Ant Colony Heuristic Algorithm For Multi-Level Synthesis of Multiple-Valued Logic Functions

Mostafa Abd-El-Barr, *Senior Member IEEE, Member, IAENG*

***Abstract*— A number of successful implementation of Multiple-Valued Logic (MVL) circuits using Very Large Scale Integration (VLSI) technology has been reported in the literature. The Ant Colony (ACO) optimization algorithm is a meta-heuristic that mimics the ants' behavior in finding the shortest path to reach food sources. We have previously introduced ACO-based heuristic for synthesis of two-level MVL functions. In this paper, we introduce a hybrid ACO-Direct Cover (DC) technique for synthesis of multi-level MVL functions. In this technique, we use ants to decompose the given MVL function into a number of levels and synthesize each sub-function using a DC-based technique. A benchmark set consisting of 50000 randomly generated 2-varaible 4-valued functions is used to compare the results obtained using the proposed approach with those obtained using existing techniques. It is shown that on average the proposed hybrid technique produces more efficient realizations in terms of the chip area consumed in synthesizing a given MVL function.**

***Index Terms*—Multiple-Valued Logic, Multi-level Synthesis, Direct Cover Algorithms, Ant Colony Optimization, Heuristic Optimization Techniques.**

I. INTRODUCTION

A number of successful implementations of multi-valued logic (MVL) circuits using Complementary Metal Oxide Semiconductor (CMOS) technology have been reported in the literature. These include memory applications [1]-[4], high speed arithmetic circuits [5]-[6], image processing [7], Radix-4 encryption [8], and robotics/machine learning [9]-[10]. The use of MVL in the above mentioned applications has led to the advantages of requiring less chip area, less pin-count, and faster speed as compared to those achieved using binary logic. Recent review on the use of MVL in Very Large Scale Integration (VLSI) technology can be found in [11]-[13].

There are $r^{(r^n)} = 4^{4^2} = 2^{32}$ 2-variable 4-valued functions as compared to 2^4 2–varaible 2–valued (binary) functions. Exploring the search space in finding optimal synthesis of such large number of 4-valued functions is prohibitively expensive and that resorting to heuristic algorithms (HAs) is a necessity [14]. The use of HAs in producing near-minimal sum-of products (PLA) realization of MVL functions can be categorized as functional decomposition [15]-[16], iterative functional improvement [17], direct cover [18]-[21], and evolutionary optimization [22]-[27]. The author has introduced a hybrid Ant-Colony (ACO) Direct Cover algorithm for two-level synthesis of MVL functions [28]. The algorithm used the ACO in finding the shortest path to the (near) optimal number of product terms that cover a given MVL function.

Multi-level synthesis of binary and MVL functions has been proposed in the literature [29]-[33]. None of the reported techniques investigated the possibility of integrating the DC and the ACO in multi-level synthesis of MVL functions. In this paper a hybrid ACO-DC algorithm for multi-level synthesis of MVL functions is introduced. The proposed technique works by decomposing a given MVL function using ACO and synthesizing the sub-functions using the best known DC-based algorithm. A benchmark set consisting of 50000 randomly generated 2-varaibale 4-valued function is used to test the results obtained using the proposed hybrid ACO-DC multi-level synthesis algorithm. The results obtained using the ACO-DC algorithm are compared to those obtained using existing techniques in terms of the average number of MVL gates needed to synthesize a given MVL function.

The paper is organized as follows. In Section 2, we present some background material. In Section 3, we present the use of the Ant Colony (ACO) optimization technique in two-level synthesis of MVL functions. The proposed ACO-DC technique for synthesis of multi-level MVL functions is introduced in Section 4. In Section 5 we present the experimental results obtained and comparison with other techniques. Section 6 concludes the paper.

II. BACKGROUND MATERIAL

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

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

Manuscript received January 10, 2010. This work was supported in part by Kuwait University under Grants WI 05/04 and WI 02/07.

