Synthesis of MVL Functions – Part II: The Ant Colony Optimization Approach

Mostafa Abd-El-Barr
Department of Information Science CFW, Kuwait University
965-254-9408
mostafa@cfw.kuniv.edu

Bambang A. B. Sarif
Department of Computer Engineering
KFUPM, KSA
+966-3-8604920
sarif@ccse.kfupm.edu.sa

***Abstract*— In this paper, an Ant Colony Optimization (ACO) based algorithm for synthesis of Multiple-Valued Logic (MVL) functions is proposed. The algorithm is tested using 50000 randomly generated 2-variable 4-valued functions. The proposed approach was compared to the existing direct cover techniques [2][6][7] as well as the approach proposed in [9]. The results obtained show that the proposed algorithm outperforms other approaches in terms of the average number of product terms required to realize a given MVL function.**

***Keywords*— MVL, multi-valued logic, synthesis, and ACO.**

I. INTRODUCTION

As integrated circuits (IC) technology improves, conventional Very Large Scale Integration (VLSI) circuits using binary logic show certain limitations and face serious problems. These problems include large layout area for interconnections and increased power consumption. As a result, interest in multiple-valued logic (MVL) has been renewed and intensified in the hope to overcome some of the binary logic limitations.

Unfortunately, the MVL synthesis problem is more involved as compared to its binary counterpart. Consider, for example, synthesis of 2-variable 4-valued functions. There are $r^{(r^n)} = 4^{4^2} = 2^{32}$ such functions as compared to 2^4 binary functions. This indicates that the search space for finding optimal MVL function synthesis is so large. This makes exact minimization of MVL functions prohibitively expensive. It has been reported that with only 2 inputs, absolute minimization of MVL functions requires computation time on the order of days [17]. Therefore, a number of heuristic algorithms for producing near minimal sum-of products (PLA) realization of MVL functions have been introduced. These algorithms can be classified into five categories: iterative improvement [14], decomposition-based [1][12], decision diagram based approach [3][13], algebraic minimization approach [8][15], and direct cover-based approach [2][4][6][7][17].

Iterative heuristics offer the possibility of exploring larger solution space in arriving at near optimal solutions. A number of these techniques have been reported in the literature [5][9][16] [18][19].

In this paper, we introduce a heuristic technique based on Ant Colony Optimization for synthesis of multiple-valued logic (MVL) functions. The paper is organized as follows. In Section 2, we provide some background material. In Section 3, we introduce the ant colony optimization technique. In Sections 4 and 5, the experimental work, together with the results obtained, is presented. Some concluding remarks are drawn in Section 6.

II. BACKGROUND MATERIAL

An n-variable MVL function of radix r, f(X), is defined as a mapping f : Rn→R, where R={0,1,…,r-1} is a set of r logic values with r ≥ 2 and X={x1,x2,…,xn} is a set of n variables. The MVL operators reported in the literature include the minimum operator, min, e.g. min(5,2,3) = 2, the maximum operator, max, e.g. max(5,2,3) = 5, the truncated sum operator, tsum, defined as tsum(a1, a2, …, an) = a1 ⊕ a2 ⊕ … ⊕ an = min(a1 + a2 + … + an,r – 1), where ⊕ is the symbol used to indicate tsum, and the complement of a logic level l, defined as l' = (r – 1) – l. These are used, together with the literal operator, to synthesize MVL functions. The literal operator is defined as follows.

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

where a, b ∊ R, and a ≤ b.

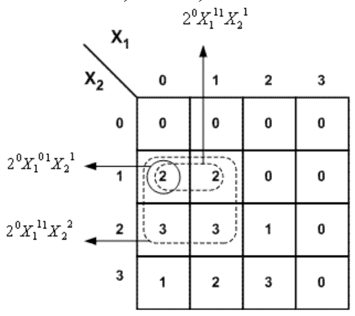

Fig. 1: A Tabular Representation of f(X1,x2).

