www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2. Issue 9 September 2013 Page No. 2702-2708

Towards Automated design of Combinational Circuits Using Evolutionary Techniques

K.Sagar[1] *and Dr.S.Vathsal*[2]

[1]Department of Computer Science and Engineering,
Chitanya Bharathi Institute of Technology, Hyderabad, INDIA
Email: kadapasagar@yahoo.com

[2]HOD (EEE) & Dean R&D, JBIET, Moinabad
Hyderabad, INDIA
Email: svathsal@gmail.com

Abstract: ***We introduce a technique, based on Evolutionary algorithms to automate and optimize the design of combinational circuits. Logic circuits are at the core of modern computing. The process of designing circuits which are efficient is thus of critical importance. By exploring the full range of possible solutions, circuits could be discovered which are superior to the best known human designs. Automated design techniques borrowed from artificial intelligence have allowed exactly that. Specifically, the application of genetic algorithms has allowed the creation of circuits which are substantially superior to the best known human designs. Systematic search is, perhaps, best exemplified by its simplest and most intuitive manifestation. This proposal expands on such previous research with a three-fold approach comprised of, A distinct optimizations for the application of genetic algorithms to design, the formulation and implementation of a systematic search technique to the problem and a comparison of the relative merits of the optimized genetic algorithm and the systematic search technique and the results also compared with Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) algorithms.***

Keywords: Evolutionary algorithms , Systematic Search, Particle Swarm Optimization, Ant Colony Optimization.

1. INTRODUCTION

Central to modern computing is the ability to perform logic. In its most fundamental form, the logic in computers is facilitated by digital logic circuits. Moreover, the basic components of these circuits are known as logic gates.

However, these gates are most often combined and interconnected in various ways to create more complex circuits. Logic design is one example of a discrete combinatorial system.

The characteristics of such a system are that it has a finite collection of discrete elements, which are combined to create new distinct objects. In the case of logic circuit design, gates are the discrete elements, and they are combined to create new circuits which function differently than any of the individual gates. By automating design, the goal is to remove human effort, and human limitations, from the design process. This can be done by taking advantage of what computers do very well, quickly examine a huge number of possible solutions.

In automatically designing logic circuits of this type, techniques from artificial intelligence have been extremely useful. Specifically, genetic algorithms have been highly researched as a candidate for automating circuit design. Moreover, there has been a good amount of success with using these algorithms. Genetic algorithms have been able to produce better results than human designers, and in a shorter period of time.

Systematic search to automate logic circuit design is a new idea. The specific search method employed is a version of BFS. The advantages and disadvantages associated with both types of search: systematic search and local search algorithms like Gas are discussed. The major advantage of systematic search methods like BFS is that they can return optimal solutions. Indeed, with careful problem formulation, one can ensure that BFS will return optimal solutions every single time. Fortunately, it is not difficult to formulate most problems so that BFS can meets these guarantees. Unfortunately, however, the benefits of BFS come at a high cost. The space and time requirements can be prohibitively large, making this method of searching totally impractical for some problem instances.

PSO is another technique for automating the circuit design which is a robust stochastic optimization technique based on the movement and intelligence of swarms. When the search space is too large to search exhaustively, population based searches may be a good alternative, however, population based search techniques cannot guarantee you the optimal (best) solution. A population based search technique, Particle Swarm Optimization (PSO) Algorithm shares similar characteristics to Genetic Algorithm, and however, the manner in which the two algorithms traverse the search space is fundamentally different.

Ant Colony Optimization (ACO) algorithm is a new meta-heuristic algorithm with a combination of distributed computation, auto-catalysis (positive feedback) and constructive greedy heuristic in finding optimal solutions for combinatorial problems. The ACO algorithm has been inspired by behavior 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.

2. PREVIOUS WORK

The design process for combinational logic circuits has evolved from its first notions [1] to a standard element of undergraduate computing curricula [6]. Standard graphical design aids such as Karnaugh Maps [5] are widely used and tool suitable for computer implementation have evolved from the QuineMcCluskey Method [4].