Mostafa Abd-El-Barr is with the Information Science Department, CFW, Kuwait University, Adylia Campus, P. O. Box 5969, Safat 13060, Kuwait. Phone 965-2-498-3301; Fax: 965-2-532-9417; email: mostafa.abedelbarr@gmail.com

Definition 2: *A window literal* ${}^{a}x^{b}$ on a MVL variable x is defined as follows:

$$x^{a,b} = \begin{cases} r-1 & if \quad a \le x \le b \\ 0 & otherwise \end{cases}$$

Where $a,b \in R$ and $a \le b$. □

Definition 3: A *tsum* (truncated sum) operator is defined as

$$tsum(a_1,a_2,...,a_n) = a_1 \oplus a_2 \oplus ... \oplus a_n = \begin{cases} a_1 + a_2 + ... + a_n & if \quad a_1 + a_2 + ... + a_n < r-1 \\ r-1 & otherwise \end{cases}$$

Where $a_i \in R$ and $\oplus$ represents the truncated sum operation.

Definition 4: A *product term* (PT), $P(x_1, x_2, ..., x_n)$, is defined as the minimum of a set of *window literals* such that $P(x_1,x_2,...,x_n) = c \bullet x_1^{a_1,b_1} , x_2^{a_2,b_2} ,..., x_n^{a_n,b_n} = \min(c, x_1^{a_1,b_1} , x_2^{a_2,b_2} ,..., x_n^{a_n,b_n})$ where $a_i, b_i \in R$, $a_i \le b_i$ and $c \in \{1,2,\ldots, r\text{-}1\}$ is called the value of the PT. □

Definition 5: 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 *product term*, *PT*, which is dependent on all variables and for which $a_1 = b_1, a_2 = b_2 = ..., a_n = b_n$. Consider, for example, the 2-variable 4-valued function shown in Fig. 1. In this function $1 \bullet x_1^{3,3} x_2^{0,0}$, $2 \bullet x_1^{1,1} x_2^{1,1}$, and $3 \bullet x_1^{3,3} x_2^{2,2}$ are examples of minterms in the function.

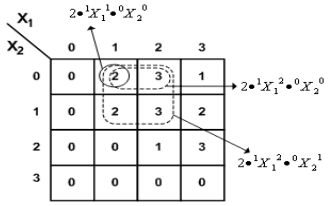

Fig. 1: Tabular representation of a 2-variable 4-valued function.

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

(1) choose a minterm,
(2) identify a suitable implicant 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 DC approaches reported in the literature differ in the way *minterms* are chosen and the way according to which *implicants* are identified. In [20] *minterms* are selected randomly and implicants are selected such that they result in the largest number of zero *minterms* (LRZ). In [18] a metric called the *isolation weight* (IW) is used in selecting *minterms* while an efficiency factor is used in selecting the implicant most suitable for covering the selected minterm. The IW is measure of the degree to which other minterms cluster around the targeted minterm. The efficiency factor is defined as the largest factor resulting from dividing the cost of each implicant that covers the targeted minterm by the number of minterms it covers. In [19] a metric called the *isolation factor* (IF) is used in selecting *minterms* while implicants are selected based on a metric called the *Relative Break Count* (RBC). The IF provides a measure of the degree to which a specific minterm can combine with other minterms in the function. The RBC provides a degree to which the function is simplified if the implicant under consideration is selected. The last two techniques choose *minterms* in increasing order of values, i.e., they start with lower minterm values and proceed to higher minterm values. In [21], the most isolated minterm is selected first and from all *implicants* which cover that minterm, the one that is not strongly "coupled" with its neighbours is selected. A measure of coupling the strength of an implicant with its neighbours, called *Neighbourhood Relative Count* (NRC), is used in selecting implicants such that the *implicant* with lowest NRC is selected. It is observed that there is no general agreement on which of the above criterion is the best for synthesizing a given MVL function. An attempt has been made in the work reported in [34] to analyze combination of restricted subset of criteria and comparing the results obtained in terms of the number of *implicants* needed to cover a given function.

