN [8]	$CFN = EDC_\alpha * (r-1) + CMC_\alpha$	See Fig. 3.

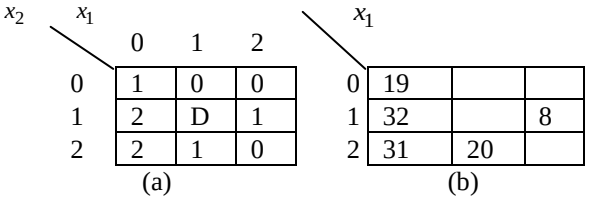
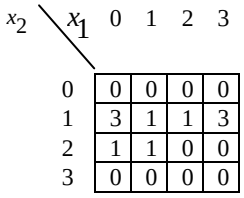


Fig. 2: Example IW calculation.



Minterms	DEA	DEA*(r-1)	EA	CF
$1\bullet^1X_1{}^1\bullet^1X_2{}^1$	2	6	5	11
$1\bullet^2X_1{}^2\bullet^1X_2{}^1$	1	3	3	6
$1\bullet^0X_1{}^0\bullet^2X_2{}^2$	2	6	3	9
$1\bullet^1X_1{}^1\bullet^2X_2{}^2$	2	6	3	9

Minterms	EDC	EDC*(r-1)	CMC	CFN
$1\bullet^1X_1{}^1\bullet^1X_2{}^1$	2	6	2	8
$1\bullet^2X_1{}^2\bullet^1X_2{}^1$	1	3	1	4
$1\bullet^0X_1{}^0\bullet^2X_2{}^2$	1	3	1	4
$1\bullet^1X_1{}^1\bullet^2X_2{}^2$	2	6	2	8
$3\bullet^0X_1{}^0\bullet^1X_2{}^1$	2	6	4	10
$3\bullet^3X_1{}^3\bullet^1X_2{}^1$	1	3	3	6

Fig. 3: Example computation of CFs and CFNs.

2.2. Implicant Selection

Relative Break Count (RBC): The Relative Break Count (RBC) is a measure of the degree to which a function is simplified if a specific implicant is chosen [10]. A "Break" is said to take place in a function if two adjacent minterms have different values. The number of breaks in the function is called the "*Break Count* (BC)". The removal of an implicant from a given function (during the coverage process) may result in a change in the number of BCs. A change in the number of BCs is called "*Relative Break Count* (RBC)". A RBC is positive if the removal of an implicant introduces breaks in the function. It is negative if the removal of an implicant eliminates some breaks from the function. The fewer the number of breaks in a given MVL function the less number of implicants needed to cover the function. The larger the number of breaks in a given MVL function the larger the number of implicants needed to cover the function.

Neighborhood Relative Count (NRC): The Neighborhood Relative Count (NRC) measures the degree of the strength to which a given implicant couples with its neighbors [8]. The removal of an implicant is equivalent to breaking the coupling between that implicant and its neighbors. Candidate implicant for removal should have the smallest coupling strength with its neighbors, i.e. the lowest NRC. In case of a tie, the implicant which covers the largest number of minterms is selected. The NRC evaluates implicants in order of their sizes.

Largest Number of Minterms Reduced to Zero (LRZ): The LRZ is a metric that counts the number of zeros or don't cares an implicant can produce if it is selected for removal from a given function [3][15]. A don't care will be produced if *c* of the removed minterm is equal to *r-1*; otherwise, 0 will be produced.

The quality of solution provided by DC algorithms is determined by the way they select minterm and implicants covering it. Unfortunately, there is no general agreement on which criteria is the best for a given MVL

function. An attempt has been made in [24] to analyze the different criteria used by combining different selected criteria and compare the results obtained in terms of the number of implicants needed to cover a given MVL given function.

3. GENETIC ALGORITHMS

Genetic Algorithms (GAs) are powerful, domain-independent meta-heuristic, search techniques. A GA emulates the natural process of evolution to perform an efficient and systematic search in the solution space to reach a possible optimum [18]. Genetic Algorithms operate on a population (set) of individuals (solutions) encoded strings. These strings represent points in the search space. In each iteration, referred to as a *generation*, a new set of strings that represent solutions (called *offspring*) is created by crossing some of the strings of the current generation. Occasionally, new characteristics are injected to add diversity. This can be done using for example mutations. GAs combines information exchange along with survival of the fittest among individuals to conduct the search. Using GA, a given MVL function is represented as a string of chromosomes. Each chromosome consists of several Genes, which represent a product term. Each gene consists of five integer attributes. The first attribute represents the value of the constant of the corresponding product term. The 2nd and 3rd attributes are for the window's boundary of product term for the first variable X_1. The 4th and 5th attributes are for the window's boundary for the second variable, X_2. For example, the first product term of function F shown in Fig. 4(c) ($1\bullet^0X_1{}^0\bullet^0X_2{}^2$) is represented by 10002. It is also shown that function F can be represented by two different chromosomes (see Fig. 4 (c) and Fig. 4 (d)).

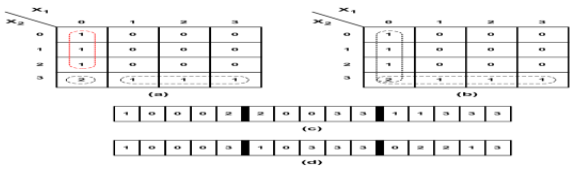


Fig. 4:Two different chromosome representations.

