State assignment for Sequential Circuits using Multi-Objective Genetic Algorithm

Journal:	*IET Computers & Digital Techniques*
Manuscript ID:	CDT-2010-0045.R2
Manuscript Type:	Research Paper
Date Submitted by the Author:	n/a
Complete List of Authors:	AL JASSANI, BAN; Edinburgh Napier University, Schoole of Engineering and Built environment
Keyword:	BOOLEAN FUNCTIONS, CAD, CIRCUIT OPTIMISATION, DIGITAL CIRCUITS, FINITE STATE MACHINES, GENETIC ALGORITHMS, LOGIC CIRCUITS, SEQUENTIAL CIRCUITS, STATE ASSIGNMENT, SWITCHING CIRCUITS

State assignment for Sequential Circuits using Multi-Objective Genetic Algorithm

B. A. AL JASSANI
b.al-jassani@napier.ac.uk

N. URQUHART
n.urquhart@napier.ac.uk

A.E.A. ALMAINI
a.almaini@napier.ac.uk

Edinburgh Napier University
Edinburgh, UK

ABSTRACT

In this paper, a new approach using a Multi Objective Genetic Algorithm (MOGA) is proposed to determine the optimal state assignment with less area and power dissipations for completely and incompletely specified sequential circuits. The goal is to find the best assignments which reduce the component count and switching activity. The MOGA employs a Pareto ranking scheme and produces a set of state assignments, which are optimal in both objectives. The ESPRESSO tool is used to optimise the combinational parts of the sequential circuits.

Experimental results are given using a personal computer with an Intel CPU of 2.4 GHz and 2 GB RAM. The algorithm is implemented using C++ and fully tested with benchmark examples. The experimental results show that saving in components and switching activity are achieved in most of the benchmarks tested compared with recent published research.

1. INTRODUCTION

In sequential logic circuits, the output at any given time is function of both present and past inputs. Therefore additional logic is necessary to remember the state of the circuit. Sequential circuits can be represented by a combinational circuit in conjunction with memory elements as in Fig. (1).

Sequential circuits can be classified into two categories; synchronous (clocked) and asynchronous (unclocked). This paper is concerned with synchronous sequential circuits where the transition between states is controlled by a clock pulse.

A Finite State Machine (FSM) is a mathematical model of the sequential circuit with discrete inputs, discrete outputs, and internal states. Synthesis tools are required to give each state a specific binary code. The state assignment is one of the most important problems which received a great deal of attention from researchers.

There are two different types of FSM, depending on the output transition function, namely, the Moore and Mealy models. In the Moore model, the outputs depend on the states only while in the Mealy model, the outputs depend on the inputs as well as on the states.

The complexity of the sequential circuit depends on the state assignment. Different assignments generate networks with different complexities. The state assignment refers to the allocations of the binary codes to the states of the sequential circuits. The resulting combinational logic and the switching between the states depend on the codes assigned to the states. One of the best known techniques which were used for state assignments is that of partitions and decomposition [1], but not all state machines have useful closed partitions and may be minimised using these techniques.

In [2], a new approach is proposed utilizing a Genetic algorithm (GA) with Evolvable Hardware (EHW) to produce optimal logic circuits. In [3-6], the authors proposed the use of GA to generate state assignments which minimise the gate count and/or power dissipation.

A new comprehensive method consisting of an efficient state minimisation and state assignment technique is presented in [7].

The authors of [8] proposed a new approach to the synthesis problem for finite state machines with the reduction of power dissipation as a design objective. A finite state machine is

decomposed into a number of coupled sub machines. Most of the time, only one of the sub machines will be activated which, consequently, could lead to savings in power consumption.

In [9], the authors present heuristic algorithms for state minimisation of FSM's for a large class of practical examples**.** The authors discuss two steps of the minimisation procedure, called state mapping and solution shrinking, which play a significant role in delivering an optimally implemented reduced machine. A decomposition of sequential machines is outlined in [10]; this paper discusses the theory of general decomposition of incompletely specified sequential machines to realize the behaviour of the machine.

In [11], the authors present a heuristic for state reduction of incompletely specified finite state machines (ISFSMs). The proposed heuristic is based on a branch-and-bound search technique and identification of sets of compatible states of a given ISFSM specification. A new FSM partitioning approach for low power using GA is presented in [12].

In [13], a new (m-block) partitioning technique for the state assignment is proposed for testabilities and power consumption. In [14], the usage of a stochastic search technique inspired by simulated annealing is explored to solve the state assignment problem.

Generally, it is possible to find state assignments to minimise the hardware only [1-3, 15], or the power dissipation only [4, 12, 16, 17]. It is known, however, that minimising either the power or logic complexity could be at the expense of the other and in most cases it is not possible to find a solution that is optimum in both domains. For large circuits, there are millions or possibly billions of assignments [18] and hence it is possible to find assignments that minimise either the logic or power.

As the title suggests, this paper employs GA adopting the Pareto Ranking scheme [19, 20] to find state assignments that minimise both the hardware and power dissipation of the state machine. The MOGA algorithm used in this paper employs multi-objective GA to find assignments that reduce both the hardware and power dissipation due to switching activity

and leaves it to the designer to give the priority to either power dissipation or logic complexity or select a compromise solution that reduces both but not guarantee absolute minimum in either.

The remainder of the paper is organized as follows. Section 2 defines the state assignment for the sequential circuits. Section 3 explains the multi-objective GA. The proposed algorithm is described in Section 4. Experimental results and conclusions are given in sections 5 and 6, respectively.

2. STATE ASSIGNMENT

A state machine having *n* distinct states and x inputs, requires $\mathrm{s} = [\log_2 n]$ state variables and $\mathcal{L} = [\log_2 x]$ input variables for the complete assignments, where [g] is defined as the smallest integer equal to or greater than g. The total number of the different possible encodings [18] is given by L (n) as defined by equation (1).

$$\mathrm{L}(n) = \frac{2^{\mathrm{S}}!}{(2^{\mathrm{S}} - n)!} \tag{1}$$

while the total number of unique state assignments [18] is given by $\mathrm{A}(n)$ as defined by equation (2).

$$\mathrm{A}(n) = \frac{(2^{\mathrm{S}} - 1)!}{(2^{\mathrm{S}} - n)!\ \mathrm{s}!} \tag{2}$$

The total number of unique assignments is large and has many local minima; e.g. FSMs with 10 states, have 75675600 different assignments.

The problem is how to find an efficient state assignment, in terms of switching and hardware, among the very large number of assignments, without resorting to exhaustive search.

An incompletely specified sequential circuit is one in which at least one state transition edge from some state is not specified. These states are called don't-care (DC) conditions [18] and represented using "-" in the State Transition Table (STT).