The Ant Colony Optimization (ACO) algorithm is based on experimental work which concludes that ants select the shortest path between their nest and food resource, in the existence of alternate paths between the two. It is hypothesized that while traveling their way, ants deposit a substance called *pheromone*, the intensity of which is used by individual ants to make probabilistic choice at decision points (see Figure 2(a)). The probability that a given path will be selected again by future ants is increased 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 (see Figure 2(b)). Shortly, all ants will select the shortest path as shown in Figure 2(c) [35]-[36], and [38].

The behaviour of ants in the ACO algorithm can be summarized as follows. The problem is represented as a graph *G*. A colony of ants concurrently and asynchronously moves through each neighbor nodes of *G*. At each node, ants select the best partial solution by applying a stochastic local decision policy which makes use of the information contained in the local node and an ant's routing table. As they move, ants incrementally build optimized solutions to the problem. When the solution is being built, it is evaluated by every ant and the information about that solution goodness is put on the pheromone trails of the path used. This pheromone information will direct the search of future ants, until a feasible solution is found. A general outline of the ACO algorithm is presented in Figure 3 [35]-[36], [38], and [40].

III. Two-Level Synthesis of MVL Functions Using ACO

We have introduced in [28] an algorithm that uses ACO for two-level synthesis of MVL functions. According to this

algorithm, each implicant in the given function is represented by a string consisting of five integers attribute, see Figure 4.

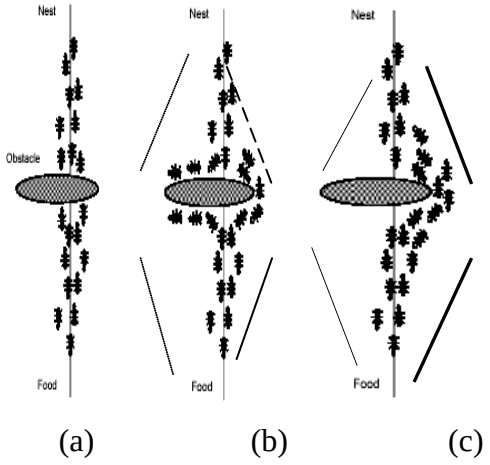

Fig. 2: Selection of shortest path to food sources by Ants.

```
procedure AC_ MetaHeuristic();
  while (not_termination)
    generateSolutions ();
    pheromoneUpdate ();
    daemon actions (); //optional
   end while
end procedure
```

Figure 3: Ant Colony pseudo-code algorithm.

C_{pn}	$X1_{pn_1}$	$X1_{pn_2}$	$X2_{pn_1}$	$X2_{pn_2}$

Figure 4: Implicant representation in ACO.

The first attribute in Fig. 4, C_{pn}, represents the value of the constant of n-th product term. The 2nd and 3rd attributes, $x_{1_{pn1}}$ and $x_{1_{pn2}}$ represent the boundary of the literal on the first variable x_1 of the corresponding product term (see Definition 2). The 4th and the 5th attributes, $x_{2_{pn1}}$ and $x_{2_{pn2}}$ represent the boundary of the literal on the second variable x_2 of the corresponding product term.

The idea is to use the ants to find the best coverage (analogous to choosing the shortest path) by choosing the right minterms and the appropriate implicants. Every time an ant selects a minterm (or an implicant) it will put some pheromone trails on that minterm (or implicant). This action will make the next ant to perform its selection based on the additional pheromone information. The probability of choosing a minterm (or implicant) depends on the pheromone values and a heuristic value of that minterm (or implicant). 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. Two different approaches: Ant Colony Optimization for MVL synthesis (ACO-MVL) and Ant Colony Optimization – using Selection Criteria – for MVL synthesis (ACOSC-MVL) were introduced. Figure 5 shows the pseudo code of ACOSC-MVL [24]. 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 ants' movement and if a stagnancy in the solution is found.

```
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
```

Figure 5: Pseudo code of ACOSC-MVL.

IV. The Proposed Multi-level Synthesis of MVL Functions using ACO-DC

In this technique ants are used to decompose a given function into a number of levels. Each sub-function is then synthesized using the best known DC-based technique found in the literature. Working from the circuit's output, the proposed algorithm proceeds as follows:

(1) place a certain gate type at this level, e.g. *tsum* gate (see Definition 3),
(2) decompose the given function using ACO, and
(3) Synthesize the (sub)-functions using the best DC-based technique found in the literature.