3.1. Fitness Function Calculation

Two goals must be reflected in the fitness function calculation. These are correctness and optimality. Correctness deals with the functionality of the representation. A correct expression will cover all minterms and only those minterms of the function. Optimality deals with the quality of solution, i.e., the minimum number of implicants required to implement the function. We divide the fitness function into two parts. The first part is called *Functional Fitness*, F_f. The second part of the fitness function is called *Objective Fitness*, F_o. These are explained below.

Functional fitness: Functional fitness is obtained by comparing the truth table of the given function with that of the obtained one. The value of the functional fitness is equal to the number of minterm matching (hits) between the two truth tables less the number of mismatch between the two truth tables. This can be formulated as follows: $F_f = N_h - N_m$ where N_h = Number of hits and N_m = Number of misses.

The truth table of a given function is obtained by scanning the tabular format of the function row-wise and transforms it into a string of integer. For example, the function shown in Fig. 4will be represented by the string "1000100010002111".

The truth table of the obtained function is synthesized by adding the truth table of all of its product terms. Consider, for example, the chromosome X= "101111002310233" as the one obtained for the function shown in Fig. 4. The chromosome consists of three genes (three product terms). Thus, the truth table of the obtained function is:

- G_1 = "10111". Truth table of G_1= "0000110000000000". Truth table of obtained function so far is "0000110000000000".
- G_2 = "10023". Truth table of G_2= "0000000010001000". Truth table of obtained function will become "0000110000000000" ⊕ "0000000010001000" = "0000110010001000".

- G_3 = "10233". Truth table of G_3 = "0000000000001110". Truth table of obtained function will become "0000110010001000" ⊕ "0000000000001110" = "0000110010002110".

Using the functional fitness formulation shown above, the following are obtained N_h = 13, N_m = 3 and F_f = 10.

Objective fitness: This is obtained by calculating the number of product terms needed in a given solution. The less the number of product terms the better the Objective Fitness is. There exists a number of ways to measure the Objective Fitness. In this paper, the following formulation is used: $F_o = (100 - N_p) / 100$ where N_p = Number of product terms needed.

The justification for the use of the above formulation is twofold. First, it is easy to interpret the value of the Objective Fitness to obtain the number of product terms. Second, there is a benefit in calculating the overall fitness, as explained below. Using the above formulation, the objective fitness of chromosome X given above is 0.97.

Overall fitness: Overall fitness of a chromosome is Fitness = $F_f + F_o$.

Thus a chromosome with the highest functional fitness will be considered best solution. However, if there is more than one chromosome that has the highest functional fitness, then the value of objective fitness will determine which of these to use as the best solution.

Crossover Operator: Due to its simplicity single point crossover is in this paper. Fig. 5 shows an example of the single point crossover operation as used in this paper. Parents, P1 and P2, are selected using a Roulette Wheel procedure so that better quality solutions have better opportunity to be picked.

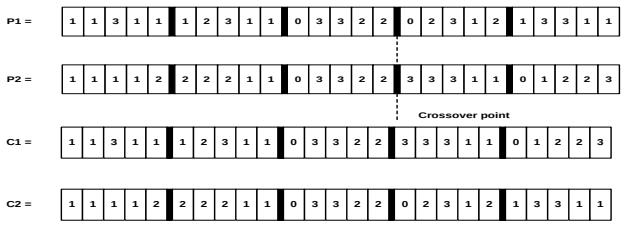


Fig. 5: Single point crossover

The next step is to generate a random crossover point. Since the chromosomes consist of five genes, there are only four crossover points allowed. According to Fig. 5, the first child, C1, is obtained by combining the left part (of crossover point) of P1's gene and the right part of P2's. Child C2 in Fig. 4 is obtained by combining the right part (of the crossover point) of parent P1 and the left part of parent P2.

Mutation Operator: The mutation operator can affect the quality of solution in two ways:

1. Adding or removing a product term: Mutation can add a product term from a solution by changing the value of the constant (of the gene) from 0 to 1 and remove a product term by changing the value of the constant from 1 to 0.
2. Changing a product term: A specific product term from a solution can be changed if the mutation operator changes the value of either position1 or position2 of X1 (or X2).

Selection Operator: Fig. 6 shows the process of selection operator as used in this paper. Assume that a population contains *n* individuals (solutions). Let's call an old population (the parents) as Pop1 while the population obtained from crossover and mutation as Pop2. The two populations, Pop1 and Pop2 are merged into a temporary population called Pop3. Then, Pop3 is sorted according to its fitness. The first quarter of Pop3 is copied into new population while the second, third and fourth quarters must join a tournament in order to be admitted to the remaining places in the new population. Using this approach, the best solution will be carried out to the new generation, while not reducing the possibility for not-so-good solutions to survive as well.

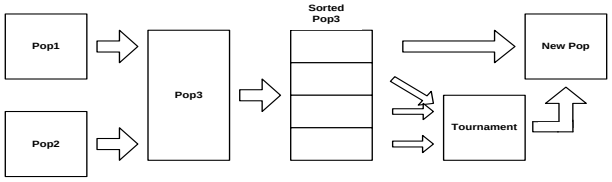


Fig. 6: Selection Operator.