Louis [7] is one of few sources found in the literature to address the use of GAs for the combinational logic design problem. Louis combines knowledge- based systems with the genetic algorithm, making use of a genetic operator called masked crossover that adapts to the encoding being able to exploit information unused by classical crossover operators [8]. His results, although very encouraging for certain examples, but do not seem to have solved the combinational circuit design problem completely. However his idea of incorporating knowledge about the domain in the genetic operator constitutes a big step toward increasing the power of the GA as a design tool. Unfortunately, the incorporation of knowledge in to the GA decreases its usefulness as a general search tool. Louis overcomes this problem by defining an operator that he claims to be domain independent, but whose efficiency turns out to depend on the representation used.

Koza [6] has used genetic programming to design combinational circuits. He has designed, for example, a two-bit adder, using a small set of gates (AND, OR, NOT), but his emphasis has been on generating functional circuits rather than on optimizing them. In fact, this is also the case in Louis' research, where the main focus was to provide an easier way to generate functional designs using the GA rather than in optimizing a functional design according to certain metrics. In more recent work, has focused more towards the design of an a log circuits in which the goal is to produce their appropriate topology and size so that they are functional given a certain set of components. So far, genetic programming has been considered a more powerful tool in such tasks, because the representation it uses is more powerful for structural design in general.

Miller et al [9] developed (independently) an approach similar to ours, but using a more compact representation that instead of considering the inputs and gates as completely separate elements in the chromosome string, use a single gene to encode a complete Boolean expression. Miller's notation does not decrease the total length of the chromosome, but it increases the cardinality of the alphabet needed, having as its main drawback the lack of flexibility of the representation to handle a larger number of inputs

3. GENETIC ALGORITHMS FOR LOGIC CIRCUIT DESIGN

Up to the present, most research has focused on using local search algorithms for the design of logic circuits. More specifically, genetic algorithms have been the most common choice.

In order to use GAs for this purpose, though, there must be some additional formulation of the problem. As we have seen, GAs use strings as their basic elements, in the same way that biological systems use DNA strands. Therefore, if we are to use GAs for circuit design, all of the information about gates and connections must be encoded in a string. In accordance with the terminology from biology, this string is known as the "genotype".

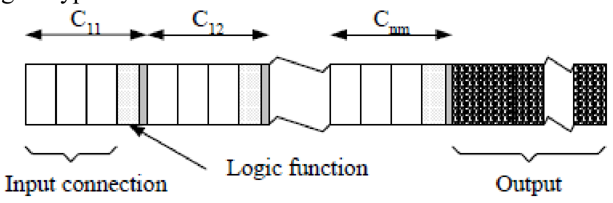


Fig. 3.1 Representation of genotype

Genotype is structured from phenotype shows in fig. 3.1. Cells in phenotype are lined from C11 to Cnm and finish by outputs for set to genotype.

The genotype is an encoding of all the relevant information about the circuit. The relevant information, which is encoded, is known as the "phenotype".

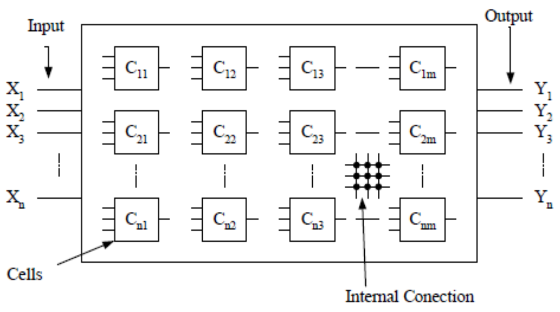


Fig. 3.2 Representation of phenotype

Phenotype consists of inputs, cells, internal connections, and outputs shown in Fig. 3.3. Inputs are input signals of a digital logic circuit. Each cell is a logic gate which is connected thru internal connection.

The phenotype includes the gates used in the circuit, the connections between gates and other essential properties. The phenotype can be derived from the genotype, and in turn, the operation of the circuit can be derived from the phenotype.

Attempts were made to explore other circuit formulations , but the one which has gained the most favor is the array formulation. In this formulation, a circuit is conceptualized as an array of logic gates and connections between them.