Consider an example shown in Fig. 1, each cell in the box, which has a value other than 0, is a minterm, e.g. 2•0X10•1X21. The product term (implicant) I(X)= 2•0X11•1X22 is called a prime implicant, since no other implicant, I'(X), in f(X) such that I'(X) ≥ I(X). On the other hand, 2•0X11•1X21 is an implicant but not a prime implicant.

A functionally complete set of operators is the set capable of realizing all possible functions. A number of functionally complete sets have been used in synthesizing MVL functions. In terms of the set of MVL operators discussed above, the set consisting of {Literal, MIN, TSUM, Constant} is used in this paper.

III. ANT COLONY OPTIMIZATION

The Ant Colony Optimization (ACO) algorithm is a 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, and resource allocation problems [10][11].

The ACO algorithm has been inspired by the experiments using 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 known as stigmergy 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. After a while, all ants will select the shortest path. A general outline of the ACO algorithm is presented in Fig. 2.

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

Fig. 2: Ant Colony Optimization Algorithm

IV. ACO FOR MVL SYNTHESIS

A. *Solution Representation*

The basic idea of using ACO for 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 the 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.

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 values and heuristic value of that minterm (or implicant). This process is explained below.

B. *Minterm Selection*

In selecting a minterm, the ant needs the following information: pheromone value and a heuristic value. The pheromone value is obtained from pheromone trails of previous ant. Since we want to use solely the pheromone trail for minterm selection, the value of heuristic value is made constant. Thus, with τ_M represents the pheromone value of minterm M and η_M is the heuristic value of minterm M (a constant), the probability of choosing minterm M will be:

$$p = \frac{\tau_M \bullet \eta_M}{\sum \tau_M \bullet \eta_M}$$

C. *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 represents the pheromone value of implicant L and η_L is the heuristic value of literal L (a constant), the probability of choosing implicant L will be:

$$p = \frac{\tau_L \bullet \eta_L}{\sum \tau_L \bullet \eta_L}$$

It should be noted that we only consider implicant covering minterm that has been previously selected. In addition to that, to add more randomness in our implicant selection, an additional implicant 00000 is made available as one of ant's option to select implicant for previously selected minterm.

D. *Fitness Function Calculation*

The fitness function is divided into two parts. The first part is called Functional Fitness, Ff. The second part of the fitness function is called Objective Fitness, Fo. 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), Nh, between the two truth tables less the number of mismatch, Nm, between the two truth tables. This can be formulated as Ff = Nh – Nm. With Np is the number of product terms, the Objective Fitness is formulated as Fo = (100 – Np)/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

Top

fitness, then the value of objective fitness will determine which of these to use as the best solution.