The power consumption [21] of a sequential circuit is proportional to its switching activity which can be represented by equation (3)

$$P_{ave} = \frac{1}{2} C_L V_{dd}^2 \ f_{clk} \ E_{sw} \quad (3)$$

where C_L is the physical capacitance of the output for the node, V_{dd} is the supply voltage, E_{sw} is the expected switching activity, and f_{clk} is the clock frequency. Since the register capacitance is fixed and cannot be affected, therefore; we consider the switching activity E_{sw} as cost function C which is one of the proposed objectives.

$$\mathrm{C} = \sum_{i,j \in S} tp_{(i \leftrightarrow j)} \ HD_{i,j} \quad (4)$$

where $HD_{i,j}$ represents the Hamming Distance between the coding of the two states $s_i \ and \ s_j$, and $tp_{(i \leftrightarrow j)} = tp_{ij} + tp_{ji}$ and tp_{ij} is defined as the total state transition probability from states s_i to states s_j.

The Hamming Distance (HD) [17] between two Boolean vectors a_i, b_i is defined by the number of bits in same position $a_i, b_i$with different phases as in equation (5).

$$HD_{a,b} = \sum_{i=0}^{n-1} ai \oplus bi \quad (5)$$

State assignments that result in a lower E_{sw} value and lower number of terms to structure the combinational circuit are considered to be optimal assignments. The switching activity and logic complexity of sequential circuits heavily depend on the code assigned to the states which is influenced by the HD between codes of the states.

The total state transition probability tp_{ij} between two states $s_i \& s_j$, defined as the probability that the transition from s_i to s_j , occurs in an arbitrary sequence and can be calculated using equation (6).

$$tp_{ij} = P_i\, p_{ij} \tag{6}$$

Steady state probability P_i of state s_i is defined as the probability that the state is visited within an arbitrary random sequence.

$$p_{ij} = Prob\left(Next = s_j \,\middle|\, Present = s_i\right) = \frac{N_{ij}}{\sum_k N_{ik}} \tag{7}$$

where p_{ij} represents the conditional state transition probability, N_{ij} is the number of transitions from s_i to s_j , ($i, j \in \{0,1,2,3, \dots, n-1\}$ where n is the number of states). While $\sum_k N_{ik}$ is all transitions that begin with state s_i.

$$\sum_{i=0}^{i=n-1} P_i = 1 \tag{8}$$

$$P_i = \sum_{j=0}^{j=n-1} P_j\, p_{ji} \tag{9}$$

The steady state probabilities P_i can be calculated by solving these set of linear equations using Gaussian elimination methods. The calculations of these parameters are further explained in [17].

3. MULTI-OBJECTIVE GENETIC ALGORITHM

In GA terminology, a solution vector is known as individual or chromosome which comprises a number of discrete units called genes. Population is defined as a collection of chromosomes. The population is normally initialised randomly.

The parents are selected from existing chromosomes in the population according to their fitness. The chromosomes which are better in their fitness will have more chance to be selected as parents to produce a child chromosome than others. The fitness function produces a fitness value based on the genes; this value represents the chromosome efficiency to solve the problem.

Crossover and Mutation operators are used by the GA to generate new solutions from existing ones. Crossover combines two chromosomes to produce one new child chromosome. The mutation operator takes a chromosome and then alters random genes of it. Mutation helps to prevent the population from converging to a local optimum.

Replacement operation replaces the worst chromosome in the population with child chromosome. By iteratively applying the crossover and mutation operators, chromosomes with good genes are expected to appear more frequently in the population. The pseudo code for the GA in the general form is shown in Fig. (2).

Single objective optimisation seeks to find the best (highest or lowest) value of the defined objective. For many problems, there is a need for simultaneous optimisation of possibly conflicting objectives. Therefore, if there are two objectives to be optimised, it might be possible to find two solutions; one of these solutions being optimal in terms of the first objective while the other is the optimal for the second objective [20].

Multi-objective GA's may be applied to many complex engineering optimisation problems**.** A number of different evolutionary algorithms were suggested to solve multi-objective optimisation problems [22, 23].

In this research, there are two objectives to be optimised. Using a Pareto scheme [20, 24], it is convenient to classify all the potential solutions into dominated and non-dominated (Pareto optimal set) solutions. "The solution $\vec{u}$ is dominated if there is a feasible solution $\vec{v}$ not worse than $\vec{u}$ for all objectives f_h $(h = 1, \ldots\ldots, r)$, where r is the total number of objectives. If a solution is not dominated by any other feasible solution, we call it non-dominated (or Pareto optimal set) solutions" [20]. This could be expressed in mathematical form by equation (10)

$$f_h(\vec{u}) \leq f_h(\vec{v}) \quad for\ all \quad 1 \leq h \leq r \tag{10}$$

There is another approach for multi objective optimisation using a method of objective weighting [20], which is simplified by combining the multiple objectives into a single

composite function using weighted sum method. The weight w for each objective h is $(0 < w_h < 1), \sum_1^r w_h = 1$, and different weight vectors lead to different solutions. The problem becomes one of finding the solution which minimizes $\sum_1^r w_h f_h$.

The proposed algorithm produces a set of optimal solutions (known as Pareto-optimal solutions), instead of a single optimal solution. Without knowing what the user requirements are, it cannot be said that any one of these solutions is better than the other. Therefore, the proposed algorithm has the ability to find multiple Pareto-optimal solutions in one single simulation run. The MOGA used in this paper has two objectives, the first being to reduce the number of components required to design the combinational part of the sequential circuit. The second is to reduce the switching activities.

Example 1: Consider the benchmark Lion which has 4 states. The state transition graph (STG) for this example is shown in Fig. (3). The conditional state transition probability p_{ij} is calculated by equation (7). Using equations (8) and (9) and Gaussian elimination method, steady state probabilities P_i can be obtained. These conditional and steady state probabilities are calculated as shown in Fig. (4). The FSM –STT in Table (1) consists of four symbolically encoded states *ST0, ST1, ST2, and ST3*. These states can be assigned unique codes using two state variables $y_1 and\ y_2$. The inputs can be represented by x_1 and x_2 and the single output is represented by Z. The next states are represented by $y_1^+ \ and\ y_2^+$. Two different assignments are displayed in Table (2).

The Cost C as function of switching activity for this example can be calculated as follows:

$$C = HD_{s0,s1} * (tp_{01} + tp_{10}) + HD_{s1,s2} * (tp_{12} + tp_{21}) + HD_{s2,s3} * (tp_{23} + tp_{32})$$

Table (1) FSM - STT representation of Lion benchmark

Present states y_1, y_2	Next states (y_1^+, y_2^+) / Output Z			
	(y_1^+, y_2^+) / Z	(y_1^+, y_2^+) / Z	(y_1^+, y_2^+) / Z	(y_1^+, y_2^+) / Z
	x_1x_2 *00*	x_1x_2 *01*	x_1x_2 *11*	x_1x_2 *10*
ST0	ST0 / 0	ST1/-	ST0/0	ST0/0
ST1	ST1 / 1	ST1/1	ST0/0	ST2/1
ST2	ST1 / 1	ST3/1	ST2/1	ST2/1
ST3	ST3 / 1	ST3/1	ST2/1	-/-