Steps 2 and 3 can be performed repeatedly to create a multi-level structure. However, only 3-level synthesis is performed in this paper. In addition, we limit the application of the proposed algorithm to the case of 2- input *tsum* gate (see Definition 3) at the output of the circuit's last level. We opted to use the DC-based algorithm proposed in [25] since it represents the best baseline DC technique available in the literature. For 4-valued functions, Table 1 summarizes the different possible decompositions of values 0, 1, 2, and 3 using the *tsum* operation (see Definition 3).

TABLE 1: 4-valued decomposition table

Logic Value	Possible decomposition
0	(0,0)
1	(0,1), (1,0)
2	(0,2), (1,1), (2,0)
3	(0,3), (1,2), (1,3), (2,1), (2,2), (2,3), (3,0), (3,1), (3,2), (3,3)

In the proposed algorithm an ant will travel through the truth table of the given function and for each entry in the table it

will select one combination out of the possible decompositions shown in Table 1. The probability of selecting a possible decomposition is calculated as $p = \tau_d / \Sigma \tau_d$, where τ_d is the pheromone value of d^{th} possible decomposition. After an ant finishes selecting a possible decomposition for all positions in the table, the best ant will update the pheromone on the selected combination of decomposition. The amount of pheromone dropped is proportional to its fitness and calculated as follows: $\Delta\tau = PW \bullet F_f$, where PW is the pheromone weight and $F_f = (100\text{-}N_g)/100$ is the functional fitness, where N_g is the number of gates used. Using such fitness function calculation; the representation that has least number of gates will have the highest F_f. In addition to F_f, we use an additional criterion in selecting the best representation called *logic balance*, *B*, which is calculated as the difference in the number of gates between the two sub-functions generated by the ant's decomposition process. The lower the difference is, the better the selection. We believe that having a balanced circuit is desirable. The proposed algorithm will try to find the circuits representation that uses the least number of gates. Out of these representations, the one that has the best logic balance will be selected.

Example: Consider the 2-variable 4-valued function shown in Fig. 1. Table 2 shows an enumeration of the possible paths for ants to travel through the function. This enumeration is done by passing through the function row-wise from left to right (from 0 to 15). Let us assume that an ant selects the following path: ((0,0), (2,0), (2,1), (0,1), (0,0), (2,0), (2,1), (1,1), (0,0), (0,0), (0,1), (2,1), (0,0), (0,0), (0,0), (0,0)). This leads to the decomposition shown in Fig. 6.

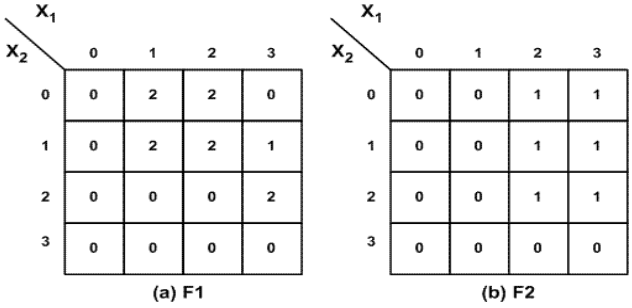

Fig. 6: Ant decomposition of the function of in Fig. 1.

From this figure, it is easy to see that F1 can be synthesized using 3 literal gates, $2 \bullet x_1^{1,2}\, x_2^{0,1}, 1 \bullet x_1^{3,3}\, x_2^{1,2}$, and $1 \bullet x_1^{3,3}\, x_2^{2,2}$ while F2 requires only one literal gate, $1 \bullet x_1^{2,3}\, x_2^{0,2}$. This makes the total number of gates needed to realize the function in Fig. 1 to be 5 gates, including the *tsum* combining both F1 and F2. The F_f value of this representation is equal to (100-5)/100 = 0.95 while the balance is equal to 2. Suppose that any other ant managed to get a representation with higher F_f, then the representation of the later will be used. The algorithm will iterate until a certain stopping criteria such as the number of iterations is met. We provide in the next section the experimental results obtained using the proposed technique.

