Computers and Electrical Engineering 44 (2015) 13–23

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

State assignment for area minimization of sequential circuits based on cuckoo search optimization ☆

Aiman H. El-Maleh [a], Sadiq M. Sait [a,b,*], Abubakar Bala [a]

[a] Computer Engineering Department, KFUPM, Dhahran, Saudi Arabia
[b] Center for Communications and IT Research, Research Institute, KFUPM, Dhahran, Saudi Arabia

ARTICLE INFO

Article history:
Received 5 November 2014
Received in revised form 9 March 2015
Accepted 9 March 2015
Available online 21 April 2015

Keywords:
Cuckoo search
State Assignment
Heuristics
Sequential circuit
Area minimization
Finite state machines

ABSTRACT

A major optimization problem in the synthesis of sequential circuits is State Assignment or State Encoding in Finite State Machines (FSMs). The state assignment of an FSM determines the complexity of its combinational circuit and thus area, delay, testability and power dissipation. Since optimal state assignment is an NP-hard problem and existing deterministic algorithms produce solutions far from best known solutions, we resort to the use of non-deterministic iterative optimization heuristics. This paper proposes the use of cuckoo search optimization (CSO) algorithm for solving the state assignment problem (SAP) of FSMs with the aim of minimizing area of the resulting sequential circuit. Results obtained from the CSO algorithm are compared with those obtained from binary particle swarm optimization (BPSO) algorithm, genetic algorithm (GA), and the well-known deterministic methods of NOVA and JEDI. The results indicate that CSO outperforms deterministic methods as well as other non-deterministic heuristic optimization methods.

1. Introduction

As the density and size of integrated circuits (ICs) keep increasing rapidly, area and power dissipation have become and are still a significant concern in Very Large Scale Integration (VLSI) designs. VLSI systems by nature are mostly sequential circuits and are modeled as finite state machines (FSMs). In FSMs, the behavior of sequential circuits is characterized by using symbolic names to represent states. State assignment is the mapping of the state names of an FSM to a set of binary codes. This mapping has a significant impact on the circuit area and power dissipation [1]. An example of an FSM is given in Table 1, which has 4 states, one input and one output. To understand the example in Table 1, consider the case when *Present State=S0*. If input *X=0*, then *Next State=S0* and *Output=1*, but if *X=1*, then *Next State=S2* and *Output=0*.

Since there are 4 states in the FSM, a 2-bit code is sufficient for encoding each state. Table 2 shows two typical state assignments for the FSM labeled as "Ass. 1" and "Ass. 2". The number of literals of the Boolean equations that implement the FSM as a multi-level circuit with "Ass. 1" is 6 literals while that with "Ass. 2" is 14 literals. The number of literals is a cost measure that correlates with the number of transistors in the circuit and hence its area.

☆ Reviews processed and approved for publication by the Editor-in-Chief.

* Corresponding author at: Computer Engineering Department, KFUPM-#673, Dhahran-#31261, Saudi Arabia.
E-mail addresses: aimane@kfupm.edu.sa (A.H. El-Maleh), sadiq@kfupm.edu.sa (S.M. Sait), g201201620@kfupm.edu.sa (A. Bala).

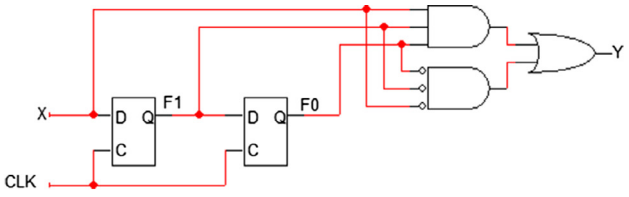
http://dx.doi.org/10.1016/j.compeleceng.2015.03.014
0045-7906/

Table 1
An example of an FSM.

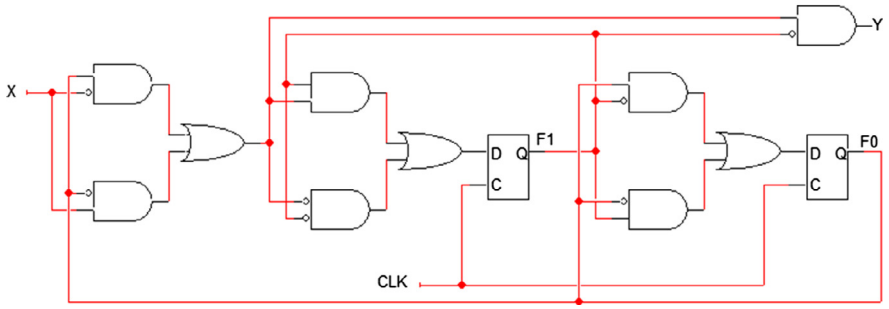
Present state	Next state		Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
S0	S0	S2	1	0
S1	S0	S2	0	0
S2	S1	S3	0	0
S3	S1	S3	0	1

Table 2
State assignment for "Ass. 1" and "Ass. 2" and their resulting literal count.

State	"Ass. 1"	"Ass. 2"
S0	00	01
S1	01	10
S2	10	11
S3	11	00
Area (no. of literals)	6	14



(a) *Circuit resulting from "Ass. 1".*



(b) *Circuit resulting from "Ass. 2".*

Fig. 1. Multi-level circuits synthesized based on "Ass. 1" and "Ass. 2" for the FSM example.