On The length of Chromosome: The length of the chromosome is an important design parameter in any GA-based optimization problem. If the chromosome is too small, the GA will unlikely find the best solution. On the other hand, if the chromosome is too large, the GA may waste a lot of time finding solution in the no-solution part of the space. A straightforward approach is to use the length of truth table as the length of chromosome. But this does not guarantee that the GA will find the best solution in an effective way. In this paper, different lengths of chromosome are tested. These are explained below.

In this paper, the length of truth table (LTT) is defined as the number of all possible minterms of an MVL function. For 2-variable 4-valued function, the LTT is equal to 16. For 3-variable 4-valued function, LTT is equal to $4^3 = 64$. For 2-variable 4-valued function, the size of LTT =16 can be used as the length of chromosome since each minterm (rather than an implicant) can be represented as a gene. However, in most cases, the number of product terms will likely to be less than the number of minterms. It has been shown in [16] that the maximum number of implicants to cover any 2-variable 4-valued function is 12, which is 75% of LTT. Consider the 2-varaible 4-valued function shown in Fig. 7(a). The number of non-zero minterms of this function 7. This same function can be expressed as $F=1\bullet^0X_1{}^0\bullet^0X_2{}^3 \oplus 1\bullet^0X_1{}^3\bullet^3X_2{}^3$, which contains 2 implicants only. There is no need to use 12 or even 16 genes to represent this function. Even if the length of chromosome is 7, the search space for the GA to explore is still large. Now, consider the function shown in part (b) of Fig. 7. The value of NM is 8 and the minimum number of implicants to represent this function is 8. From these two examples, we conjecture that the length of the chromosome shouldn't be kept static. It should depend on the number of minterms in the given function.

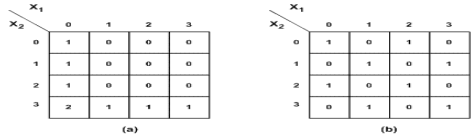


Fig. 7: Two examples of a 2-variable 4-valued function.

Based on the above discussion, we introduce the following rule in setting the length of the chromosome: the chromosome length is chosen as equal to ***NM*** if ***NM*** ≤ ***LTT*** / ***NV***, where ***NV***=Number of variable; or 0.75 • ***NM*** otherwise. Accordingly, four approaches can be adopted in setting the length of chromosome:

1. Static (SGA-MVL) [16]: In this approach, the length of chromosome is equal to ***LTT***.
2. Reduced Static (RSGA-MVL) [16]: In this approach, the length of chromosome is equal to 0.75 • ***LTT***.
3. Variable (VGA-MVL): In this approach, the length of chromosome is equal to ***NM***.
4. Reduced Variable (RVGA-MVL): In this approach, the length of chromosome is equal to ***NM*** if ***NM*** ≤ ***LTT*** / ***NV***; 0.75 • ***NM*** otherwise

Experimental Results: The experiments that have conducted and the results obtained using the GA meta-heuristic are presented below. Two main merits are measured. These are:

1. SR = Successful Rate (the number of successful iterations divided by the number of runs)
2. PT = Number of Product Terms

Unless it is mentioned explicitly, for all experiments, the following settings are used:

1. Number of Populations = 2000
2. Maximum number of Iterations = 500
3. Maximum number of run = 10
4. Crossover Rate = 100%
5. Mutation Rate = 5-10%

For the purpose of the experiment, 200 randomly generated 2-variable 4-valued functions are generated. The proposed approaches were able to find solutions for those functions. To illustrate the effectiveness of GA, the results of some of the functions is shown in Table 3. The table shows the value of the ***PT*** for the first 20 functions for RSGA-MVL, VGA-MVL and RVGA-MVL. On average, the best solution was obtained in less than 50 iterations. Only in few cases, GA needs more than 100 iterations to find a solution.

For the 200 randomly generated functions, the first three proposed approaches (SGA-MVL, RSGA-MVL and VGA-MVL) were able to find solutions. However, RVGA-MVL failed to find solutions for one out of 200 functions for the given experimental setting stated before. This is simply because of the inadequacy of the length of chromosome used to represent this function. To show this, consider the MVL function shown in Fig. 8. The value of NM for this function is 11. Using the rules for RVGA-MVL, the length of chromosome should equal to 8. However, there is no way to find solution having less than 10 implicants for this specific function.

Table 3: Values of the ***PT***.

function	RSGA-MVL	VGA-MVL	RVGA-MVL
0	8.14	8.43	8.00
1	7.00	6.43	6.60
2	8.22	8.40	8.00
3	7.20	7.00	7.00
4	6.50	7.17	6.83
5	5.40	5.00	5.30
6	5.00	5.00	5.00
7	7.00	7.00	7.00
8	7.20	7.13	7.60
9	8.00	8.25	8.10
10	7.10	7.44	7.00
11	6.00	6.00	6.00
12	8.11	9.50	8.29
13	6.20	6.00	6.00
14	6.00	6.10	6.10
15	8.00	8.14	8.00
16	6.11	6.25	6.38
17	8.25	8.50	7.86
18	8.10	7.86	7.86
19	6.40	6.10	6.00

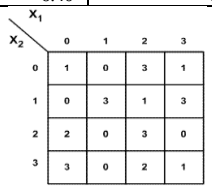


Fig. 8: Example RVGA-MVL fail to find solution.