Table (2) Different assignments

states	1st assignment	2nd assignment
ST0	00	10
ST1	01	01
ST2	11	11
ST3	10	00

For the first assignment, the Cost C as function of switching activity is:

$$C = HD_{s0,s1}(tp_{01} + tp_{10}) + HD_{s1,s2}(tp_{12} + tp_{21}) + HD_{s2,s3}(tp_{23} + tp_{32})$$

$$= HD_{s0,s1}(P_0p_{01} + P_1p_{10}) + HD_{s1,s2}(P_1p_{12} + P_2p_{21}) + HD_{s2,s3}(P_2p_{23} + P_3p_{32})$$

$$= 1\,(0.06667 + 0.06667) + 1(0.06667 + 0.06667) + 1(0.06667 + 0.0666)$$

$$= 0.4$$

The following file (PLA format) is produced by the proposed algorithm for the first assignment to be ready for minimisation using ESPRESSO [25].

```
.i 4  // Primary inputs and present states
.o 3  // Primary outputs and next states
.p 11 // Number of product terms
-000 000                          1-11 111
1100 000                          0011 011
0100 01-                          0111 101
0-01 011                          0-10 101
1101 000                          1110 111
1001 111                          .e
```

After minimisation, this assignment results in the following:

```
.i 4
.o 3
.p 6
10-1 100                              010- 011
111- 010                              -0-1 011
0-10 101                              -11- 101
.e
```

Considering the sharing of terms, this implementation requires 6 terms. The equations for this circuit after minimisation are as follows:

$$y_1^+ = x_1\overline{x}_2y_2 + \overline{x}_1y_1\overline{y}_2 + x_2y_1$$

$$y_2^+ = x_1x_2y_1 + \overline{x}_1x_2\overline{y}_1 + \overline{x}_2y_2$$

$$\mathrm{Z} = \overline{x}_1y_1\overline{y}_2 + \overline{x}_1x_2\overline{y}_1 + \overline{x}_2y_2 + x_2y_1$$

For the second assignment, the Cost C as function of switching activity is:

$$C = HD_{s0,s1}(tp_{01} + tp_{10}) + HD_{s1,s2}(tp_{12} + tp_{21}) + HD_{s2,s3}(tp_{23} + tp_{32})$$

$$= 2\ (0.06667 + 0.06667) + 1\ (0.06667 + 0.06667) + 2\ (0.06667 + 0.0666)$$

$$= 0.666$$

The following file (PLA format) is produced for the second assignment to be ready for minimisation using ESSPRESSO [25].

```
.i 4   // Primary inputs and present states
.o 3   // Primary outputs and next states
.p 11  // Number of product terms
-010 100                1101 100                0111 001
1110 100                1001 111                0-00 001
0110 01-                1-11 111                1100 111
0-01 011                0011 011                .e
```

After minimisation, this assignment results in the following:

```
.i 4
.o 3
.p 10
0110 010                1-1- 100                0--1 001
-010 100                0-0- 001                -0-1 011
1100 111                1--1 100                .e
0-01 010                1-11 011
```

The equations for this circuit after minimisation are as follows:

$$y_1^+ = \overline{x}_2 y_1 \overline{y}_2 + x_1 x_2 \overline{y}_1 \overline{y}_2 + x_1 y_1 + x_1 y_2$$

$$y_2^+ = \overline{x}_1 x_2 y_1 \overline{y}_2 + x_1 x_2 \overline{y}_1 \overline{y}_2 + \overline{x}_1 \overline{y}_1 y_2 + x_1 y_1 y_2 + \overline{x}_2 y_2$$

$$Z = x_1 x_2 \overline{y}_1 \overline{y}_2 + \overline{x}_1 \overline{y}_1 + x_1 y_1 y_2 + \overline{x}_1 y_2 + \overline{x}_2 y_2$$

Considering the sharing of terms, this implementation requires 10 terms. Therefore; the first assignment is better in both objectives for this example. Optimal state assignments for circuits with large number of states become computationally complex as well as crucial for larger FSMs.

In this paper, two level logic implementation is adopted and ESPRESSO tools are used to generate the combinational logic.

4. THE PROPOSED ALGORITHM

The MOGA is proposed to optimise the state assignment for completely and incompletely specified sequential circuits without doing an exhaustive search. The aim is to identify the good state assignments which can be used to design the circuit with fewer components and reduced switching activity simultaneously. The MOGA algorithm is implemented in C++. It is tested with 15 benchmark examples of up to 48 states. The search space of the proposed algorithm is defined by equation (1).

The proposed algorithm for finding the best state assignment represents a solution by a chromosome containing the code for each state of the sequential circuit. The chromosome is represented using decimal numbers. The length of each chromosome equals to 2^s , where s is number of state variables. Each gene in the chromosome holds the decimal code for the states used including the DC states.

For example, a circuit with 6 states; requires 3 flip flops. Therefore; this circuit has two DC states. The length of the chromosomes = 2^3=8 bits. If the chromosome is | 3 4 2 1 0 6 5 7 |, then the state assignment is shown in Table (3).

The MOGA with the two objectives has two fitness functions. The first fitness function "*Fitness_term* (c_i)" calls the ESSPRESSO tool [25] to minimise the combinational part of the circuit and to produce the terms for the minimised circuit. The second fitness function "*Fitness_ switching (ci)*" calculates the switching activity as given by equation (4).

Table (3) One possible state assignment

States	Chromosome	assignment
ST0	3	011
ST1	4	100
ST2	2	010
ST3	1	001
ST4	0	000
ST5	6	110
(DC) ST6	5	101
(DC) ST7	7	111

The Pareto ranking is integrated into the proposed algorithm by replacing the chromosome fitness by the Pareto ranks. This scheme is based on several layers of classifications [20]. All non dominated solutions are given rank one. Figure (5) gives the pseudo code of the proposed algorithm.

The GA uses a tournament selection method where the main parameter of selection is the tournament size *(T)* which can be changed by the operator. A number of individuals (*T*) are selected from the population randomly and the one with the smaller rank (i.e. best rank) is then used as the selected individual.

Crossover is the principle genetic operator. Uniform crossover shown in Fig. (6) is adopted. A string of binary bits is initialized by the proposed algorithm randomly. The length of this string equals to the length of the chromosome. This string determines which genes are copied from the first parent and which genes are copied from the second parent. The child inherits

the gene from the first parent if the corresponding bit in the string is zero while the child inherits the gene from the second parent if the corresponding bit in the string is one. Continuous check is required before the inheritance for each gene avoiding the repetition of the same coding for different states which is not allowed. The mutation operator swaps the positions of two randomly chosen genes as shown in Fig. (7).

Replacement strategy controls the composition of the new generation for each evolution loop. The proposed algorithm uses a tournament replacement method which is simplified by randomly choosing T individuals (independently of their ranks) from the population and replacing the chromosome which has the biggest rank (i.e worse rank) with the new offspring generated. The successful application of GA depends on the diversity of the whole population in the search space. It may be difficult for GA to find the global optimum solution, if it couldn't hold its diversity well, and sometimes results in the premature convergence to the local optimum solution. Premature Convergence is one of the major problems associated with GA. It means that all the chromosomes in the population have the same fitness. To prevent the premature convergence and to avoid the loss of genetic diversity of the whole population, the algorithm will not replace the new chromosome if there is another individual having the same fitness in the current populations. A usual strategy is to stop evolution after a fixed number of evaluations, which is determined by the user.