The multi-level circuits resulting from synthesizing the FSM example using "Ass. 1" and "Ass. 2" are shown in Fig. 1. This example demonstrates the significant impact of state assignment on the area of a synthesized sequential circuit.

Formally, the state-assignment problem of an FSM is one that maps state symbols to binary codes using the mapping function $f : S \rightarrow B^n$, where n is the code length, $n \geqslant \lceil \log_2 |S| \rceil$, B^n is an n-dimensional Boolean hypercube and $|S|$ is the number of states. To encode S states, using k bits, the number of possible state-assignment combinations is given in Eq. (1).

$$\frac{(2^k)!}{(2^k - |S|)!} \tag{1}$$

As an illustration, if we have an FSM with 10 states, then each state will require 4 bits for distinctive encoding. As a result, the number of possible state assignments for these 10 states as obtained from Eq. (1) is 29,059,430,400. Hence, exhausting all the

possible combinations in order to find the state assignment that optimizes a certain objective will require a very huge amount of time. Clearly, SAP is computationally hard [2].

Among the best-known methods that were employed for state assignments is that of partitions and decomposition [3]. However, not all state machines have useful closed partitions and can be minimized using these techniques. Implicant merging, code covering and disjunctive coding [4,5] are previous deterministic methods employed for area minimization of two-level combinational circuit implementations of FSMs. These techniques work well for certain FSMs and produce good results but for others they may not apply effectively. This is because these techniques rely mainly on symbolic minimization of state tables and not all FSMs can be minimized symbolically to lead to the best state assignment solutions.

The objective of state assignment targeting multi-level circuit implementation is that of finding state assignments that result in common expressions and maximum literal savings. Devadas et al. suggested two algorithms [6]. One of these algorithms is fan-in oriented and looks for state pairs with higher numbers of incoming transitions from the same states. Subsequently, higher weights are assigned to those pairs of states to be given close codes in terms of Hamming distance. The Hamming distance between the codes of two states is the number of positions of non-identical bits between the codes of the two states. The aim is to maximize the frequency of common cubes in the encoded next-state functions. The second algorithm is fan-out oriented and assigns close codes to state pairs that have similar next-state transitions. A similar deterministic method is JEDI [7], which calculates the encoding affinity cost as a function of how often a pair of states is represented in the next-state and output functions. Wang et al. [8] attempted to solve the problem of state assignment in order to minimize both area and power dissipation for FSMs. They suggested a novel matching-based state assignment algorithm that takes into account area and state transitions simultaneously. The experimental results they obtained show that they outperform NOVA [4] in both area and power.

Due to limitations of existing deterministic algorithms and the intractable nature of the state-assignment problem [9], a lot of work has been done in the area of employing non-deterministic optimization heuristic methods. Some of these methods have been used to solve many combinatorial optimization problems successfully. Examples of these heuristics include simulated annealing, tabu search, particle swarm optimization, cuckoo search and genetic algorithms [2]. Several non-deterministic optimization heuristic methods have been applied to solve SAP. Chaudhury et al. [10] used a genetic algorithm (GA) based state encoding to solve SAP targeting area and power optimization. A unified approach is used targeting static and dynamic power along with area trade-off. Other efforts in the use the GAs to solve SAP include the work by Almaini et al. [11,12], El-Maleh et al. [13], and others [14,15]. Other heuristic optimization techniques for solving SAP include the use of simulated annealing (SA) [16], simulated evolution (SE) [17] and binary particle swarm optimization (BPSO) [18].

This paper proposes the application of a recent heuristic algorithm, the cuckoo search optimization (CSO) algorithm [19,20] integrated with Lévy walk [21], which enables the algorithm to make random walks in the design space for the state-assignment problem (SAP) targeting area minimization. The CSO algorithm has been applied successfully to many computationally difficult problems [22,23]. The main advantage of the CSO algorithm is its simplicity when it comes to implementation as it involves the tuning of few parameters [19].

The rest of the paper is organized as follows. In Section 2 the proposed algorithm is presented. In Section 3 we provide an illustrative example of running the CSO algorithm on a typical benchmark circuit. Subsequently, in Section 4 the experimental results are presented. Finally, Section 5 concludes the paper.

2. CSO algorithm

Yang et al. [19] developed a new meta-heuristic optimization algorithm called cuckoo search (CS). Cuckoos are fascinating birds due to their reproduction strategy. Cuckoos lay their eggs in communal nests and may remove others' eggs to increase the hatching probability of their own eggs. They are often very specialized in laying eggs that mimic the color and pattern of that of their hosts. This reduces the probability that their eggs will be identified by the host and thus get discarded. As a result, this increases their reproductivity. They often choose a nest where the host bird has just laid its own eggs. In general, the cuckoo's eggs hatch slightly earlier than their hosts' eggs. Once the first cuckoo chick is hatched, it blindly propels other eggs out of the nest in order to increase the share of food it gets from the host bird.

This section briefly discuses the modified cuckoo search optimization (CSO) algorithm [20] for solving the state-assignment problem (SAP) of sequential circuits.[1] The modification of the cuckoo search algorithm in [20] resulted from the inadequacy of the original cuckoo search algorithm developed by Yang et al. [19] to have a faster convergence rate. This modified version presented two improvements in order to make the cuckoo search have a wider application but at the same time not losing the attractive features of the original method.