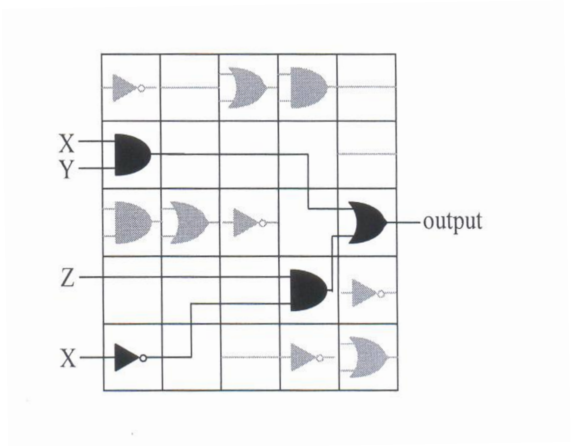


Fig. 3.3 Array formulation

This is conceptually similar to the way an FPGA is structured. At one end of the array are presented the inputs to the circuit, and at the other end of the array are the outputs. Each gate at a particular location is a member of an array column, and it can get its inputs from any gates in the previous column. The gates in the left-most column get their inputs from any of the circuit inputs, rather than any gates. The benefits of applying GAs to logic circuit design have been as good as expected. By automating the entire process, GAs have been able to quickly develop circuits which are fully functional. Moreover, some circuits which have been developed are superior to those designed by humans.

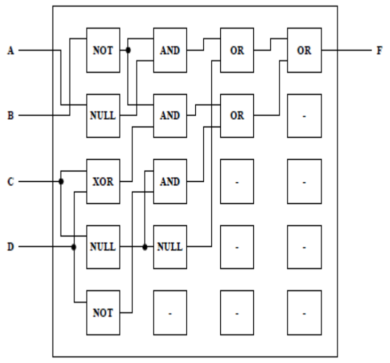


Fig. 3.4 Example for Array Formulation of Logic Gates

The Fig. 3.4 describes an example of array formulation for two dimensional template, Gates gets its input from one of the gates from the previous columns, From the figure 3.4, Second column and first of the AND gate is getting the input from an not gate and the other input is directly connected from the input variable ‘A’ and similarly the other gates in the array gets inputs from the gates in the previous columns and the output is produced by the last gate from the last column of the array.

Genetic algorithm have been a developing technology which is been used in every sector. Circuit designing is the new concept applied using genetic algorithm.GA is a trial and error technique; we can take advantage of the speed capability of computers to perform thousands of these trials and converge upon an optimal solution

Truth table which contains data in the form of 0’s and 1’s is converted into variables and stored into string which is can be referred as expression, since the last column of the truth table contains function values these are separated from the expression . The function values which contain 1’s are stored and remaining are discarded. After the regular expression is formed, then the expression is sub divided based on the ‘OR’ operator (which joins the expression), the sub expression formed can be referred as genes. Genetic algorithm is applied on these genes, the algorithm includes fitness function, mutation value and finally the population list.

The genetic algorithm approach is mainly based on its genetic operators like iteration, the number of times the loop is being repeated the probability of getting the appropriate solution is high. But genetic algorithm is a stochastic process where the chance of getting the exact solution is 50/50.

The generational process is repeated until termination condition is reached, the common terminating conditions are:

-Fixed number of generation reached.

-An individual is found which satisfies all the minimum condition-Highest ranking individual’s fitness is reached or has a plateau such that further iterations does not produce better results

-Combinations of above.

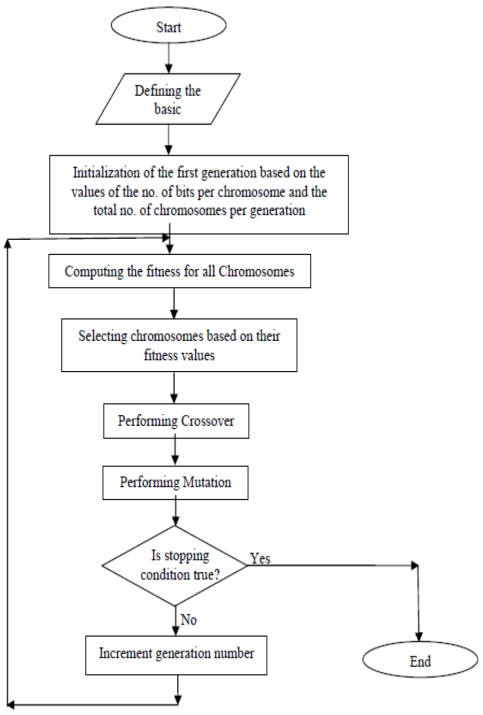

Start
Defining the basic
Initialization of the first generation based on the values of the no. of bits per chromosome and the total no. of chromosomes per generation
Computing the fitness for all Chromosomes
Selecting chromosomes based on their fitness values
Performing Crossover
Performing Mutation
Is stopping condition true?
Yes
No
Increment generation number
End

Fig. 3.5 Design of the Genetic algorithm for Circuit designing

4. SYSTEMATIC SEARCH

Systematic search is, perhaps, best exemplified by its simplest and most intuitive manifestation. Specifically, the type of search known as breadth-first search (BFS). It illustrates the ideas of systematic search very nicely, including all of its strengths and weaknesses. BFS explores all possible options at each step, until a solution is found.