Fig. 3 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.

```
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. 3: Pseudo code of ACO-MVL

Consider the example shown in Fig. 4. The function has three minterms, namely: 1•1X11•1X21, 2•2X12•1X21 and 3•3X13•1X21. These three minterms are the options for ant to select. Assume that the ant select minterm 1•1X11•1X21. Then the ant should select an implicant covering this minterm. Fig. 4 (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•1X13•1X21. 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. 4(b) and (c). When all minterms are covered, the ant will stop the selection process and return the function. The Example in Fig. 4 shows that the ant results a functions having the following implicants: 1•1X13•1X21, 1•2X13•1X21, and 1•3X13•1X21.

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.

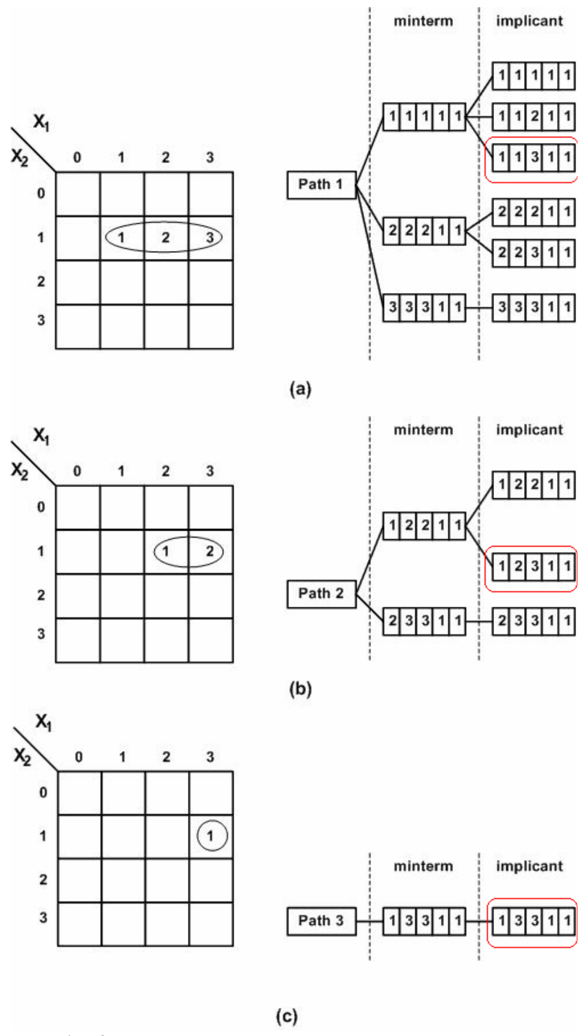

Fig. 4: Example of ACO-MVL

V. EXPERIMENTS AND RESULTS

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

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

The merits that are to be measured are:

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

The number of functions used as benchmark is 50000.

Table 1 shows a comparison of ACO-MVL with other techniques. The table clearly shows that the proposed

Top

approach outperforms other techniques.

TABLE 1: COMPARISON WITH DC TECHNIQUES

Algorithm	PT
ARM [7]	7.8901
BS [2]	7.9388
DM [6]	7.2478
ACO-MVL	6.9827

Table 2 shows a comparison of ACO-MVL with GA-MVL proposed in [9] for 200 2-variable 4-valued functions. The table shows that ACO-MVL is not only better in terms of quality of solution, i.e., average product term (PT) of the function, but also in terms of the successful rate (SR) value, which means ACO-MVL produce solutions ‘more often’ as compared to GA.

TABLE 2: COMPARISON WITH GA-MVL [9]

Algorithm	SR	PT
SGA-MVL [9]	0.45	7.24
RSGA-MVL [9]	0.58	7.18
ACO-MVL	0.997	6.925

In addition to that, the CPU time for ACO-MVL is much less as compared to GA-MVL. GA needs around 300 seconds to finish all 10 runs of 500 iterations to find a solution of one MVL function, while ACO-MVL needs around 10 seconds.

VI. CONCLUDING REMARKS

ACO algorithm for MVL synthesis has been proposed. The algorithm (ACO-MVL) was tested with 50000 randomly generated 2-variable 4-valued functions. The result of the proposed algorithm was compared to both existing DC techniques and GA-MVL approach proposed in [9]. It has been shown that the proposed approach outperforms all other techniques found in literature.

ACKNOWLEDGEMENT

The author would like to acknowledge the support of Kuwait University for the work reported in this paper through Research Grant # WI 05/04.

REFERENCES

[1] A. Mishchenko, B. Steinbach, M. Perkowski, ”Bi-decomposition of multi-valued relations”. Proc. IWLS’01, pp. 35-40.

[2] Besslich P. W. "Heuristic Minimization of MVL functions: A Direct Cover Approach." IEEE Transactions on computer, Feb1986, pp.134-144.

[3] 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.

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

[5] 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.

[6] 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.

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

[8] J.-H. J. M. Goa and R. Brayton. Optimization of multi-valued multi-level networks. Proceedings of the International Symposium on Multiple-Valued Logic, Tahoe City, June 2002.

[9] Mostafa Abd-El-Barr and Bambang A. B. Sarif, ”Genetic Algorithm in the Synthesis of MVL Functions”, Accepted, GECCO 2006, to be held July 2006.

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

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

[12] 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.

[13] Rolf Drechsler Mitch Thornton David Wessels, MDD-Based Synthesis of Multi-Valued Logic Networks

[14] 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.

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

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

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

[18] 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 (1998).

[19] 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).

Top