[1] The terms egg and nest are used interchangeably because each of the nests contains a single solution termed egg.

Algorithm 1. CSO algorithm

```
1: P_a ← 0.75, ψ ← 1.62, MAXiter ← 350
2 : Initialize the Population
3 : Rank the entire population according to cost
4 : for G = 1 to MAXiter do
5 :   partition the population into top and bottom nests
6:   for all X_i in bottom nests do
7 :     use Lévy flight to create a new nest X_k from X_i
8:     replace X_i with X_k
9:   end for
10 :   Rank the entire population again according to cost
11 :   for all X_i such that X_i is in top nests do
12 :     Select another random top nest X_j
13:       if (X_i = X_j) then
14 :         perform Lévy flight from X_i to create a new nest X_k
15 :         Select a random nest X_l
16:         if (Cost(X_k) < Cost(X_l)) then
17 :           replace X_l with X_k
18:         end if
19:     else
20 :       Move a distance d_x = |X_i − X_j|/ψ from X_i to create a new nest X_k
21 :       validate X_k
22 :       Select a random nest X_l
23:       if (Cost(X_k) < Cost(X_l)) then
24 :         replace X_l with X_k
25:       end if
26:     end if
27:   end for
28 :  Rank the nests according to their costs
29: end for
30 : Save the best achieved state assignment and its cost
```

For solving the SA problem, the CSO algorithm, shown in Algorithm 1, starts by setting some essential variables such as the size of the population, percentage of nests to be abandoned P_a, Maximum iteration *MAXiter*, golden ratio (ψ) (a constant number approximately = 1.62 [24]). The population is then initialized with random state assignment solutions. The entire generated population is then ranked according to the cost of each nest. The cost used is the literal count which is obtained by the sequential interactive synthesis (SIS) tool [25]. Low literal count implies low cost and thus less area for the resulting sequential circuit. After the population is ranked, a procedure is repeated for a number of times up to the *MAXiter* value.

This procedure commences by partitioning the population into top and bottom nests. Then for each of the bottom nests (nests to be abandoned), we generate a Lévy flight from the particular nest and generate a new nest to replace the existing nest. The value of the Lévy flight function determines the number of code-pair swaps to be made. A code-pair swap picks randomly two of the available codes and swaps their positions. For example, if code 000 is assigned to state S1 and code 111 is assigned to state S2, and the two codes 000 and 111 are picked to swap, then the code for S1 becomes 111 and the code of S2 becomes 000. Similarly, if code 000 is assigned to state S1 and code 111 is unassigned and the two codes are swapped, the code for S1 becomes 111 and the code 000 becomes unassigned. In essence, all the existing bottom nests are replaced by fresh nests by performing Lévy flight from each of the bottom nests.

In contrast, for each of the top nests, X_i, we select another random nest, X_j, from the top nests. If the second randomly chosen nest X_j is the same as the present top nest, then Lévy flight is conducted from X_i to generate a new nest X_k. Furthermore, another nest, X_l, is then selected randomly from the entire population. If the cost of X_k is less than that of X_l, then the nest X_l is replaced with X_k.

On the other hand, if the second nest selected from the top nests, X_j, is not the same as the first selected nest, X_i, then we determine d_x (d_x is found by taking the Hamming distance between X_i and X_j and dividing the result by the golden ratio which is approximately = 1.62 [24]). Then, a new nest X_k is generated from X_i by replacing d_x bits from X_i with their corresponding bits from X_j. The bits replaced in X_i are the first d_x non-identical bits between X_i and X_j. The new generated solution, X_k, needs to be validated to avoid the assignment of the same code to two or more states. If any duplicate code is found then it is replaced with one of the unassigned codes. Minimum Hamming distance criterion is used to pick a code from a set of unassigned codes to replace a duplicate code. Subsequently, we select a random nest, X_l, from the entire population and compare the cost of X_l with X_k. If X_k has a lower cost than X_l, then it replaces it.

Finally, the nests are ranked according to their cost. The algorithm then repeats the larger loop until the maximum iteration number is reached. Since the entire bottom nests are replaced at the beginning of each iteration via Lévy flights, the average cost is bound to rise and fall rather than improve steadily. This is similar to hill climbing in elitist algorithms where the best solution is always retained while non-determinism (via Lèvy flights) guides the introduction of other solutions.

The CSO algorithm has the advantage that it has few parameters to tune, namely the percentage of nests to be abandoned (P_a), the golden ratio (ψ) and the function used to compute the Lévy flight. Suggestions for empirical values for the golden ratio (ψ) and the function used to compute the Lévy flight are available in the literature [19,20]. In essence, CSO has only P_a to tune. This is in contrast to other evolutionary algorithms that involve more parameters to tune such as the genetic algorithm which has at least four parameters to tune [2].