Not only is BFS systematic, but it is exhaustive. It considers all possible search paths at each step. Therefore, it will always find a solution if one exists. This is a very desirable property. Moreover, by considering nodes closer to the root first, BFS will always find the solution which is shallowest in the tree.

4.1 Representing Systematic Search

The new formulation hinges on one important idea: the path to the goal must represent the desired circuit.

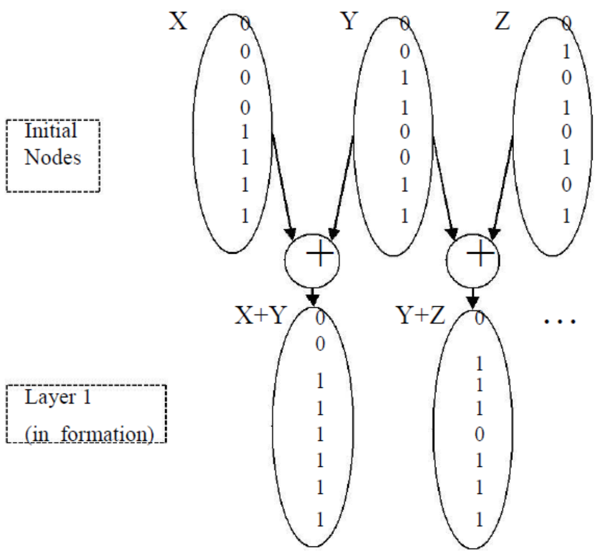
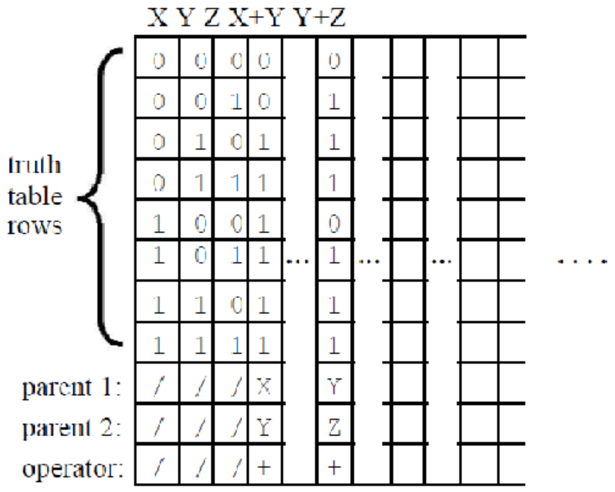


Figure 4.1 Systematic search as applied to logic circuit

design. The initial nodes are the input variables. Nodes are combined via logical operators to create new nodes

If the path to the goal represents a finished circuit, then it may be unclear what the search nodes represent. It may not be immediately obvious why, but in this research the search nodes were conceptualized as truth-table columns. Initially, there are several nodes present in the search tree. Each initial node represents the truth-table column for one of the input variables (figure 6.1). Just as a conventional truth table is seeded with columns that represent every possible combination of the inputs, so is the search space initially seeded with search nodes representing the truth-table columns of the inputs. To forward the search, nodes are not expanded in the conventional sense in this formulation. Rather, new nodes are formed by selecting two existing nodes and a logic operation. This is similar to the way new columns are formed in a conventional truth table. After selecting, the logic operation is then applied to the corresponding rows in the chosen nodes, and a new truth-table column is formed which constitutes a new search node.

4.2Implementing Systematic Search



	X	Y	Z	X+Y	Y+Z
truth table rows	0	0	0	0	0
	0	0	1	0	1
	0	1	0	1	1
	0	1	1	1	1
	1	0	0	1	0
	1	0	1	1	1
	1	1	0	1	1
	1	1	1	1	1
parent 1:	/	/	/	X	Y
parent 2:	/	/	/	Y	Z
operator:	/	/	/	+	+

Figure 4.2 Systematic search as it was implemented using a 2-D array to store the nodes.