Example 2: Consider the FSM-STT for the benchmark bbtas which has 6 states as shown in Table (4). Figure (8) shows 17 different assignments randomly initialized which result in different number of terms and switching activities. It is clear that some assignments produce the same number of terms with different switching activity like assignments 4, 5, 7 and 12. Table (5) shows the codes and ranks for the different assignments shown in Fig. (8).

Table (4) FSM - STT representation of bbtas benchmark

Present states y_1, y_2	Next states (y_1^+, y_2^+)/ Outputs Z_1Z_2			
	(y_1^+, y_2^+)/ Z_1Z_2	(y_1^+, y_2^+)/ Z_1Z_2	(y_1^+, y_2^+)/ Z_1Z_2	(y_1^+, y_2^+)/ Z_1Z_2
	x_1x_2 00	x_1x_2 01	x_1x_2 11	x_1x_2 10
ST0	ST0/00	ST1/00	ST1/00	ST1/00
ST1	ST0/00	ST2/00	ST2/00	ST2/00
ST2	ST1/00	ST3/00	ST3/00	ST3/00
ST3	ST4/00	ST3/01	ST3/11	ST3/10
ST4	ST5/00	ST4/00	ST4/00	ST4/00
ST5	ST0/00	ST5/00	ST5/00	ST5/00

The first five ranks are shown in Fig. (9). Solutions which are the best either in number of terms, switching activity or both have rank one.

Table (5) Codes and ranks for different state assignments for example (2)

Solution Number as in Fig. (8)	Codes for different state assignments st_7 st_0	Terms	Switching Activity	Ranks
1	0 5 3 1 4 2 6 7	10	0.56	1
2	1 4 5 6 0 2 3 7	12	0.56	3
3	1 0 6 4 3 2 7 5	12	0.717	5
4	4 2 0 3 6 7 1 5	11	0.717	3
5	3 1 7 4 5 0 2 6	11	0.56	2
6	5 7 0 1 3 2 4 6	12	0.6	4
7	0 5 3 1 6 4 2 7	11	0.769	4
8	6 4 5 1 7 0 3 2	14	0.834	10
9	4 2 5 6 7 1 3 0	13	0.73	7
10	2 7 0 3 4 1 5 6	15	0.847	11
11	4 7 6 1 0 3 5 2	13	0.939	9
12	0 2 4 5 1 3 7 6	11	0.44	1
13	1 0 2 4 7 5 6 3	14	0.76	9
14	4 0 7 2 5 6 1 3	13	0.873	8
15	0 6 3 2 1 4 7 5	13	0.717	6
16	4 2 0 5 6 1 3 7	12	0.79	6
17	4 7 6 5 1 2 3 0	14	0.7304	8

After one run of the proposed algorithm, it produces three results as below.

0 5 3 1 4 2 6 7 terms = 10 switching activities =0.56
6 1 5 4 0 3 2 7 terms = 9 switching activities =0.613
0 2 4 5 1 3 7 6 terms = 11 switching activities =0.44

The user has the choice to select one of these results depending on requirements. The time required to produce these solutions is one minute only. The results are obtained using population size=30, tournament size=3 and 300 for the number of evaluations. These parameters are determined after testing various population sizes and different tournament sizes.

5. EXPERIMENTAL RESULTS

The program is applied to several MCNC benchmark functions. The algorithm is implemented using C++ and is tested using a PC with INTEL CPU, 2.4 GHz clock and 2GB RAM. Test results are given in Table (6). ESSPRESSO is used to minimise the circuit for each state assignment. The second column in Table (6) denotes the number of inputs, number of output and number of states for the given benchmark in the first column. The set of results produced by the MOGA is giving in column 3. *PT* denotes to number of product terms and *C* refers to the cost as function of switching activity. It is obvious that the number of solutions produced is different from one example to another and depends on how many non dominated solutions having rank one are produced by the proposed algorithm.

In Table (6), the comparison is made between the results produced by the proposed algorithm and the results published by other references as shown in columns 4, 5 & 6.

Our results are compared with NOVA tool [26] results, which were published in [27]. It is obvious that MOGA results are better than NOVA results in most cases. From Table 6, first set of MOGA results for all benchmarks tested, it can be seen than on average MOGA produces results requiring 21% fewer product terms and 15% less switching activity compared to NOVA.

Reference [4] developed GA for finding good assignment to minimise area and power for the FSM. The author combined the two objectives into a single composite function using

Table (6) Experimental results for the benchmark

Benchmarks	In/out/ No. of states	Results of MOGA		Result of NOVA[27]		Results of Ref. [4]		Results of Ref.[5]	Saving compared to Ref[4]		Saving compared to Ref[5]	Time
		PT	C	PT	C	PT	C	PT	PT	C		
bbtas	2/2/6	9 10 11	0.613 0.56 0.44	8	0.815	---	---	9	---	---	0%	1 min.
bbara	4/2/10	22 27	0.49 0.39	24	0.459	22	0.317	23	0% -18%	-24% -20%	4%	8 min.
opus	5/6/10	15 17 16	0.49 0.49 0.488	16	0.809	15	0.556	12	0% -25%	12% 12%	-20%	40 min.
Lion9	2/1/9	10	0.34	9	--	---	---	11	---	---	9%	8 min.
Dk16	1/2/27	57 68 59	2.1 1.64 1.7	72	--	---	---	68	---	---	16%	6 hours & 3 min.
keyb	7/2/19	46 47 55	0.98 0.75 0.54	48	1.469	46	0.674	46	0% -16%	-31% 20%	0%	3 hours & 32 min.
Cse	7/7/16	43 49 54	0.39 0.32 0.30	46	0.602	43	0.355	45	0% -20%	-9% 15%	4%	3 hours & 9 min.
donfile	2/1/24	22 26	1.375 1.29	28	1.75	36	1.6	31	39% 27%	14% 19%	29%	6 hours & 4 min.
Ex1	9/19/20	48 49 51	0.78 0.63 0.621	44	1.338	52	0.842	47	8% 2%	7% 26%	-2%	6 hours & 7 min.
Ex4	6/9/14	13 14	0.568 0.468	19	1.310	14	0.421	15	7% 0%	-25% -10%	13%	6 hours & 1 min.
Modulo 12	1/1/12	10 11	0.75 0.58	12	1.00	12	0.583	10	8% 8%	-22% 0%	0%	5 hours & 56 min.
S1	8/6/20	43 53 60	1.37 1.19 1.04	80	1.698	66	1.48	68	35% 9%	-7% 30%	37%	6 hours
S1a	8/6/20	29 30	1.21 1.174	80	--	---	---	66	---	---	56%	5 hours & 19 min.
stry	9/10/30	78 79 84 88	1.1 0.93 0.736 0.674	94	1.278	88	0.943	78	11% 0%	-14% 29%	0%	6 hours & 5 min.
Planet	7/19/48	81 82 86 87	2.49 2.09 1.79 1.69	87	2.833	86	2.24	84	6% -1%	-10% 33%	4%	25 hours & 23 min.

weighted sum method. Table (6) shows that the proposed algorithm can achieve more saving, compared to reference [4], especially for large functions.