Table 4 shows a comparison of the average value of SR and PT for all 200 2-variable 4-valued functions using the proposed approaches. Except for the observation made above, the table shows that RVGA-MVL outperforms the three other approaches. Table 5 shows the average number of implicants required to synthesize the 200 randomly generated 2-variable 4-valued functions for both the proposed approach and existing techniques. It can be seen that the proposed algorithms outperforms other techniques.

Table 4: SR and PT values for all proposed approaches

Algorithm	SGA-MVL	RSGA-MVL	VGA-MVL	RVGA-MVL
SR	0.45	0.58	0.58	0.62
PT	7.24	7.18	7.18	7.13

Table 5: Comparison with other techniques

Algorithm	Number of PTs
ARM [15]	7.955
BS [3]	8.055
DM [10]	7.355
SGA-MVL [16]	7.24
RSGA-MVL [16]	7.18
VGA-MVL	7.18
RVGA-MVL	7.13

4. ANT COLONY ALGORITHMS

The Ant Colony Optimization (ACO) algorithm is a new meta-heuristic that has a combination of distributed computation, autocatalysis (positive feedback) and constructive greediness to find an optimal solution for a number of combinatorial optimization problems. This algorithm tries to mimic the ant's behavior in the real world. Since its introduction, the ACO algorithm has received much attention and has been incorporated in many optimization problems, namely network routing, traveling salesman, quadratic assignment, and resource allocation problems [25][26].

The ACO algorithm (see Fig. 9) has been inspired by the experiments using a colony of real ants. It was observed that real ants were able to select the shortest path between their nest and food resource, in the existence of alternate paths between the two. The search is made possible by an indirect communication amongst the ants. While traveling their way, ants deposit a chemical substance, called *pheromone*, on the ground. When they arrive at a decision point, they make a probabilistic choice, biased by the intensity of pheromone they smell. This behavior has an autocatalytic effect because of the very fact that choosing a path will increase the probability that the same path will be chosen again by future ants. When

they return back, the probability of choosing the same path is higher (due to the increase in the amount of pheromone). New pheromone will be released on the chosen path, which makes it more attractive for future ants. Shortly, all ants will select the shortest path.

```
Algorithm ACO meta heuristic();
while (termination criterion is not satisfied)
        ant generation and activity();
        pheromone evaporation();
        daemon actions(); //optional
end while
end Algorithm
```

$2 \bullet {}^{0}x_1^{1} \bullet {}^{0}x_2^{1}$

Fig. 9: Ant Colony Optimization Algorithm

4.1. Solution Representation

The basic idea of using ACO in MVL synthesis is to use the ants to find the best representation by choosing the right minterm and implicant. Similar to the DC (direct cover) algorithms, first an ant will select the best minterm and then select the best implicant for that minterm. Then, it will reduce the MVL function table by removing the selected implicant. This will be performed iteratively, until all minterms in the table is covered. However, unlike the DC algorithms, every time an ant selects a minterm (or an implicant) it will put some pheromone trails on that minterm (or implicant). By doing this, the next ant will perform selection based on the additional pheromone information that will hopefully drive it to the best solution.

In our proposed ACO for MVL (ACO-MVL), a given MVL function is represented in a tabular format. Each ant will carry a 'bag' in which it stores all selected implicants. The size of the bag itself is equal to the length of the truth table of the function. Each implicant is represented by a string of integer consisting five integer attributes. The first attribute represents the value of the constant of that corresponding implicant. The 2nd and 3rd attributes are for the first and second position of the first variable of the corresponding product term. The first and second positions of the second variable are represented by the fourth and fifth attributes, respectively. This is similar to the representation used in the case of the GA (see Section 3).

Selection of a minterm (or an implicant) is performed using a stochastic process of Roulette Wheel. The probability of choosing a minterm (or implicant) depends on the pheromone value and a heuristic value for that minterm (or implicant). The pheromone value is obtained from pheromone trails of previous ants while the heuristic value is the fuzzy weighted averaging of minterm's (or implicant's) selection criteria. The selection process is explained below.

4.2. Minterm Selection

In selecting a minterm, the ant needs the minterm's *pheromone value* and *heuristic value*. Thus, with τ_M representing the pheromone value of minterm M and η_M representing the heuristic value of minterm M, the probability of choosing minterm M is computed as: $p = \frac{\tau_M \bullet \eta_M}{\sum \tau_M \bullet \eta_M}$

4.3. Implicant Selection

The procedure for implicant selection is similar to the minterm selection above. The only difference is that it operates on implicants. Thus, with τ_L representing the pheromone value of implicant L and η_L representing the heuristic value of literal L (fuzzy weighted averaging of literal's L selection criteria), the probability of choosing implicant L is computed as: $p = \frac{\tau_L \bullet \eta_L}{\sum \tau_L \bullet \eta_L}$