2.1. Lévy flight

Numerous findings have revealed that characteristics of Lévy flight are being demonstrated by the flight behavior of many insects and animals. Contemporary research by Reynolds and Frye have confirmed that fruit flies explore their landscape using a series of straight flight paths disrupted by a sudden 90° turn, thus leading to a Lévy-flight-style irregular scale free search pattern. Recent application of such behavior in optimization and optimal search have produced good results [26,27]. The random walk around the design space is essentially provided by the Lévy flight with the random step length drawn from a Lévy distribution. Fig. 2 shows a typical plot of the Lévy flight. In our work, the Lévy flight is performed by swapping a number of code pairs according to the random step length generated.

The number of code pairs to swap, k, called, the Lévy flight step length, is computed by the following equation [28]:

$$k = (1-u)^{-1/\alpha} \tag{2}$$

where u is a uniform random variable in the range [0,1] and $\alpha = i^{1/5}$, where i is the iteration number.

3. Illustrative example

In this section, the CSO algorithm is demonstrated with an illustrative example. The example chosen is the dk14 circuit, which is one of the MCNC/LGSynth [29] benchmark circuits. The dk14 circuit has 7 states, 3 inputs and 5 outputs. We optimize this circuit using the proposed CSO algorithm and show how the cost (literal count) converges to the near optimal with iterations. Since the circuit has 7 sates, we need a minimum of 3 bits to encode each of the states uniquely.

In this illustrative example, we set the population size to 10, which is generated initially randomly, and the percentage of nests to abandon as 70%. The population is then ranked according to cost with the nest with minimum cost at the top, as shown in Fig. 3.

Iteration 1: We begin the first iteration by dividing the population into top and bottom nests as shown by the shaded and unshaded rows in Fig. 3, respectively. Then, perturbations are performed on the bottom part of the population by using Lévy flights. These newly generated nests are made to replace the old ones. For example, for the first bottom nest, $N4$, the Lévy flight step length generated is 1. So $N4$ changes from {011,110,100,111,010, 101,000} to {011,100,110,111,010,101,000}, which has a cost of 151. It can be seen that the codes of S2 and S3 have swapped positions. Similarly, Lévy flight step length is generated for each of the other nests in the bottom nests, and perturbations are carried out accordingly. Consequently, all of the bottom nests are replaced with the newly generated ones. The resulting new generated bottom nests are as shown in Fig. 4.

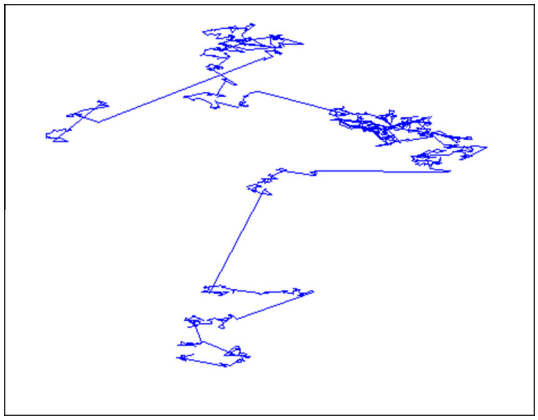


Fig. 2. Typical plot of Lévy flight.

	S1	S2	S3	S4	S5	S6	S7	cost
N1	111	010	101	000	011	001	110	115
N2	011	111	010	101	110	000	001	117
N3	010	011	111	101	110	001	000	120
N4	011	110	100	111	010	101	000	125
N5	110	010	101	011	001	000	100	126
N6	001	011	010	000	111	101	110	130
N7	011	010	110	100	111	001	000	133
N8	001	011	111	010	100	110	101	136
N9	010	000	111	110	011	101	001	140
N10	000	110	111	100	011	101	010	147

Fig. 3. Initial population after ranking.

	S1	S2	S3	S4	S5	S6	S7	cost
N4	011	100	110	111	010	101	000	151
N5	111	101	010	001	100	110	011	132
N6	001	011	110	000	111	101	010	143
N7	101	010	110	100	111	001	000	146
N8	000	011	111	010	101	110	100	149
N9	101	000	111	110	011	010	001	130
N10	101	001	111	100	011	000	010	114

Fig. 4. Newly generated bottom nests.

	S1	S2	S3	S4	S5	S6	S7	cost
N1	101	001	111	100	011	000	010	114
N2	111	010	101	000	011	001	110	115
N3	011	111	010	101	110	000	001	117
N4	010	011	111	101	110	001	000	120
N5	101	000	111	110	011	010	001	130
N6	111	101	010	001	100	110	011	132
N7	001	011	110	000	111	101	010	143
N8	101	010	110	100	111	001	000	146
N9	000	011	111	010	101	110	100	149
N10	011	100	110	111	010	101	000	151

Fig. 5. Ranked population after merging newly generated bottom nests.

Next, the newly generated bottom nests are merged with the top nests, and the entire population is ranked again as shown in Fig. 5. Then we proceed to top nests perturbation. For the first top nest N1: {101,001,111,100,011,000,010}, we pick another random top nest. In this case the second random top nest selected is N2: {111,010,101,000,011, 001,110}. Since the second nest picked is not the same as the first top nest, we compute the Hamming distance between the two nests (equal to 7) divided by the golden ratio (1.62) [24] to generate the perturbation number 4. Then, we create a new nest, X_k, by replacing 4 bits from N1 with their corresponding 4 bits from N2. The replaced bits in N1 are the first non-identical 4 bits between N1 and N2. The resulting nest X_k is {111,010,101,100,011,000,110} with a cost of 127. Next we select a random nest from the entire population X_l which is N7: {001,011,110,000,111,101,010} whose cost is 143. Since the cost of X_k is lower than that of X_l, X_l is replaced with X_k.