This systematic search strategy can be implemented quite easily using a two dimensional array(figure 6.2). Each array column can hold a truth table column from a single node. Extra rows can be made below each column to hold the additional information about parents, depth size, etc. the first columns
of the array are initially seeded with the truth-table columns of the inputs. Creating new nodes is simply a matter of selecting two columns and a logical operator, and applying the operator to each row of the columns. When new nodes are created, their respective truth-table columns can be placed in the next open column of the array.

5. PARTICLE SWARM OPTIMIZATION TECHNIQUE

In PSO, a swarm of particles "fly" through the search space. Each particle keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. This value is called pbest . Another "best" value that is tracked by a particle is the best value, obtained so far by any particle In the neighbors of the particle. This location is called lbest . When a particle takes all the population as its topological neighbors, the best value is a global best and is called gbest.

The PSO concept consists of, at each time step, changing the velocity of (accelerating) each particle toward its pbest and lbest locations (local version of PSO). Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward pbest and lbest locations. PSO was first introduced in 1995. It is a very efficient stochastic optimization tool for optimization problems. Recently, more and more researchers have been attracted by this promising research area. It is demonstrated that PSO gets better results in a faster, cheaper way compared with other methods Another reason that PSO is attractive is that there are few parameters to adjust. One version, with slight variations, works well in a wide variety of applications. PSO has been used for approaches that can be used across a wide range of applications, as well as for specific applications focused on a specific requirement.

5.1 PSO Algorithm:

1. Particle start from a random point
2. 2 types of information's available to the particle for taking decision where to move next:
   a. Own Search Experience (particle's pervious history & its own optimal state)
   b. Performance of the particles in the neighborhood (best states reached by its neighbors' so far)
3. For this is uses the following formula –

$P(x_{id}(t)=1)=f(x_{id}(t-1), v_{id}(t-1), P_{id}, P_{gd})$ (1)

Where:

- $P(x_{id}(t)=1)$ is the probability that individual *i* will choose 1 for the bit at the d-th position of the binary string.
- $x_{id}(t)$ is the current state of the string position *d* of individual *i*.
- t refers to the current iteration.
- $v_{id}(t-1)$ is a measure of the individual's predisposition or current probability of deciding 1.
- P_{id} is the best state found so far.
- P_{gd} is the best state found in the neighborhood so far.

5.2 Representation of the circuit:

In this the matrix representation is used to encode a circuit as in work of the Genetic Algorithm. Such representation is shown in Figure 4.1. This matrix is encoded as a fixed length string of bits or integers from 0 to N – 1, where N refers to the number of rows allowed in the matrix (we call it n-cardinality alphabet). In this, we will be referring to the Genetic Algorithm (GA) that uses an n-cardinality alphabet, since it has been found in the past that this version of the algorithm consistently produces better results than its binary counterpart.

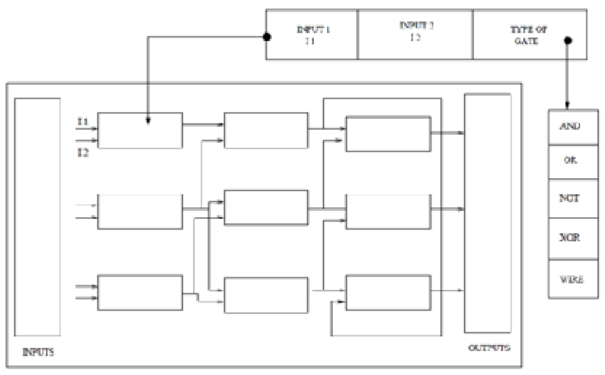


Fig. 5.1 Matrix used to represent a circuit. Each gate gets its input from either of the gates in the previous column. Note the encoding adopted for each element of the matrix as well as the set of available gates used.

More formally, we can say that any circuit can be represented as a bi-dimensional array of gates $S_{i,j}$, where j indicates the level of a gate; so that those gates closer to the inputs have lower values of j. (Level values are incremented from left to right in the Figure 4.1). For a fixed j, the index i varies with respect to the gates that are "next" to each other in the circuit, but without being necessarily connected. Each matrix element is a gate (there are 5 types of gates: AND, NOT, OR, XOR and WIRE1) that receives its 2 inputs from any gate at the previous column as shown in Figure 1. Although our implementation allows gates with more inputs and these inputs might come from any previous level of the circuit, we limited ourselves to 2-input gates and restricted the inputs to come only from the previous level. This restriction could, of course, be relaxed, but we adopted it to allow a fair comparison with the GA-based approach.