Reference [5] presented a GA for finding good assignment to reduce the area requirement. Comparing our results and results obtained from this reference, it is found that our results could save cubes in most examples tested with reduction in the switching activities. It is also obvious that saving in cubes becomes larger for the large functions, (56% in one case).

The time required to produce the good assignment is different for each example and depends on the complexity of the circuit. The time required by the proposed algorithm is large due to the fact that MOGA has to communicate with ESSPRESSO to minimize the logical expressions. For each evaluation of the GA, the proposed algorithm calls the ESSPRESSO to minimise the circuit for each assignment. Even allowing for this overhead, the time required to produce a good assignment is still acceptable. It is in the range of 1 minute for the circuit with 6 states to 25 hours for the circuit with 48 states.

6. CONCLUSIONS

In this paper, a Multi Objective Genetic algorithm approach to the state assignment problem is adopted with the aim of minimizing gate count and power dissipation for completely and incompletely specified sequential circuits. The target for this algorithm is to find the best assignments which have less hardware with reduced switching activity to minimise the power dissipation and the area simultaneously. The Pareto ranking scheme has been integrated with the genetic algorithm by creating a set of integral ranks for all chromosomes in the population which are used by the GA as fitness.

Table [6] compares the switching activity and number of terms produced by NOVA [26] with the results produced by the proposed algorithm. The results show that the proposed algorithm

produces better results in either switching activity or number of terms or both for most benchmark.

Next comparison is made between the proposed GA with previously published work. The results show that the proposed algorithm using Pareto ranking scheme achieves saving in either number of cubes or in switching activity in most examples as compared with references [4] and [5]. From the comparison, it can be seen that the saving in cubes of the MOGA increases with the increase of the number of states for the tested benchmark functions.

One of the advantages of using the Pareto ranking scheme with the MOGA is producing more than one solution and giving the choice to the user depending on whether the user wants less number of cubes or less switching activity or in between to design the circuit. The other advantage of integrating the proposed algorithm with Pareto ranking scheme as opposed to the weighted sum method can be seen from the savings which are achieved by our results compared with the results of reference [4].

Further, testing shows that MOGA can find good assignments in a reasonable time in all the examples attempted compared with the long time required by exhaustive search. The MOGA requires from 1 minute to 25 hours to find the good assignment for benchmark with 6 states to 48 states respectively.

REFRENCES

[1] Hartmanis J. and Stearns R.E. :'Algebraic Structure Theory of Sequential Machines', Prentice-Hall, Inc. Englewood Cliffs N.J. , 1966.

[2] Ali B.., Almaini A. E. A., and Kalganova T.,: 'Evolutionary Algorithms and Their Use in the Design of Sequential Logic Circuits', Genetic Programming and Evolvable Machines, 2004, 5, pp. 11-29.

[3] Almaini A.E.A., Miller J.F., Thomson P., and Billina S.: 'State assignment of finite state machines using a genetic algorithm', IEE Proc. Computer and Digital Tech. , 1995, 142, (4) ,pp. 279-286.

[4] Yinshui X., Almaini A.E.A., and Xunwei Wu.: 'Power Optimization Of Finite State Machines based on Genetic Algorithm', Journal of Electronics, 2003, 20, (3), pp. 194- 201.

[5] Chattopadhyay S. : 'Area Conscious State Assignment with Flip Flop and Output Polarity Selection for Finite State Machine Synthesis Genetic Algorithm Approach', The Computer Journal, 2005, 48, (4),pp.443-450 .

[6] Olson E. P.: 'Optimal State Assignment of Sequential Circuits using A Genetic Local Search with Flexible Cost Functions', PHD thesis, Urbana-Champaign, University of Illinois, 1995.

[7] Shiue W.-T. : 'Novel state minimization and state assignment in finite state machine design for low power portable devices', Integration, the VLSI journal , 2005, 38, pp. 549-570.

[8] Chow S., Ho Y., Hwang T., and Liu C.: 'Low Power Realization of Finite State Machines—A Decomposition Approach', ACM Transactions on Design Automation of Electronic Systems, 1996, 1, (3) , pp. 315–340.

[9] Rho J. –K., Hachtel G. , Somenzi F., and Jacoby R.: 'Exact and Heuristic Algorithms for the Minimization of Incompletely Specified State Machines', IEEE Trans. on Computer-aided design of integrated circuits and systems, 1994, 13, (2), pp. 167-177.

[10] Jóźwiak L. , Ślusarczyk A.: 'General decomposition of Incompletely specified sequential machines with multi-state behaviour realization', Journel of systems architecture, 2004, 50 , pp. 445-492.

[11] Go¨ren S. and Ferguson F. J.: 'On state reduction of incompletely specified finite state machines', Computers and Electrical Engineering, 2007, 33 , pp. 58–69.

[12] Xia Y., Ye X., Wang L., Tao W. and Almaini A.E.A.: ' A Uniform Framework of Low Power FSM Partition Approach', IEEE International Conference on communication , circuits and systems, China , 2006, pp. 2642-2646.

[13] Cho S. and Park S.: 'A new synthesis technique of sequential circuits for low power and Testing', Current Applied Physics, 2004,1, pp.83-86.

[14] Aly W. M.: 'Solving the State Assignment Problem Using Stochastic Search Aided with Simulated Annealing', American J. of Engineering and Applied Sciences , 2009, 2, (4): pp. 710-714.

[15] Almaini, A. E. A.: 'Sequential Machine implementations using universal logic Modules', IEEE Transactions on Computers, 1978, C-27,(10), pp 951-960.

[16] Tsui, C. Pedram, M., Chen, C. and Despain, A. M.:' Low power state assignment targeting two and multi level logic implementations' ICCAD 1994 , pp 82-87.

[17] Benini L., Micheli G. De. : 'State assignment for Low Power Dissipation', IEEE Custom Integrated Circuits Conference, 1994, 30, (3), pp. 136-139.

[18] Dolotta T.A., McCluskey E.J.: 'The Coding of Inernal States of Sequential Circuits', IEEE Trans. on Electronic Computers, EC13, 1964, pp 549-562.

[19] Goldberg D.E.: 'Genetic Algorithms in Search, Optimisation, and Machine Learning' , Addison Wesley, 2003.

[20] Michalewicz Z., Fogel D.B. :'How to Solve it: Modern Heuristics', Springer, Berlin 2004, 2nd edition.

[21] Chattopadhyay S. , Reddy P.N.: 'Finite state machine state assignment targeting low power consumption', IEE Proceeding Computer Digital Technology, 2003, 151, (1) , pp. 61-70.

[22] Deb K., Pratap A., Agarwal S., and Meyarivan T.: 'A Fast and Elitist Multi objective Genetic Algorithm: NSGA-II', IEEE Trans. on evolutionary computation, 2002, 6, (2), pp182-197.

[23] Abdullah K. , David W. C., Smith A. E.: 'Multi-objective optimization using genetic algorithms: A tutorial', Reliability Engineering and System Safety 91, 2006, pp. 992–1007.