4.4. Fitness Function Calculation

Similar to the GA case, the fitness function is divided into two parts. The first part is called *Functional Fitness*, $\boldsymbol{F_f}$. The second part of the fitness function is called *Objective Fitness*, $\boldsymbol{F_o}$. Functional fitness is obtained by comparing the truth table of the given function with that of the obtained one. It is equal to the number of minterm matching (hits), $\boldsymbol{N_h}$, between the two truth tables less the number of mismatch, $\boldsymbol{N_m}$, between the two truth tables. This can be formulated as $\boldsymbol{F_f = N_h - N_m}$. With $\boldsymbol{N_p}$ is the number of product terms, the Objective Fitness is formulated as $\boldsymbol{F_o = (100 - N_p)/100}$. A chromosome with the highest functional fitness

will be considered best solution. If there is more than one chromosome that has the highest functional fitness, then the value of objective fitness will determine which of these to use as the best solution.

Figure 10 shows the pseudo code of ACO-MVL. The algorithm used is basically similar approach to the DC algorithm. The daemon_action() function is a procedure that will be performed periodically or when it is needed, e.g., when it is required to reset the pheromone value on all minterms (or implicants). This is performed at the beginning of all ant's movement and if we found a stagnancy in the solution obtained.

Consider the example shown in Fig. 11. The function has three minterms, namely: $1\bullet^{1}\mathbf{X_1}^{1}\bullet^{1}\mathbf{X_2}^{1}$, $2\bullet^{2}\mathbf{X_1}^{2}\bullet^{1}\mathbf{X_2}^{1}$ and $3\bullet^{3}\mathbf{X_1}^{3}\bullet^{1}\mathbf{X_2}^{1}$. Assume these three minterms are the options for ant to select. Assume also that the ant select minterm $1\bullet^{1}\mathbf{X_1}^{1}\bullet^{1}\mathbf{X_2}^{1}$. Then the ant should select an implicant covering this minterm. Fig. 12 (a) shows that there are three possible implicants covering it. The ant will then need to select any of these implicant based on the pheromone trails on those implicants. Suppose that the ant select implicant $1\bullet^{1}\mathbf{X_1}^{3}\bullet^{1}\mathbf{X_2}^{1}$. This implicant will then be stored in ant's bag. Once an implicant is

selected, the function's table is updated. This is done by reducing the table with the selected implicant. After that, the ant will check whether all minterms are covered or not. If not, the process of selecting minterm and implicant will be repeated. This is shown in Fig. 11 (b) and (c). When all minterms are covered, the ant will stop the selection process and return the function. The Example in Fig. 11 shows that the ant results a functions having the following implicants: $1\bullet^{1}\mathbf{X_1}^{3}\bullet^{1}\mathbf{X_2}^{1}$, $1\bullet^{2}\mathbf{X_1}^{3}\bullet^{1}\mathbf{X_2}^{1}$, and $1\bullet^{3}\mathbf{X_1}^{3}\bullet^{1}\mathbf{X_2}^{1}$.

The next step is to calculate the fitness of the above result. Based on this result, the pheromone of the selected minterms and implicants will be updated. The next generation of ants will then used the new updated pheromone values to guide their selection. At the end, the best results will be returned by ACO-MVL algorithm.

```
For r number of runs do
    For a number of ants
            daemon_action();
                ant[a].L = {};
                while (checkTable()){
                        M = selectMintermACO();
                        L = selectImplicantACO(M);
                        ant[a].L ← ant[a].L + L;
                }
            calculate_fitness(ant[a]);
    done
    pheromone_update_minterm();
    pheromone_update_implicant();
done
```

Fig. 10: Pseudo code of ACO-MVL

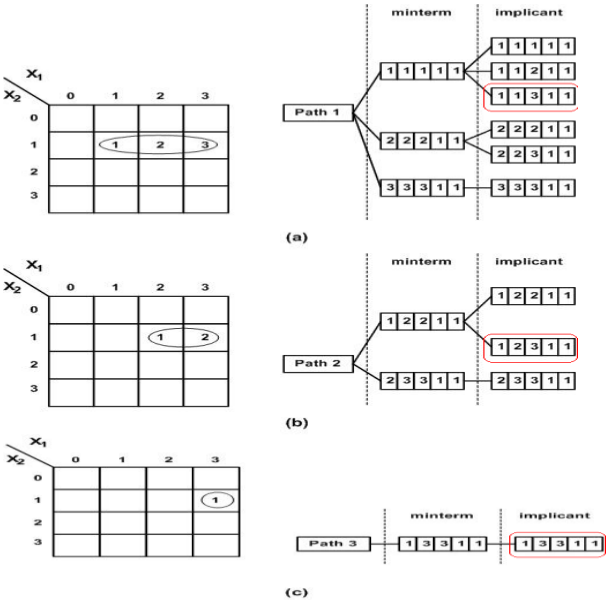


Fig. 11: Example of ACOSC-MVL

4.5. Experimental Results for ACOSC-MVL

The following are the parameters used in our ACOSC-MVL.

1. Number of Populations = 20
2. Maximum number of Iterations = 100
3. Maximum number of runs = 10
4. Pheromone evaporation rate = 0.95

Table 6 shows the average PT (Product Term) with respect to different number of minterms of MVL functions used as benchmark.