Input 1	Input 2	Gate Type

Fig. 5.2 Encoding used for each of the matrix elements that represent a circuit

A chromosomic string encodes the matrix shown in Figure 4.2 by using triplets in which the 2 first elements refer to each oftheinputs used, and the third is the corresponding gate from the available set.The fitness function works in two stages: first, it maximizes the number of matches. However, once feasible solutions are found, we maximize the number of WIREs in the circuit. By doing this, we actually optimize the circuit in terms of the number of gates that it uses. The main goal is to produce a fully functional design (i.e., one that produces all the expected outputs for any combination of inputs according to the truth table given for the problem) which maximizes the number of WIREs.

The main motivation for using particle swarm optimization (PSO) to design combinational circuits is that this algorithm has been found to be very efficient in a variety of tasks.

6. ANT COLONY OPTIMIZATION

In ACO algorithm, the optimization problem is formulated as a graph G = (C; L), where C is the set of components of the problem, and L is the possible connection or transition among the elements of C. The solution is expressed in terms of feasible paths on the graph G, with respect to a set of given constraints.

A circuit is modeled as a matrix M of size n * m. The content of matrix M is dynamically filled.

Consider the Boolean function f = xyz + xyz + xyz. Figure 5.1 shows a graph of some possible paths connecting literal x to the intended function f. Assume that the ants start the tour from literal x. The ant will traverse the paths by selecting the edges through a probabilistic process. Assume that the goal is to find the shortest path to repre-sent function f. Therefore, the ants that found the path f = x'yz + xy'z + xyz' would return the best representation for function f.

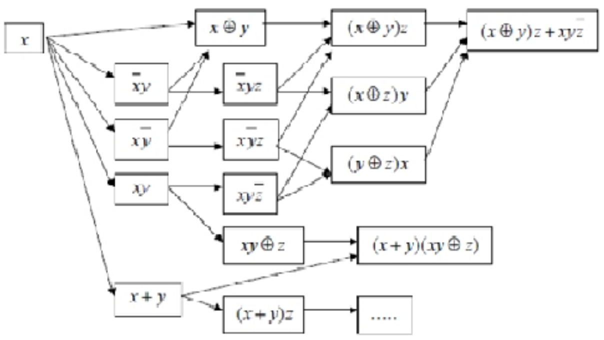


Fig.6.1 Some of the possible paths in the function *f*.

At first, matrix M is filled with randomly generated cells. Then, each ant will traverse the matrix. These ants originate from a dummy cell called nest (see Figure 5.2), and traverse each state (a cell in a column) until it reaches the last column or a cell that has no successor.

The selection edges to traverse is determined by a stochstic probability function. It depends on the pheromone value (τ) and heuristic value (η) of the edge (or the next cell).

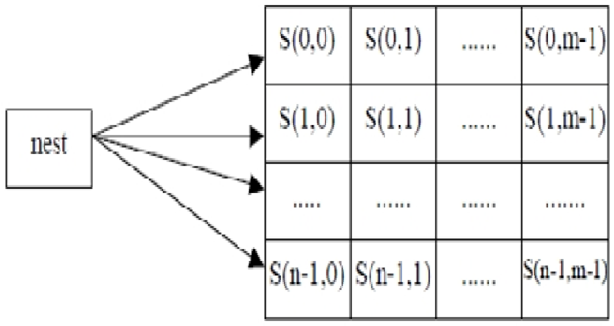


Fig.6.2 Nest cell and matrix *M* for ant to be traversed.

7. Experimental Results

Consider the truth table

Z	W	X	Y	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Our PSO algorithm found a solution with a fitness value of 34 (i.e., a circuit with 7 gates) 20% of the time, and feasible circuits were found 67% of the time. The average fitness of the 20 runs performed was 29.35, with a standard deviation of 7.4. The graphical representation of the best solution found is depicted in Figure 7.1.