[24] Michalewicz Z., 'Genetic Algorithms + Data Structures = Evolutionary Programs', Springer, Berlin/London, 1996.

[25] http://sontrak.com/download_esp.aspx, Sontrak, Technical software, accessed March 2010.

[26] Villa T., Sangiovanni-Vincentelli A.,' NOVA: State assignment of finite state machine for optimal two-level logic implementation. IEEE Trans. Computer_Aided Design of Integrated Circuits and Systems, 1990, Vol. 9, (9), pp.905-924.

[27] Hong S.K. , Park I.C. , Hwang S.H. , and Kyung C.M.:' State assignment in finite state machines for minimal switching power consumption', IEE Electronic letter, 1994, Vol. 30 , (8), pp 627-629.

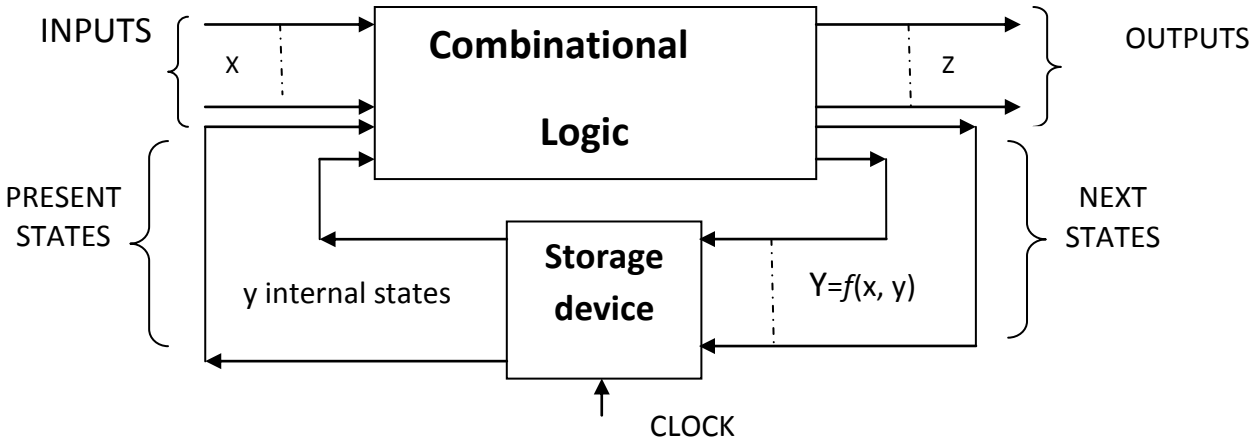

Figure (1) Sequential circuit

```
Procedure GA
{
     Create an initial population of random genes
      Evaluate all Chromosomes         // find their fitness

Repeat
     {
      Select chromosomes with the best fitness to reproduce
      Apply Crossover operator
      Apply Mutation operator
      Evaluate the new Child          // Find its fitness
      If (Child Fitness!= any existing Fitness)//!= indicates not equal
             Apply Replacement operator
     } Until termination condition

} End GA
```

Figure (2) Pseudo code for general form of GA

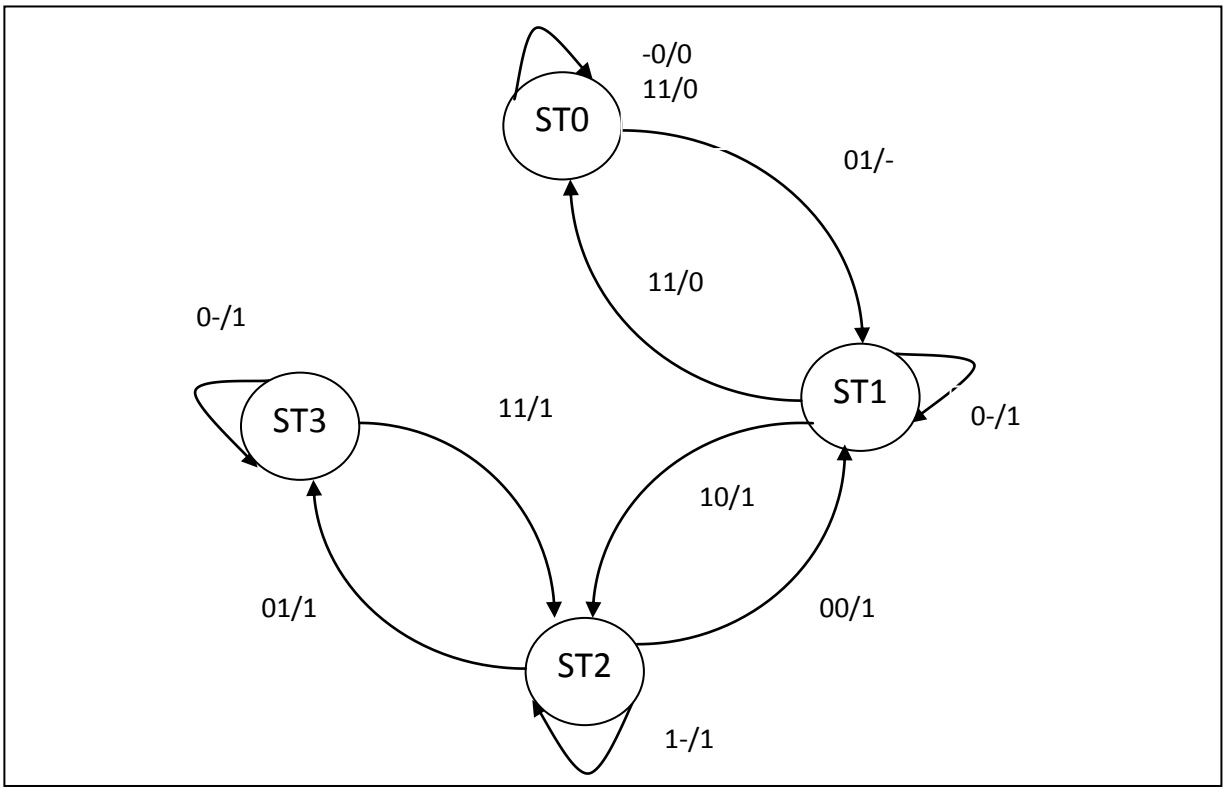

Figure (3) STG of example (1)