It can be seen clearly that the proposed algorithm outperforms other techniques, regardless the number of minterms the MVL function has.

The overall average PT is shown in Table 7. This table shows that the proposed algorithm outperforms other techniques in terms of average PT.

Table 6: Average PT with Respect to Different Number of Minterms of MVL Functions Used

# min.	# func.	ARM	BS	DM	ODC
16	500	7.594	7.562	7.002	7.086
15	2679	8.295	8.307	7.51	7.481
14	6589	8.355	8.405	7.569	7.516
13	10585	8.275	8.352	7.541	7.491
12	11230	8.049	8.098	7.382	7.33
11	9003	7.707	7.757	7.129	7.086
10	5434	7.323	7.366	6.831	6.787
9	2575	6.871	6.879	6.473	6.436
8	1038	6.309	6.322	6.023	5.978
7	277	5.726	5.751	5.527	5.484
6	75	5.133	5.147	4.973	4.96
5	13	4	4	4	3.923
4	1	4	4	4	4
3	1	3	3	3	3

# min.	# func.	WDC	ACO-MVL	ACOSC-MVL
16	500	7.01	6.784	6.73
15	2679	7.501	7.102	7.054
14	6589	7.559	7.197	7.163
13	10585	7.545	7.204	7.182
12	11230	7.383	7.099	7.086
11	9003	7.134	6.912	6.904
10	5434	6.837	6.695	6.66
9	2575	6.479	6.403	6.356
8	1038	6.022	6.021	5.934
7	277	5.523	5.556	5.48
6	75	4.987	5	4.96
5	13	4	4.154	3.923
4	1	4	4	4
3	1	3	3	3

Table 7: Comparison with Other Techniques

Algorithm	Number of PTs
ARM [15]	7.8901
BS [3]	7.9388
DM [10]	7.2478
ODC [65]	7.2023
FWDC [65]	7.2491
VGA-MVL	7.18
RVGA-MVL	7.13
ACO-MVL [26]	6.9827
ACOSC-MVL [26]	6.9575

5. CONCLUDING REMARKS

In this paper, we have presented two evolutionary techniques for synthesis of MVL functions: the Gas and ACO. We have also compared the results obtained using these algorithms through simulating the three algorithms using two benchmarks: 50000 2-variable 4-valued and 2-variable 5-valued functions. The results obtained showed that the two evolutionary techniques outperform the conventional DCs. Among the two evolutionary techniques the ACOSC-MVL achieved the best results.

ACKNOWLEDGEMENT

The financial support received from Kuwait University is greatly appreciated.

REFERENCES

[1] A. Mishchenko, B. Steinbach, M. Perkowski, ”Bi-decomposition of multi-valued relations”. Proc. International Workshop on Logic and Synthesis 2001, pp. 35-40.

[2] A. Mishchenko and R. Brayton, ”Simplification of non-deterministic multi-valued networks”, Proc. International Conference on Computer Aided Design 2002, pp.557-562.

[3] B. Zupan. Machine Learning Based on Functional Decomposition. PhD thesis, University of Ljubljana, Slovenia, 1997.

[4] C. Files, R. Drechsler, and M. Perkowski. Functional decomposition of MVL functions using multi-valued decision diagram. International Symposium on Multi-Valued Logic (ISMVL), May 1997, pp. 27–32.

[5] C. M. Files and M.A Perkowski, “New multi-valued functional decomposition algorithms based on MDDs”. IEEE Trans. CAD. Vol. 19(9), Sept. 2000, pp. 1081-1086.

[6] C. M. Files and M. A Perkowski, “Multi-valued functional decomposition as a machine learning method”, Proc. ISMVL '98, pp. 173 -178.

[7] C. Yang and Y.-M. Wang. A neighborhood decoupling algorithm for truncated sum minimization. Proceedings of the Twentieth ISMVL, 1990, May 1990, pp. 153-160.

[8] C. Yildirim and J. T. Butler and C. Yang, "Multiple-valued PLA minimization by concurrent multiple and mixed simulated annealing," Proceedings of the 23rd ISMVL, May 1993, pp. 17-23.

[9] Dueck, G. W. and Miller, D. M., "A Direct Cover MVL Minimization Using the Truncated Sum." Proceeding of the 17th ISMVL, May 1987, pp. 221-227.

[10] D. D. Olmsted, “The History and Principles of Multivalued Logic”, 1998.

[11] D. M. Miller. Decomposition in many valued design. PhD thesis, University of Manitoba, May 1976.

[12] E. Dubrova, Y. Jiang, R. Brayton, “Minimization of multi-valued functions in Post algebra”, Proc. International Workshop on Logic and Synthesis 2001, pp. 132-137.

[13] E. L. Post, “Introduction to a general theory of elementary propositions”, American Journal of Mathematics, Vol. 43, 1920, pp. 163-185.

[14] G. Promper and J. A. Armstrong. Representation of multiple valued functions using the direct cover method. IEEE transactions on Computers, Sept. 1981, pp. 674-679.

[15] G. W. Dueck and G. H. J. van Rees, “On the Maximum Number of Implicants Needed to Cover a Multiple-Valued Logic Function Using Window Literals”, ISMVL 1991, pp. 280-286.