A similar procedure is carried out for the second and third top nests. Finally, the population is ranked and the best nest and its resulting cost are reported. The best nest in this case is {101,001,111,100,011,000,010}, and its cost is 114. The ranked population after iteration 1 is shown in Fig. 6.

Iteration 2: A similar procedure is repeated in iteration 2 and the final population is as shown in Fig. 7 with the best nest being {101,001,111,100,011,000, 010} with a cost of 114.

	S1	S2	S3	S4	S5	S6	S7	cost
N1	101	001	111	100	011	000	010	114
N2	111	010	101	000	011	001	110	115
N3	011	111	010	101	110	000	001	117
N4	010	011	111	101	110	001	000	120
N5	111	010	101	100	011	000	110	127
N6	011	111	010	101	000	001	110	127
N7	101	000	111	110	011	010	001	130
N8	111	101	010	001	100	110	011	132
N9	000	011	111	010	101	110	100	149
N10	011	100	110	111	010	101	000	151

Fig. 6. Ranked population after iteration 1.

	S1	S2	S3	S4	S5	S6	S7	cost
N1	101	001	111	100	011	000	010	114
N2	111	010	101	000	011	001	110	115
N3	111	010	101	100	011	000	001	116
N4	011	111	010	101	110	000	001	117
N5	111	101	010	001	000	100	011	121
N6	101	001	111	011	000	110	010	123
N7	100	000	111	110	011	010	001	129
N8	001	111	010	101	000	011	110	130
N9	010	001	111	101	110	000	011	140
N10	011	100	010	111	110	101	000	151

Fig. 7. Ranked population after iteration 2.

	S1	S2	S3	S4	S5	S6	S7	cost
N1	010	011	110	111	001	000	100	101
N2	010	011	110	111	001	000	100	101
N3	010	011	110	111	001	000	100	101
N4	010	011	110	111	001	000	100	101
N5	010	011	110	111	001	101	100	106
N6	110	011	010	111	001	101	100	121
N7	111	001	100	101	010	011	000	129
N8	010	011	111	110	001	101	100	136
N9	010	000	111	110	100	011	101	136
N10	010	101	011	110	001	111	100	143

Fig. 8. Ranked population after iteration 20.

Iteration 20: The same procedure is repeated and at the end of iteration 20 the final ranked population obtained is as shown in Fig. 8. In this case the best nest seen so far after the 20th iteration is {010,011,110,111,001,000,100} with a resulting cost of 101. It can be seen that the algorithm has begun to converge.

4. Experimental results

The MCNC/LGSynth [29] benchmark circuits were employed to test the efficiency and competitiveness of the proposed CSO algorithm as compared with other existing algorithms in the literature. The characteristics of these benchmark circuits are listed in Table 3. For each benchmark circuit, the number of states, number of inputs and number of outputs are shown.

The *SIS 1.3* package [25] is used to compute the literal count for each of the benchmark circuits. The *stg_to_network -e 2* command is used for single-output two-level circuit optimization and then the fx (fast extraction) command is used for multi-level circuit optimization. The area cost is the number of literals of the synthesized multi-level circuit. We have used

Table 3
Benchmark circuits.

Circuit	States	Inputs	Outputs
bbara	10	4	2
bbsse	16	7	7
cse	16	7	7
planet	48	7	19
dk14	7	3	5
ex2	19	2	2
ex3	10	2	2
keyb	19	7	2
lion9	9	2	1
pma	24	8	8
s1	20	8	6
s1494	48	8	19
s832	25	18	19
sand	32	11	9
styr	30	9	10
tbk	32	6	3
train11	11	2	1

Table 4
Literal count comparison of CSO with other state assignment techniques.

Circuit	CSO				BPSO[18]				GA [13]				Jedi [8]	Nova [4]
	Best	Avg.	Worst	Time (s)	Best	Avg.	Worst	Time (s)	Best	Avg.	Worst	Time (s)		
bbara	49	50.8	53	1704	49	52.5	55	793	49	49.4	58	1204	73	57
bbsse	97	101.6	106	1992	102	107.1	111	823	99	101.6	107	1351	134	140
cse	182	183.5	185	2205	184	191	198	987	179	184.2	190	1744	240	214
dk14	98	98.3	99	1909	98	99.8	102	784	102	103.1	105	1543	108	111
ex2	67	87.7	104	1976	66	99.5	117	861	64	90.1	120	1374	123	127
ex3	49	52.6	54	1824	51	54	56	796	54	54.7	58	1149	65	71
keyb	145	153.7	158	2219	143	163.8	178	1480	142	152.4	165	2096	260	201
lion9	10	10.9	11	1798	10	11.7	13	786	10	10	10	1084	19	27
planet	475	490.2	504	3207	494	526.1	561	2288	462	501.6	562	3775	603	591
pma	149	159.4	167	2374	155	164.6	181	1163	160	165.3	180	1677	263	241
s1	171	203.1	244	2376	173	231.7	278	1592	131	215.5	310	2271	282	340
s1494	527	561.2	589	4481	588	602	628	3073	560	589.6	623	3885	679	715
s832	216	233.2	242	2684	216	245.4	267	1751	230	256.7	280	2255	357	274
sand	476	495.6	521	3282	488	510.1	527	2374	498	519.8	557	3003	554	558
styr	417	438.9	466	3444	412	437.5	455	2326	405	422.5	520	3506	518	502
tbk	303	322.5	350	9090	261	368.2	458	6671	343	376.3	429	8061	305	365
train11	12	13.6	15	1849	12	13.7	15	773	18	18.5	20	1102	34	32
Total	3443	3656.8	3868	48414	3502	3878.7	4200	29321	3506	3811.3	4294	41080	4617	4566