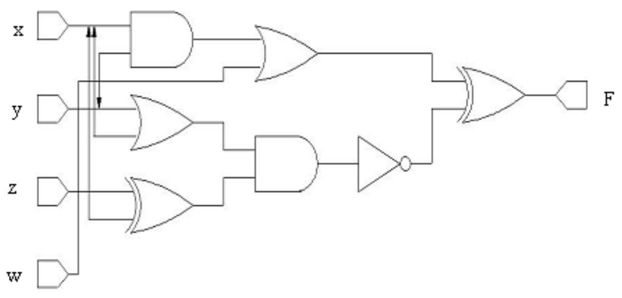


Fig. 7.1 Graphical representation of the best circuit found by PSO.

The comparison of the results produced by PSO, an *n*-cardinality GA (NGA), a human designer (using Karnaugh maps), and Sasao's approach [13] are shown in Table 7.1. Sasao has used this circuit to illustrate his circuit simplification technique based on the use of ANDs & XORs. His solution uses, however, more gates than the circuit produced by our approach.

GA	Human Designer	PSO
$F = (WYX^I \oplus ((W+Y) \oplus Z \oplus (X + Y + Z)))$	$F = ((Z^I X) \oplus (Y^1 W^1)) + ((X^1 Y)(Z \oplus W^1))$	$F = (XY + W) \oplus ((Z \oplus X)(X + Y))$
10 gates	11 gates	7 gates
2ANDs, 3ORs, 3XORs, 2NOTs	4ANDs, 1OR, 2XORs, 4NOTs	2ANDs, 2OR, 2XORs, 1NOTs

Table: 7.1 Comparison of results between PSO, an *n*-cardinality GA (NGA), and human designer for the circuit.

The graphical representation of the circuit for the above truth table by ACO algorithm is shown in figure 7.2.

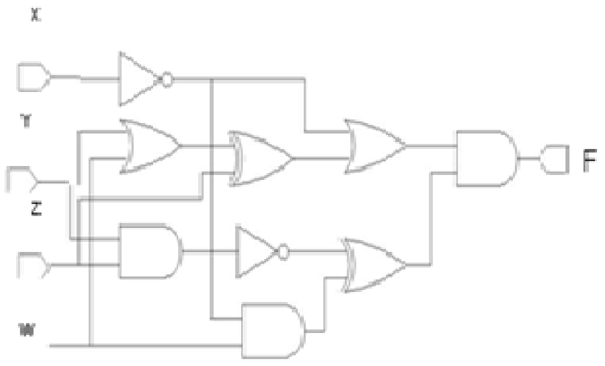


Fig.7.2 Graphical representation of the best circuit found by ACO algorithm.

The comparison of the results produced by the ACO, a genetic algorithm with GA, a human designer (using Karnaugh maps), and in the following Table 7.2. In this case, the ACO found a solution slightly better than the GA.s.

The parameters used by the GA for this example were the following: Crossover rate = 0.5, mutation rate = 0.0022, population size = 2000, maximum number of generations = 400. Convergence to the solution shown for the GA in Table 7.2 was achieved in generation 328. The matrix used by the BGA was of size 5 × 5.

GA	Human Designer	ACO
$F = (W Y X' \oplus ((W + Y) \oplus Z \oplus (X + Y + Z)))$	$F = ((Z' X) \oplus (Y' W')) + ((X' Y) (Z \oplus W'))$	$\mathbf{F = (((W+Y)} \oplus Z) + X') ((YZ') \oplus (X'W))$
10 gates	11 gates	9 gates
2ANDs, 3ORs, 3XORs, 2NOTs	4ANDs, 1OR, 2XORs, 4NOTs	3ANDs, 2OR, 2XORs, 2NOTs

Table: 7.2 Comparison of results between the ACO algorithm, the GA, and human designer for the circuit.

Circuits produced by the depth-concerned and size concerned versions of systematic search is shown in figure 7.3

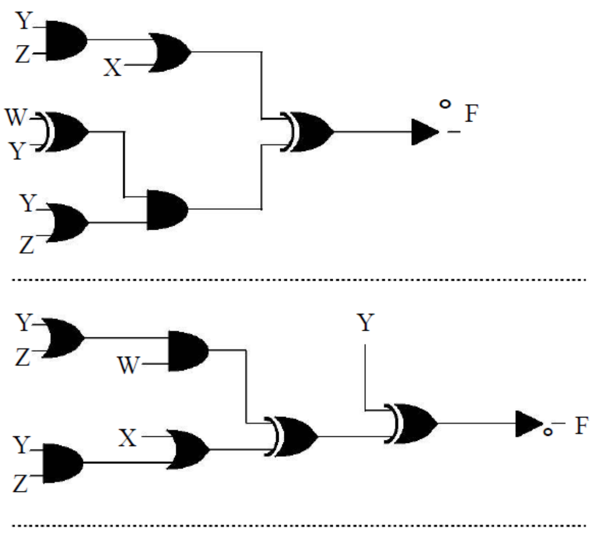


Figure 7.3 Circuits produced by the depth-concerned(above) and size-concerned(below) versions of systematic search.
The function is F(W,X,Y,Z)={0,1,3,6,7,8,10,13}.