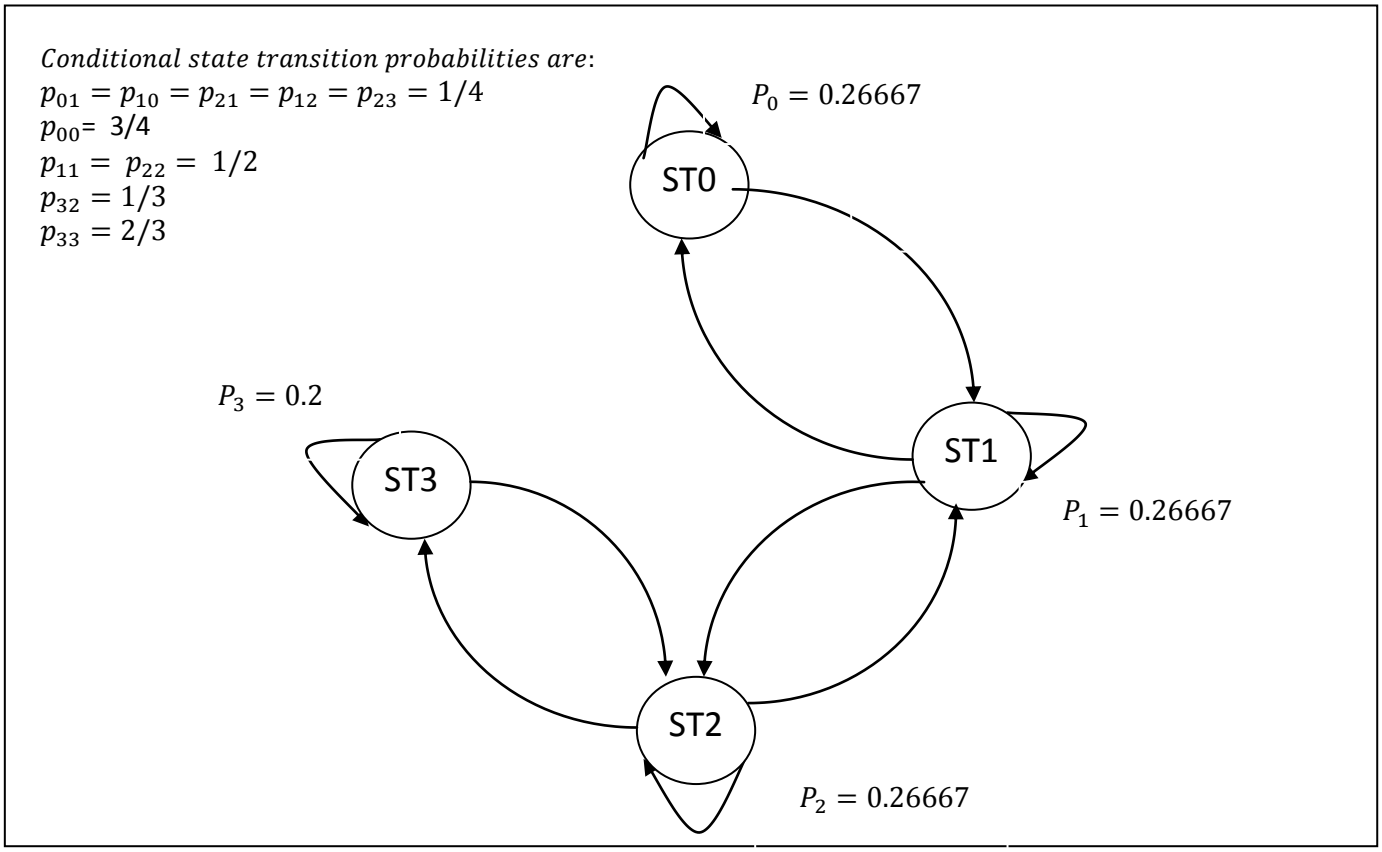

Figure (4) Conditional state transition probabilities p_{ij} and steady state probabilities P_i of example (1)

```
Procedure MOGA ( )
      {
          Input Parameters of GA (benchmark file, population size, tournament
                              size T , number of generations)

          Read_Terms ( benchmark)
          Randomely_Initialize_population( )
          Fitness_terms(pops)
          Fitness_Switching(pops)
          Set_Rank (pops)
          Loop until (Number of Generations = 0)
             {
              Tournament Select (T)
              Crossover (Child )
              Mutation(Child )
              Fitness_terms (Child )
              Fitness_Switching (Child)
              If (Child Fitness!=any existing Fitness)//!= indicates not equal
                        Tournament Replacement (T,Child )
                        Set_Rank( );
              Number of Generations := Number of Generations – 1
             }
          Output Results ( )
       } End MOGA
Set_Rank ( )
     {
          Current_Rank=1;
          All=pop_size;
          Loop For (i =0 to i=pop_size)

             {
              If (NonDominated (i, All))
                      Rank[i] =Current_Rank;
             Remove (i);
              All := All-1;
             Current_Rank := Current_Rank+1;
             }
    }End Set Rank
```