[16] G. Epstein, G. Frieder and D. C. Rine, “The development of Multiple-valued logic as related to computer science”, Computer, Vol. 7 (9), 1974, pp. 20-32.

[17] J. Holland. Adaptation in Natural and Artificial Systems. MIT Press, MA. 1997

[18] J.-H. J. M. Goa and R. Brayton. Optimization of multi-valued multi-level networks. Proceedings of the International Symposium on Multiple-Valued Logic, ISMVL 2002, May 2002, pp. 12-16.

[19] J. Lukasiewicz, english translation: On three valued-logic, in L. Borkowski (ed), Select Works, North-Holland, Amsterdam, Vol. 5, 1920, pp. 169-171.

[20] L.A. Zadeh, “Fuzzy Sets, Information and Control, 8(3): June 1965, 338-353.

[21] Miller, D.M., “Spectral transformation of multiplevalued decision diagrams”, Proc. 24th Int. Symp. On Multiple-Valued Logic, May 1994, pp. 89-96.

[22] Miller, D.M., Drechsler, R., “Implementing a multiplevalued decision diagram package”, Proc. 28th Int. Symp. on Multiple-Valued Logic, May 1998, 52-57.

[23] Mostafa Abd-El-Barr and Louai Al-Awami, “Analysis of Direct Cover Algorithms for Minimization of MVL Functions”, Dec. 2003, pp. 308- 312.

[24] M. Dorigo and G. Di Caro, “New Ideas in Optimization”, McGraw Hill, London, UK, 1999.

[25] M. Dorigo and T. Stutzle, “The Ant Colony Optimization Metaheuristic: Algorithms, Applications and Advances, 2002.

[26] M. Kameyama, S. Kawahito, T. Higuchi, “ A Multiplier chip with Multiple-Valued Bidirectional Current-Mode Logic Circuits”, IEEE Computer, April 1988, pp. 43-56.

[27] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, J. S. Zhang, “Decomposition of multiple-valued relations”. Proc. ISMVL ’97, pp. 13-18.

[28] P. T. Cheung and D. M. Purvis. A computer-oriented heuristic minimization algorithm for multiple-output multivaued switching functions. Proceeding of International Symposium on Multiple-Valued Logic (ISMVL), 1974, pp. 112-120.

[29] Rolf Drechsler Dragan Jankovi´c Radomir S. Stankovi´c, Generic Implementation of DD Packages in MVL, EUROMICRO Conference, 1999, vol. 1, pp. 352-359.

[30] Rolf Drechsler Mitch Thornton David Wessels, MDD-Based Synthesis of Multi-Valued Logic Networks, Proceedings. 30th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2000, pp. 41-46.

[31] R. G. C. Hong, S. J. and D. L. Ostapko. Mini: A heuristic approach for logic minimization. IBM J. Res. Develop., 18:443-458, 1974.

[32] R. K. Brayton and S. P. Khatri. Multi-valued logic synthesis. International Conference on VLSI Design, Goa, India, Jan 1999.

[33] R. L. Rudell and A. Sangiovanni-Vincentelli, “Multiple-valued minimization for PLA optimization”. IEEE Trans. CAD, vol. 6(5), Sept. 1987, pp. 727-750.

[34] Stankovi´c, R.S., “Functional decision diagrams for multiple-valued functions”, Proc. 25th Int. Symp. On Multiple-Valued Logic, 1995, pp. 284-289.

[35] Takahiro Hozumi, Osamu Kakusho, Kazuharu Yamato: An Evolutionary Computing Approach to Multilevel Logic Synthesis Using Various Logic Operations. ISMVL 2000: pp. 259-264

[36] Tirumalai, P. and Butler, J. “Analysis of Minimization Algorithms for Multiple-Valued Programmable Logic Arrays.” ISMVL 1988, pp. 226-236.

[37] Tirumalai, P. Multiple-valued programmable logic arrays. Ph.D. Thesis, Northwestern University, Evanston, USA, 1989.

[38] T. Kalganova and J. Miller, "Evolutionary Approach to Design Multiple-Valued Combinational Circuits," Proc. 4th-Int. Conf. on Applications of Computer Systems, 1997, pp.333-339.

[39] T. Kalganova, J. Miller and N. Lipnitskaya, "Multiple-Valued Combinational Circuits Synthesized Using Evolvable Hardware Approach," Proc. 7th-Workshop on Post-Binary Ultra Large Scale Integration Systems in Association with ISMVL'98, pp.52-54.

[40] T. Luba. Decomposition of multiple-valued functions. International Symposium on Multi-Valued Logic, pages 256–261, 1995.

[41] T. Sasao, "On the Optimal design of Multiple-Valued PLA's," IEEE Trans. Comput., C-38, 4, 1989, pp.582-592.

[42] T. Sasao and J.T. Butler. Planar multiple-valued decision diagrams. In Int'l Symp. on multi-valued Logic, pp. 28–35, 1995.

[43] W. Wang and C. Moraga, "Design of Multivalued Circuit using Genetic Algorithms," IEEE Proc. 26th-Int. Symp. on Multiple-Valued Logic, pp.216-221 (1996).

[44] W. Wang and C. Moraga, "Evolutionary Methods in the Design of Quaternary Digital Circuits," IEEE Proc. 28th-Int. Symp. on Multiple-Valued Logic, pp.89-94 (1998).