Method	**No. of Variables**	**Expression**	**Number of Gates**
Systematic Search	**4**	$F= (((Y+Z)\,W \oplus (X+YZ)) \oplus Y)'$	**7**
Human design	**4**	$F=((Z'\cdot X)+(Y'\cdot W'))+((X'\cdot Y)(Z\oplus W'))$	**11**

8. CONCLUSION

This paper presented how genetic algorithm, PSO technique and ACO can be used to design combinational logic circuits. Systematic and Local search techniques of artificial intelligence are studied and have been applied to the problem of genetic based logic circuit design.
We have implemented genetic algorithm using all genetic operators on an input for circuit designing, these genetic operators include selection, fitness function, crossover and mutation
A computer program has been developed which can reduce the number of gates on a particular input .We compared the results produced by our genetic algorithm approach against those generated by Minimization tool.
The PSO technique is implemented for the circuit designing. The obtained result by PSO is comparable with GA in most of the cases and outperforms the Human Designers (Karnaugh Maps and Quine-McCluskey Procedures) in all the cases. POS method can produce the global optimal solution but the drawback of this method is the process duration become longer for more complicated structure.
ACO technique is presented to optimize the combinational logic circuits at the gate level. Results compared fairly well with those produced with a GA and are better than those obtained using Karnaugh maps and the QuineMcCluskey Procedure. Current ACO implementation is limited to circuits of smaller size and produces better results compared to Genetic Algorithm

The systematic search approach is also presented to find the optimal circuit. The major benefit of using systematic search is a guarantee that optimal solutions will be found. Indeed, this technique can be used to find the best known circuits for any specified functions

References

[1] Uthman Salem al-saiari, **"**Digital Circuit Design Through Simulated evolution" , King Fahd University of petroleum and minerals , Dhahran, Saudi Arabia, November 2003

[2] Russell, S. and Norvig, "Artificial Intelligence: A Modern Approach. Prentice Hall", New Jersey. (2003).

[3] Carlos A. CoelloCoello, Alan D. Christiansen, Arturo Hernandez Aguirre: "Design of Combination Logic Circuits through an Evolutionary Multi-objective Optimization Approach" Department of Electrical Engineering and Computer Science, Tulane University, New Orleans, LA, USA, 2000

[4] McCluskey E. J.: "Minimization of Boolean functions" Bell System technical Journal, (1996).

[5] Karnaugh M.: "A map method for synthesis of combinational logic circuit" Transactions of the AIEE, Communications and Electronics,(1993).

[6] Koza,J.R ,"Genetic Programming on the programming of computers by means of natural selection" ,the MIT press, Cambridge, Massachusetts. (1992)

[7] Louis, and Rawlins, G.Designer Genetic algorithms: "Genetic algorithms in structure design" , In Belew, R.K. and booker,L.B.(eds) proceedings of the fourth international conference on genetic algorithms, San mateo, California, Morgan Kaufmann Publishers. (1991)

[8] Tutorials for Genetic Algorithm ;http://www.obitko.com/tutorials/genetic-algorithms

[9] Miller, J.F., Thompson, P. and Fogarty, "Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study." Genetic Algorithm and Evolution Strategy in Eng. and Comp. Sci., 105-131T. (1997).

Author Profile

K.Sagar received the B.E degree in Electronics & Communication Engineering from Chaitanya Bharathi Institute Of Technology, Gandipet, Hyderabad in 1991 and M.Tech degree in Computer Science and engineering from JNTU , Hyderabad in 1999 and persuing Ph.D in computer Science & Engineering from Osmania University, Hyderabad. Currently working as an Associate Professor in the department of Computer Science & Engineering ,C.B.I.T , Hyderabad,A.P,India.