a population size of 64 and a number of iterations equal to 350 for all the compared techniques. We have adopted the use of the value of 75% for the percentage of nests to abandon (P_a) and the value of the golden ratio (ψ) equal to 1.62 based on the findings in [19,20]. We have also validated their suitability for the state assignment problem based on experimental analysis.

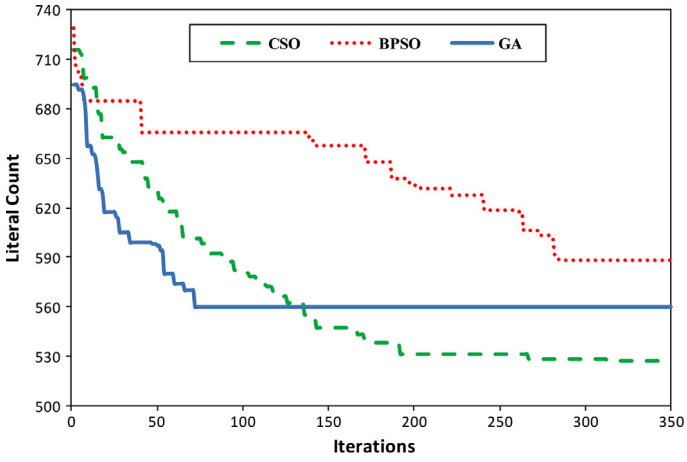
Table 4 presents the results for the proposed CSO algorithm along with the results for BPSO [18], GA [13], and the deterministic techniques NOVA [4] and Jedi [8]. Each of the non-deterministic optimization algorithms is run 10 times for each benchmark circuit and the best, average and worst costs are recorded.

The CPU time, in seconds, taken by each technique for one run is also reported in the table. Experiments were run on Linux machine with Quad-core processors and 4 GB of RAM.

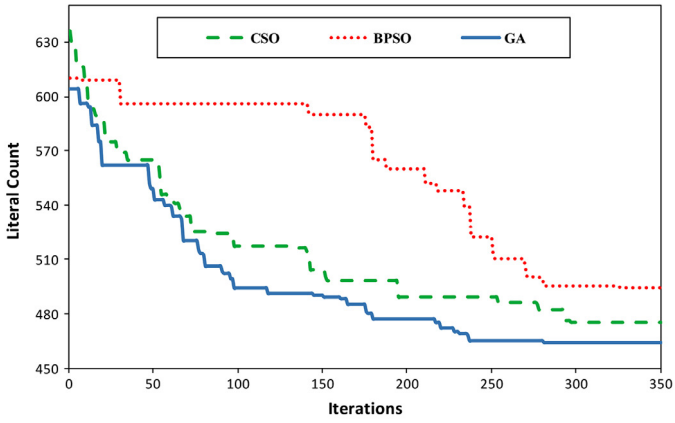
Initially it can be seen from Table 4 that non-deterministic heuristic optimization algorithms i.e., CSO, BPSO [18] and GA [13] outperform deterministic methods of Nova [4] and Jedi [8] in terms of literals. Even the worst obtained solutions have overall less literal count than those obtained by the deterministic techniques.

The CSO algorithm performs better than BPSO [18] and GA [13] in terms of overall best, worst and average literal counts. This is evident from the last row of Table 4. Based on the best results achieved, our proposed CSO state assignment algorithm achieved better results than BPSO [18] for 8 benchmark circuits and equal results in 5. However, CSO achieved better results than GA [13] for 9 benchmark circuits and equal results in 2.

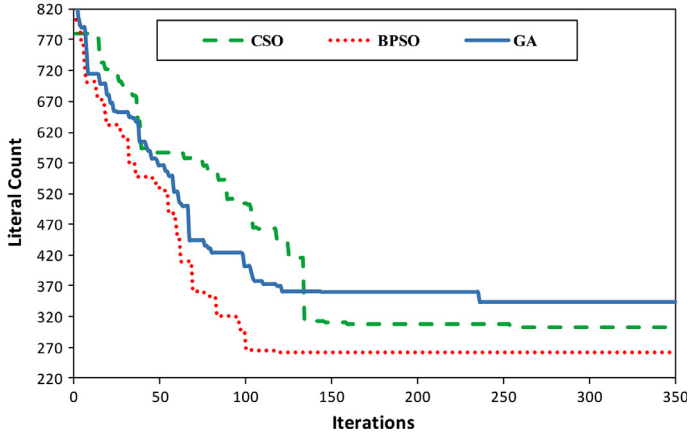
CSO achieved better results than other compared algorithms as it mimics the natural behavior of cuckoo birds to increase productivity. The use of Lévy flights, as seen in birds and bees, helps traverse the search space both locally, and with sudden jumps to distant solutions thereby combining the local search with diversification. This is in addition to the way top nests are perturbed by combining good features from other top nests.