V. Experimental Results and Comparison

The proposed algorithm has been tested using a benchmark consisting of 50000 randomly generated 2-variables 4-valued functions. This set of benchmark functions is used to evaluate the performance of the proposed algorithm as well as other existing techniques found in literature. Comparison is made based on the average number of basic gates, as a measure of the number of product terms, needed to synthesize a given MVL function.

TABLE 2: Possible Ant paths for the function in Fig. 1

Minterm		Possible path
Position	**Logic Value**	
0	0	(0,0)
1	2	(0,2), (1,1), (2,0)
2	3	(0,3), (1,2), (1,3), (2,1), (2,2), (2,3), (3,0), (3,1), (3,2), (3,3)
3	1	(0,1), (1,0)
4	0	(0,0)
5	2	(0,2), (1,1), (2,0)
6	3	(0,3), (1,2), (1,3), (2,1), (2,2), (2,3), (3,0), (3,1), (3,2), (3,3)
7	2	(0,2), (1,1), (2,0)
8	0	(0,0)
9	0	(0,0)
10	1	(0,1), (1,0)
11	3	(0,3), (1,2), (1,3), (2,1), (2,2), (2,3), (3,0), (3,1), (3,2), (3,3)
12	0	(0,0)
13	0	(0,0)
14	0	(0,0)
15	0	(0,0)

The ACO parameters used in the experiments are as follows:
(a) number of runs = 10,
(b) number of iterations = 200,
(c) Number of ants = 30.

The MAX-MIN ant system [39] is used to control the range of pheromone values in any possible path. Any stagnancy occurring during the iteration will be perturbed by forcing a pheromone initial value to all possible paths, hence, creating a chance for ant to try to explore new areas in the search space.

In the first few experiments, we tried to find out the best value for PW, pheromone evaporation rate (ρ) and pheromone range. Using the above parameters, we can see that the best performance in terms of quality of solution and stability of the algorithm can be achieved when $2 \leq PW \leq 3$ and ρ is equal to 0.05. Throughout our experiments, we use $PW = 2.5$.

Table 3 shows a comparison of the results obtained using the proposed technique with those obtained using the techniques proposed in [25], [27], and [37]. We choose to compare the proposed technique with these techniques because the latter two techniques use a multiplexer (MUX) as an additional

gate at the circuit output. In our proposed technique we add a *tsum* gat at the output. The first four techniques in Table 3 were shown to produce better results compared to those produced using the best reported DC-based technique [19].

From Table 3, we can see that on average the proposed ACO-DC technique results in a reduction is the number of gates (as a measure of the chip area) needed to realize a given 2-variable 4-valued function as compared to those needed using the techniques reported in [25], [27], and [37].

TABLE 3: Comparison Table

Algorithm	**Average # gates**
Minterm Injection [25]	7.09408
Multiple- Connected Pseudo Minterm Injection [27]	7.09276
MCPM (PI_SM method) [37]	7.09064
MCPM (PI_CM method) [37]	7.0705
The proposed ACO-DC approach	7.02906

VI. Concluding Remarks

In this paper, we have introduced a new hybrid ACO and DC heuristic algorithm for synthesis of multi-level MVL functions. The algorithm is based on using the ACO to decompose a given MVL function into a number of simpler sub-functions. The DC is then used to synthesise each of the obtained sub-functions. The performance of the proposed algorithm has been tested using a benchmark consisting of 50000 randomly generated 2-variable 4-valued functions and compared against existing comparable DC-based techniques. The results show that the proposed technique outperforms other existing techniques in terms of the average number of gates, as a measure of the chip area, needed to realize a given MVL function.

ACKNOWLEDGMENT

The author would like to acknowledge the financial support received from Kuwait University through funded Research Project # WI 02/07.