Figure (5) Pseudo Code for the MOGA

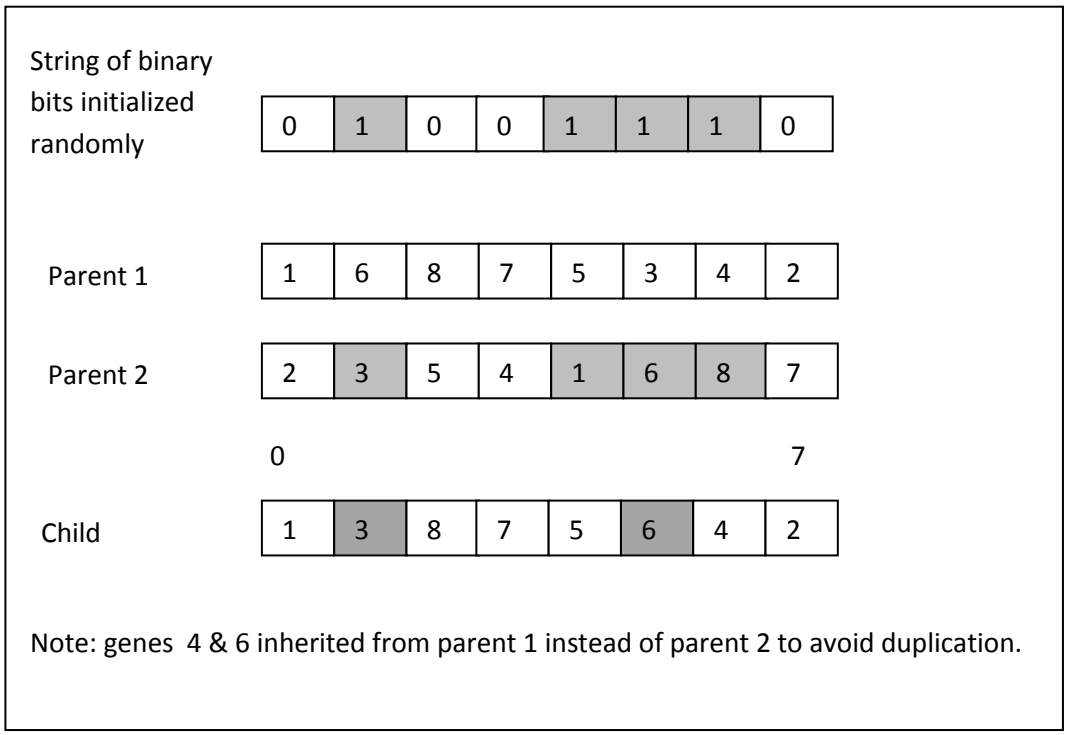

Figure (6) Uniform Crossover

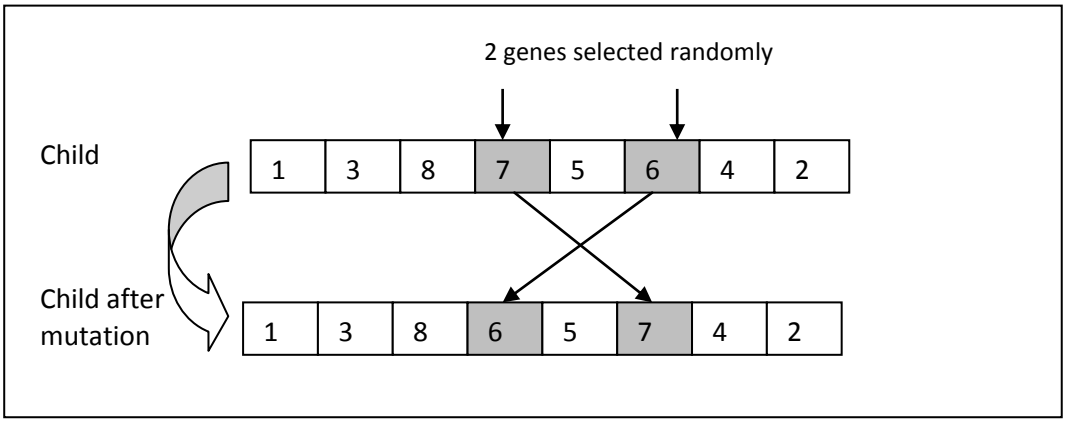

Figure (7) Mutation

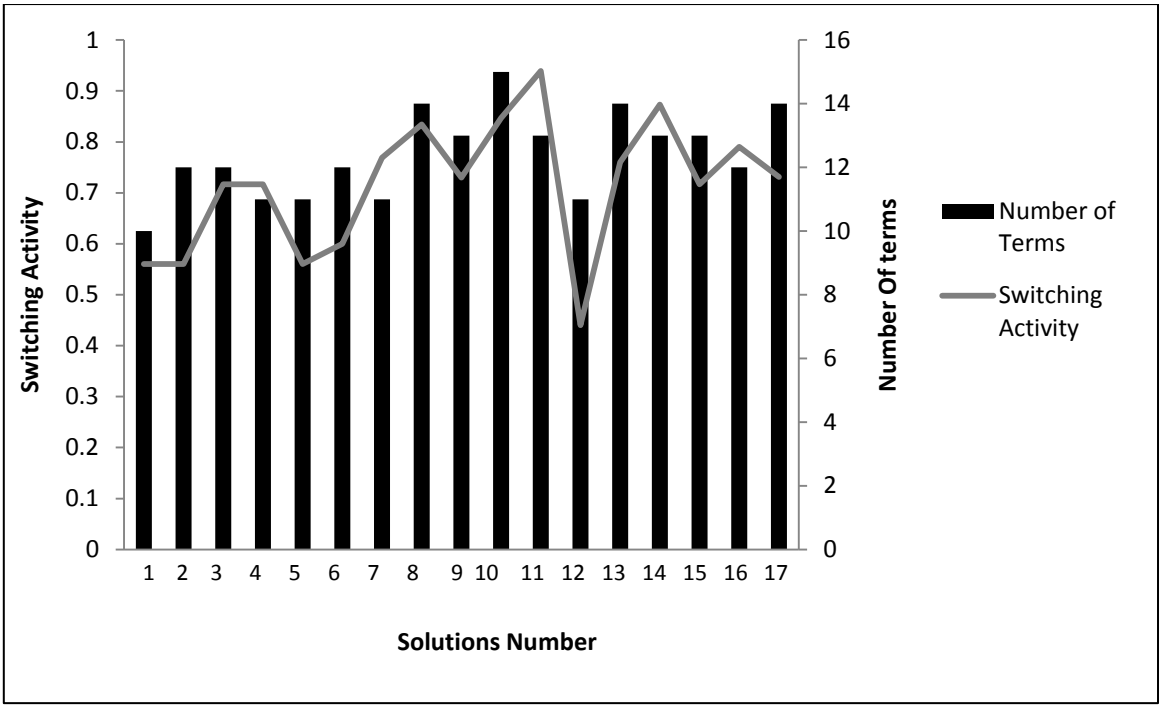

Figure (8) Terms and Switching activities for different assignments of example (2)

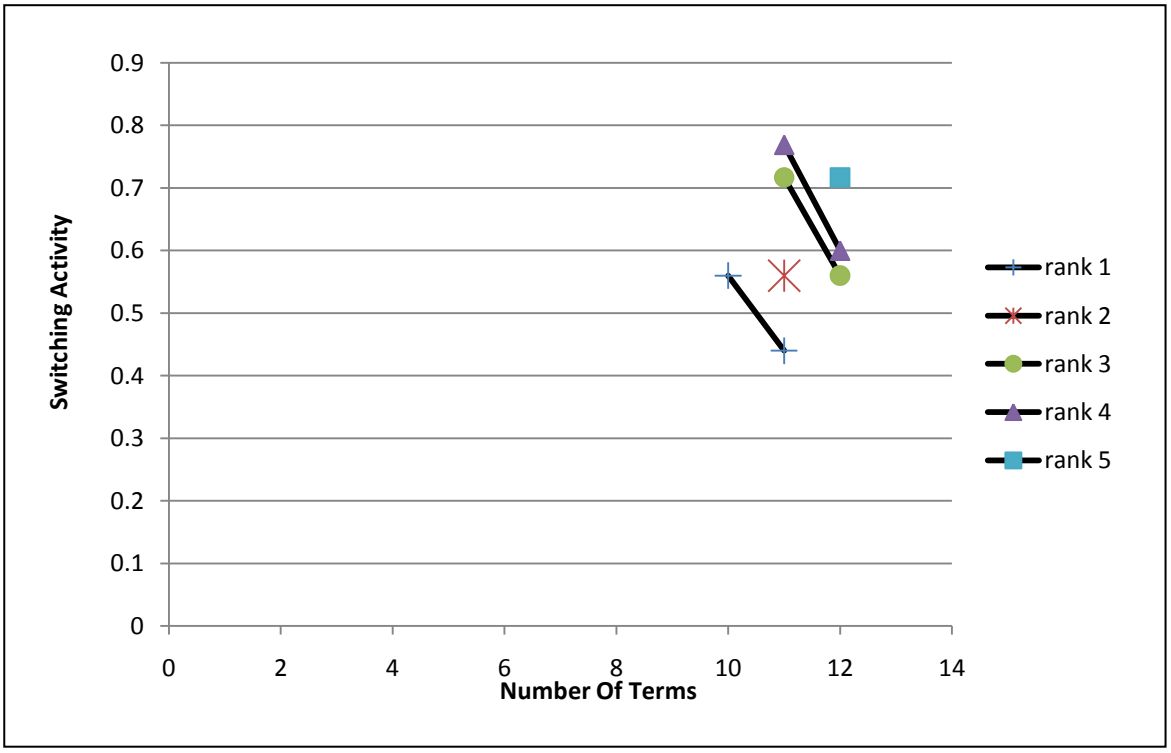

Figure (9) Different ranks for example (2)