(a) *s1494*



(b) *planet*



(c) *tbk*

Fig. 9. Literal count versus iterations of best results for the three benchmark circuits, s1494, planet and tbk.

The CPU time of CSO is higher than BPSO [18] and GA [13]. This is because the current implementation of CSO performs a sorting step twice in every iteration. This results in increasing the CPU time. The sorting step can be replaced by a more efficient implementation that separates the population into top and bottom nests based on computing the average and standard deviation cost of the entire population. Then, the bottom nests can be marked as those whose cost fall below the average cost by a certain amount determined by the standard deviation of the population cost.

Table 5
Effect of population size on cost of obtained solutions by CSO.

Circuit	Population	Best	Avg.	Worst
s1494	32	563	580.8	598
	64	527	561.2	589
	128	513	537.2	573
planet	32	464	513.5	536
	64	475	490.2	504
	128	459	492.5	516
tbk	32	305	347.3	474
	64	303	322.5	350
	128	301	333.0	464

Table 6
Effect of the percentage of nests to abandon (Pa) on cost of obtained solutions by CSO.

Circuit	P_a	Best	Avg.	Worst
s1494	0.5	539	564.0	607
	0.75	527	561.2	589
	0.85	529	567.1	586
planet	0.5	477	494.4	530
	0.75	475	490.2	504
	0.85	483	502.6	529
tbk	0.5	300	352.0	460
	0.75	303	322.5	350
	0.85	290	356.0	425

Table 7
Product term comparison of CSO with other state assignment techniques.

Circuit	CSO	Nova [4]	GA [12]	SA [16]
bbara	21	24	22	22
bbsse	26	29	28	27
cse	41	45	43	43
keyb	44	48	46	46
planet	80	86	81	81
s1	48	80	43	43
sand	86	89	94	92
styr	79	94	78	78
train11	6	9	10	10

Fig. 9 shows a plot of literal count (area) against iterations for the CSO, BPSO [18] and GA [13] algorithms for three of the largest benchmark circuits: s1494, planet and tbk. Behavior of iterative search heuristics is best understood and illustrated using large test cases. CSO achieves a better literal count than BPSO [18] and GA [13] in two out of the three compared circuits. CSO has a faster convergence rate than BPSO [18] in two out of the three compared circuits. However, although the convergence rate of GA [13] is slightly better, CSO achieves overall better results.

In order to show the effect of population size on the cost of obtained solutions by the CSO algorithm, we have run experiments using population sizes of 32, 64 and 128 for the benchmark circuits: s1494, planet and tbk. Table 5 shows the best, average and worst results obtained based on 10 runs. Considering the average cost achieved, it can be seen that the results improved when the population size was increased from 32 to 64. However, the results improved only for one circuit when the population size was increased from 64 to 128.

In order to show the effect of the percentage of nests to be abandoned (P_a) on the cost of obtained solutions by the CSO algorithm, we have run experiments using P_a values of 0.5, 0.75 and 0.85 for the benchmark circuits: s1494, planet and tbk based on a population size of 64. Table 6 shows the best, average and worst results obtained based on 10 runs. Considering the results achived, it can be seen that the impact of P_a is not significant on the obtained results. However, considering the average cost, using $P_a = 0.75$ achieved better results.

A number of state assignment techniques existing in the literature target the implementation of sequential circuits using two-level circuits instead of multi-level circuits. The area optimization criteria in this case is the number of product terms instead of the number of literals. We have run the CSO algorithm targeting the optimization of the number of product terms and compared the results with two recent implementations based on Genetic Algorithm [12] and Simulated Annealing (SA) [16]. Table 7 shows the number of product terms obtained by the proposed CSO algorithm, Nova [4], GA [12] and SA [16].

CSO achieved better results than Nova [4] for all the compared circuits. In addition, it achieved better results than GA [12] and SA [16] in 7 out of the 9 compared circuits.

5. Conclusion

This paper presented the application of a recent optimization method, the cuckoo search optimization algorithm, to solve the state assignment problem in FSM. Experimental results on MCNC/LGSynth benchmark circuits show that CSO achieved better results than other non-deterministic heuristic optimization methods such as Genetic algorithm (GA) and binary particle swarm optimization (BPSO), and the well-known deterministic methods of NOVA and JEDI. CSO achieved better results than other compared algorithms due to the built-in diversification of search via Levy flights. In addition to this it has fewer parameters and therefore easy to tune.

Acknowledgment

Authors acknowledge King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia, for all support.