REFERENCES

[1] K. Naiff, D. Rich and K. Smalley. *A Four-State ROM Using Multilevel Process Technology*. IEEE Journal of Solid-State Circuits, April 1984, Vol. 19(2). pp. 174–179.

[2] Intel Strata TM Flash. Available at: http://www.intel.com/design/flash/isf/overview.pdf and http://www.intel.com/design/flash/isf/overview.pdf.

[3] T. Okuda, T. Murotani. *A four-level storage 4Gb DRAM*. IEEE Journal of Solid-State Circuits Volume 32(11), 1997, pp. 1743 – 1747.

[4] S. Sudirgo. Quantum and Spin-Based Tunneling Devices for Memory Systems. PhD thesis, Rochester Institute of Technology, May 2006. Available at: https://ritdml.rit.edu/dspace/bitstream/1850/2066/1/SSudirgoThesis052006.pdf (2006)

[5] S., Kawahito, M. Kameyama, T. Higuchi, H. Yamada,. *A 32×32-bit multiplier using multiple-valued MOS current-mode circuits*. IEEE Journal of Solid-State Circuits. Volume 23(1), Feb. 1988, pp. 124 – 132.

[6] J. Kim and S. Ahn, "High-speed CMOS demultiplexer with redundant multi-valued logic", International Journal of Electronics, 94, 2007, pp. 915-924.

[7] M. Bhardwaj and T. Srikanthan and C. T. Clarke. *VLSI Costs of Arithmetic Parallelism: A Residue Reverse Conversion Perspective* (1999). Available at:

http://euler.ecs.umass.edu/paper/final/paper-138.ps

[8] H. Osseily, A. Haider, and A. Kassem, "Implementation of RSA Encryption Using Identical Modulus Algorithm", Proceedings of the 3rd International Conference on Information and Communication Technologies: From Theory to Applications, April 2008, pp. 1-6.

[9] C. Files and M. A Perkowski, "Multi-valued functional decomposition as a machine learning method", Proceedings of the International Symposium on Multiple-Valued Logic (ISMVL '98), May1998, pp.173 -178.

[10] B. Zupan. *Machine Learning Based on Functional Decomposition*. PhD thesis, University of Ljubljana, Slovenia (1997).

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

[12] K. Current, "Multiple-Valued Logic Circuits", Computer Engineering Handbook, 2nd Edition, CRC Presss 2008, Vojin Oklobdzija (Editior), Digital Design and Fabrication, Chapter 8, pp. 1-25.

[13] M. Khan, "Synthesis of quaternary reversible/quantum comparators", Journal of Systems Architectures, vol. 54, no. 10, October 2008, pp. 977-982.

[14] P. Tirumalai, and J. Butler, "Minimization Algorithms for Multiple-Valued Programmable Logic Arrays", IEEE Transactions of Computers, vol. 40, no. 2, February 1991, pp. 167-177.

[15] M. Abd-El-Barr, Z. Vranesic, and S. Zaky, "Algorithmic Synthesis of MVL Functions for CCD Implementation", IEEE Transactions on Computers, vol. 40, no. 8, August 1991, pp. 977-986.

[16] M. Abd-El-Barr, G. Hamid, M. Hasan, "Synthesis of MVL functions using input and output assignments", IEE Proceedings- Circuits, Devices, and Systems, vol. 145, no. 3, June 1998, pp. 207-212.

[17] M. Abd-El-Barr and L. Al-Awami, "Iterative-based minimization of unary 4-valued functions for current-mode CMOS realization", Proceedings 34th International Symposium on Multiple-Valued Logic (ISMVL), Toronto, Canada, May 2004, pp. 315-320.

[18] P. Besslich, "*Heuristic Minimization of MVL functions: A Direct Cover Approach*. IEEE Transactions on Computers, vol. 35, no. 2, February 1986, pp. 134-144.

[19] G. Dueck and D. Miller, "A Direct Cover MVL Minimization Using the Truncated Sum", *Proceeding of the 17th international symposium on multi-valued logic*, 1987, pp. 221-227.

[20] G. Promper and J. Armstrong, "Representation of Multi-valued Functions Using Direct Cover Method", *IEEE Transactions on Computers, 30*(9), 1981, pp. 674-679.