[45] Y. Hata, K. Hayase, T. Hozumi, N. Kamiura and K. Yamato, "Multiple-Valued Logic Minimization by Genetic Algorithms," IEEE Proc. 27th-Int. Symp. On Multiple-Valued Logic, pp.97-102 (1998).

[46] K. L. C. Naiff, D. A. Rich and K. G. Smalley, "A Four-State ROM Using Multilevel Process Technology", IEEE Journal of Solid-State Circuits, Vol. 19, No. 2, pp. 174–179, April 1984.

[47] H. M. Razavi, S. E. Bou-Ghazale, "Design of a Fast CMOS Ternary Adder", IEEE International Symposium on Multiple-Valued Logic (ISMVL), pages 20–23, May 1987.

[48] Bambang A.B. Sarif and Mostafa Abd-El-Barr. Synthesis of MVL Functions – Part I: The Genetic Algorithm Approach. International Conference on Microelectronics (ICM-06) held December 16-19, 2006, Dhahran Saudi Arabia, pp. 158-161.

[49] Mostafa Abd-El-Barr and Bambang A.B. Sarif. Synthesis of MVL Functions – Part II: The Ant Colony Optimization Approach. Accepted. International Conference on Microelectronics (ICM-06) held December 16-19, 2006, Dhahran Saudi Arabia, pp. 154-157.

[50] Abd-El-Barr, M., "Ant Colony Heuristic Algorithm for Multi-Level Synthesis of Multiple-Valued Logic Functions", International Journal of Computed Science (IJCS), 37:1, IJCS_37_1_07, February 2011, pp. 1-6.

[51] Perkowski, M., Foote, D., Chen, Q., and Al-Rabadi, A., "Learning Hardware using Multiple-Valued Logic", IEEE Micro, May-June, 2002, pp. 41-51.

[52] Dubrova, E., "Multiple-Valued Logic in VLSI", Multiple-Valued Logic: An International Journal, 2002, pp. 1-17.

[53] Manikas, T., and Teeters, D., "Multiple-Valued Logic Memory System Design Using Nanoscale Elctrochemical Cells", Proc. 38th ISMVL, May 2008, pp. 197-201.

[54] Hanyu, T. and Kameyama, M., "A 200 MHz Pipelined Multiplier Using 1.5 V-Supply Multiple-valued MOS Current-Mode Circuits with Dual-Rail Source-Coupled Logic", IEEE Journal of Solid-State Circuits, vol. 30, no. 11, November 1995, pp. 1239-1245.

[55] Patel, V. and Gurmurthy, K., "Arithmetic Operations in Multi-Valued Logic", International Journal of VLSI & Communication Systems (VLSICS), vol. 1, no. 1, March 2010, pp. 21-32.

[56] Hosseinzadeh, M., Jassbi, S., and Navi, K., "A Novel Multiple Valued Logic OHRNS Modulo *rn* Adder Circuit", Proceeding of World Academy of Science, Engineering and Technology, vol. 25, November 2007, ISSN 1307-6884, pp. 128-132.

[57] Kouretus, I. and Puliourus, V., "High-Radix Redundant Circuits for Modulo r_n-1, r_n or r_n+1," Proceedings of the 2003 International Symposium on Circuits and Systems, Vol. 5, 2003, pp. 25-32.

[58] Yildirim, C., Butler, J., and Yang, C., "Multiple-Valued PLA Minimization by Concurrent Multiple and Mixed Simulated Annealing", Proceedings 1993 ISMVL, pp. 17-23.

[59] Gao, S., Cao, Q., Zhang, Z., and Tang, Z., "A Chaotic Clonal Selection Algorithm and its Application to Synthesize Multiple-Valued Logic Functions", Transactions on Electrical and Electronic Engineering, vol. 5, no. 1, January 2010, pp. 105-114.

[60] Mishchenko, A. and Brayton, R., "Simplification of non-deterministic multi-valued networks", Proc. International Conference on Computer Aided Design 2002, pp.557-562.

[61] Sarif, B. and Abd-El-Barr, M., "Fuzzy-based Direct Cover Algorithm for Synthesis of Multiple-Valued Logic Functions", IASTED Circuits and Systems, held Hawaii, September 2008, pp. 625-630.

[62] Lee, K. and El-Sharkawi, M., "Modern Heuristic Optimization Techniques", Institute of Electrical and Electronics Engineers (IEEE), January 2008, ISBN 9780470225868.

[63] Patel, V. and Gurumurthy, K., "Arithmetic Operations in Multi-Valued Logic", International Journal of VLSI Design & Communication Systems, vol. 1, no. 1, March 2010, pp. 21-32.

[64] Abd-El-Barr, M and Sarif, B., "Weighted and Ordered Direct Cover Algorithms for Minimization of MVL Functions", Proceedings, the 37st ISMVL-2007, held May 13-16, 2007, Oslo, Norway, pp. 48-53.

[65] Yager, R., "On Ordered Weighted Averaging Aggregation Operators in Multi-criteria Decision Making", IEEE Transactions on Systems, Man, and Cybernetics 18(1), 1988, pp. 183-19.