References

[1] Xia Y, Almaini AEA. Genetic algorithm based state assignment for power and area optimisation. In: IEE proceedings computers and digital techniques; 2002, vol. 149, p. 128–33.
[2] Sait SM, Youssef H. Iterative computer algorithms with applications in engineering: solving combinatorial optimization problems. IEEE Computer Society Press; 1999.
[3] Eschermann B. State assignment for hardwired VLSI control units. ACM Comput Surv (CSUR) 1993;25(4):415–36.
[4] Villa T, Sangiovanni-Vincentelli A. Nova: state assignment of finite state machines for optimal two-level logic implementations. In: 26th ACM/IEEE design automation conference; 1989. p. 327–32.
[5] Ashar P, Devadas S, Newton AR. Sequential logic synthesis. Kluwer Academic Publishers; 1992.
[6] Devadas S, Ma H-K, Newton AR, Sangiovanni-Vincentelli A. Mustang: state assignment of finite state machines targeting multilevel logic implementations. IEEE Trans Comput-Aided Des Integr Circ Syst 1988;7(12):1290–300.
[7] Lin B, Newton AR. Synthesis of multiple level logic from symbolic high-level description languages. In: Proceedings of the international conference on VLSI; 1989, vol. 187, p. 196.
[8] Wang KH, Wang WS, Hwang T, Wu ACH, Lin YL. State assignment for power and area minimization. In: Proceedings of IEEE international conference on computer design, ICCD'94.; 1994. p. 250–4.
[9] Micheli GD. Synthesis and optimization of digital circuits. McGraw Hill; 1994.
[10] Chaudhury S, Sistla KT, Chattopadhyay S. Genetic algorithm-based FSM synthesis with area-power trade-offs. Integr, VLSI J 2009;42(3):376–84.
[11] Ali B, Almaini AEA, Kalganova T. Evolutionary algorithms and theirs use in the design of sequential logic circuits. Genet Program Evol Mach 2004;5(1):11–29.
[12] Al Jassani AE, Urquhart N, Almaini AEA. State assignment for sequential circuits using multi-objective genetic algorithm. IET Comput Digit Tech 2011;5:296–305.
[13] El-Maleh A, Sait SM, Nawaz Khan F. Finite state machine state assignment for area and power minimization. In: Proceedings IEEE international symposium on circuits and systems, ISCAS 2006; 2006. p. 5303–6.
[14] Chyzy M, Kosinski W. Evolutionary algorithm for state assignment of finite state machines. In: Proceedings Euromicro symposium on digital system design; 2002. p. 359–62.
[15] Amaral JN, Tumer K, Ghosh J. Designing genetic algorithms for the state assignment problem. IEEE Trans Syst, Man Cybern 1995;25(4):687–94.
[16] Yang M. State assignment for finite state machine synthesis. J Comput 2013;8(6):1406–10.
[17] Sait SM, Arafeh A, Oughali F. FSM state-encoding for area and power minimization using simulated evolution algorithm. J Appl Res Technol 2012;10(6):845–58.
[18] El-Maleh AH, Sheikh AT, Sait SM. Binary particle swarm optimization (BPSO) based state assignment for area minimization of sequential circuits. Appl Soft Comput 2013;13(12):4832–40.
[19] Yang X-S, Deb S. Engineering optimisation by cuckoo search. Int J Math Model Numer Optim 2010;1(4):330–43.
[20] Walton S, Hassan O, Morgan K, Brown M. Modified cuckoo search: a new gradient free optimisation algorithm. Chaos, Solit Fract 2011;44(9):710–8.
[21] Rhee I, Shin M, Hong S, Lee K, Kim SJ, Chong S. On the Lévy-walk nature of human mobility. IEEE/ACM Trans Network (TON) 2011;19(3):630–43.
[22] Gandomi AH, Yang X-S, Alavi AH. Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng Comput 2013;29(1):17–35.
[23] Layeb A. A novel quantum inspired cuckoo search for knapsack problems. Int J Bio-Inspired Comput 2011;3(5):297–305.
[24] Dunlap RA. The golden ratio and Fibonacci numbers. World Scientific; 1997.
[25] Sentovich EM, Singh KJ, Lavagno L, Moon C, Murgai R, Saldanha A, et al. SIS: a system for sequential circuit synthesis; 1992. <http://www.eecs.berkeley.edu/Pubs/TechRpts/1992/2010.html >.
[26] Pavlyukevich I. Cooling down Lévy flights. J Phys A: Math Theoret 2007;40(41):12299.
[27] Shlesinger MF. Mathematical physics: search research. Nature 2006;443(7109):281–2.
[28] Brown CT, Liebovitch LS, Glendon R. Lévy flights in dobe Ju/hoansi foraging patterns. Human Ecol 2007;35(1):129–38.
[29] http://www.cbl.ncsu.edu:16080/benchmarks/LGSynth89/fsmexamples.

Aiman El-Maleh is an Associate Professor in the Computer Engineering Department at King Fahd University of Petroleum & Minerals (KFUPM). He holds a PhD in Electrical Engineering, with dean's honor list, from McGill University, Canada, in 1995. His research interests are in the areas of synthesis, testing, and verification of digital systems.

Sadiq M. Sait is a professor in the Department of Computer Engineering at KFUPM. Sait has authored over 200 research papers, contributed chapters to technical books, and lectured in over 25 countries. He is the principle author of two books, and now the Director of the Center for Communications and IT Research at the Research Institute of KFUPM.

Abubakar Bala is a Graduate Assistant with Bayero University Kano, Nigeria. He holds a masters degree in computer engineering from King Fahd University of petroleum & Minerals, in 2015. Before then he had obtained a bachelor degree in computer engineering from Bayero University Kano in 2010. His research interests include: optimization, cloud computing and renewable energy.