[21] C. Yang and Y.-M. Wang, "A neighborhood decoupling algorithm for truncated sum minimization", *Proceedings, 20th International Symposium on Multiple Valued Logic*, 1990, pp. 153-160.

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

[23] B. Sarif, and M. Abd-El-Barr, "Synthesis of MVL Functions - Part I: The Genetic Algorithm Approach", *International Conference on Microelectronics*, ICM '06. 16-19 Dec. 2006, pp. 154 – 157.

[24] M. Abd-El-Barr and B. Sarif, "Synthesis of MVL Functions - Part II: The Ant Colony Optimization Approach*", International Conference on Microelectronics*, ICM '06. 16-19 Dec. 2006, pp.158 – 161.

[25] B. Sarif, and M. Abd-El-Barr, "Synthesis of MVL Functions Using Discrete Particle Swarm Optimization", *Proceedings, 2008 IEEE Swarm Intelligence Symposium, St. Louis, USA*, September 21-23, 2008.

[26] M. Abd-El-Barr and B. Sarif," Weighted and Ordered Direct Cover Algorithms for Minimization of MVL Functions", *Proceedings of the 37th International Symposium on Multiple Valued Logic*, 2007, pp. 48-53.

[27] B. Sarif and M. Abd-El-Barr, "Fuzzy-based Direct Cover Algorithm for synthesis of Multi-Valued Logic Functions", *Proceedings of IASTED International Conferen ce on Circuits and Systems*, Hawaii, USA, 2008, pp. 625-630.

[28] M. Abd-El-Barr, "Non-binary functional synthesis using ACO-based heuristic", International Journal of Electronics, vol. 95, no. 1, January 2008, pp. 39-56.

[29] C. Tasi and M. Marek-Sadowska, "Multilevel logic synthesis for arithmetic functions", Proceedings 33r Design Automation Conference (DAC), Las Vegas, USA, 1996, pp. 242-247.

[30] M. Jiang, Y. Jiang, Y. Li, A. Mishchenko, S. Sinha, T. Villa, and R. Brayton, "Optimization of multi-valued multi-level networks", Proceedings 32nd IEEE International Symposium on Multiple-Valued Logic (ISMVL), May 2002, pp. 1-10.

[31] J. Jiang and R. Brayton, "Software Synthesis from synchronous specifications using logic simulation techniques", Proceedings the 39th Annual Design Automation Conference, New Orleans, USA, 2002, pp. 319-324.

[32] T. Sasao, "Switching Theory for logic synthesis", Kluwer Academic Publishers, February 1999, ISBN: 0-7923-8456-3.

[33] M. Gao, J. Jiang, Y. Jiang, Y. Li, S. Sinha, and R. Brayton, "MVSIS: Software Synthesis from synchronous specifications using logic simulation techniques", Proceedings International Workshop on Logic Synthesis, (IWLS01), 2001.

[34] M. Abd-El-Barr and Louai Al-Awami.. *Analysis of Direct Cover Algorithms for Minimization of MVL Functions*. International Conference on Microelectronics (ICM 2003), Cairo, Egypt, Dec. 2003. pp. 308- 312.

[35] M. Dorigo and G. Di Caro, "New Ideas in Optimization", McGraw Hill, London, UK, 1999.

[36] M. Dorigo and T. Stutzle. *The Ant Colony Optimization Meta-heuristic: Algorithms, Applications and Advances*. 2002.

[37] B. Sarif and M. Abd-El-Barr, "The Use of Multiple Connected Pseudo Minterms in the Synthesis of MVL Functions", *Proceedings of the 39th International Symposium on Multiple Valued Logic*, May 2009, Okinawa, Japan, pp. 145-150.

[38] M. Dorigo, "Ant Colony Optimization", Scholarpedia, 2007, 2(3): 1461.

[39] T. Stuzle and H. Hoos, "MAX-MIN Ant System and local search for the traveling salesman problem", IEEE International Conference on Evolutionary Computation, April 1997, pp. 309-314.

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