Digital Circuit Design Through Simulated Evolution (SimE)

Sadiq M. Sait, Mostafa Abd-El-Barr, Uthman S. Al-Saiari, Bambang A.B. Sarif
Computer Engineering Department
King Fahd University of Petroleum & Minerals
KFUPM Box 673, Dhahran-31261, Saudi Arabia
{sadiq, mostafa, saiarios, sarif}@ccse.kfupm.edu.sa

Abstract- In this paper, the use of Simulated Evolution (SimE) Algorithm in the design of digital logic circuits is proposed. SimE algorithm consists of three steps: evaluation, selection and allocation. Two *goodness* measures are designed to guide the selection and allocation operations of SimE. Area, power and delay are considered in the optimization of circuits. Results obtained by SimE algorithm are compared to those obtained by Genetic Algorithm (GA).

1 Introduction

Design is usually considered to be an activity requiring considerable human creativity and knowledge. Even the definition of the term *design* itself is quite elusive, since it can be interpreted in several different ways depending on the task to be performed [10].

The definition of design that fulfills the objectives of this paper is the process of deriving, from a specified input/output behavior, a structure (in our case a certain combination of logic gates) that is functional (produces all the outputs desired for all the inputs specified) within a certain set of specified constraints. Furthermore, we want the obtained design to be optimum in terms of certain structural features (e.g., area, power and delay). The design process is a very tedious and error prone task that usually requires considerable human expertise and involves trade-offs.

Advances in logic design and synthesis is due to the automation of the design process using sophisticated computer-aided design tools. These tools have vastly improved design turn around time to keep pace with the increasing demand and complexity of circuits. However, there have been many attempts at developing programs for automated design, and such programs are difficult to build.

Several techniques in evolutionary design of digital circuits have been studied. Most of the work done in evolutionary logic synthesis is random search where the evolutionary algorithm will blindly evolve the circuit according to a given set of objectives without using rules and techniques of the conventional logic synthesis. It is believed that incorporating logic synthesis rules and guidelines combined with the idea of assemble-and-test could lead to better results.

The objective is to develop a computer-based tool that can make the design process less tedious for the human designer without sacrificing quality of the design produced. In this paper, we limit our focus to combinational logic circuits, which contains no memory elements and no feedback paths. The use of Simulated Evolution algorithm as a search heuristic will be considered [9].

This paper is organized as follows: first, Simulated Evolution Algorithm is briefly presented. Then, a formulation of the problem of interest is described. Following this, an introduction to the approach being used to implement SimE for digital logic design is presented. Finally, SimE algorithm is compared with genetic algorithm (GA) in circuit design with optimization for area, power and delay.

2 Simulated Evolution Algorithm (SimE)

Combinatorial optimization problems seek to find a global optimum of some real valued cost functions $cost : \Omega \rightarrow \mathrm{R}$ defined over a discrete set Ω. The Set Ω is called the state space and its elements are referred to as states. A state space Ω together with an underlying neighborhood structure (the way one state can be reached from another state) form the solution space [1].

The Simulated Evolution (SimE) algorithm is a general search strategy for solving a variety of combinatorial optimization problems [8, 9]. The SimE algorithm starts from an initial assignment, and then, following an evolution-based approach, it seeks to reach better assignments from one generation to the next. SimE assumes that there exists a population P of a set M of k elements. In addition, there is a cost function $Cost$ that is used to associate with each assignment of an element m a cost C_m. The cost C_m is used to compute the goodness (fitness) g_m of element m, for each $m \in M$. Furthermore, there are usually additional constraints that must be satisfied by the population as a whole or by particular elements. A general outline of the SimE algorithm is given in Figure 1.

SimE algorithm proceeds as follows. Initially, a population[1] is created at random from all populations satisfying the environmental constraints of the problem. The algorithm has one main loop consisting of three basic steps, *Evaluation*, *Selection*, and *Allocation*. The three steps are executed in sequence until the population average *goodness* reaches a

[1] In SimE terminology, a population refers to a single solution. Individuals of the population are components of the solution; they are the movable elements.

ALGORITHM *Simulated_Evolution(E, ,L, Stopping-Criteria)*;
$INITIALIZATION$;
Repeat
$EVALUATION$:
ForEach $m \in M$ **Do** $g_m = \frac{O_m}{C_m}$ **EndForEach**;

$SELECTION$:
ForEach $m \in M$ **Do**
If $Random \leq Min(1 - g_m + B; 1)$
Then $P_s = P_s \cup \{m\}$;
Else $P_r = P_r \cup \{m\}$;
EndIf;
EndForEach;
Sort the elements of P_s;
$ALLOCATION$:
ForEach $m \in P_s$ **Do** $F_a(m)$ **EndForEach**;
Until *Stopping-criteria are met*;
Return ($BestSolution$);
End *Simulated_Evolution*.

Figure 1: Simulated Evolution algorithm.

maximum value, or no noticeable improvement to the population *goodness* is observed after a number of iterations. Another possible stopping criterion could be to run the algorithm for a prefixed number of iterations (see Figure 1).

3 Simulated Evolution Algorithm (SimE) for Logic Design

3.1 Problem Formulation

The problem of digital logic circuit design can be formulated using this generic model as follows: given a set M of all types of logic gates as in Table 1 and a set L of $|L|$ locations, $|L| \leq n$, it is required to select some of the elements from M and allocate them into the L distinct locations to produce a required logic function given by its truth table and to have this allocation to be minimal according to some cost function (power, area and delay)[2].

To formulate digital logic design in terms of the above state model, choose $M = \{1, 2, \ldots, k\}$ and $L = \{1, 2, \ldots, |L|\}$. A state is defined as the onto function S: $m = \{1, 2, \ldots, k'\} \rightarrow \{1, 2, \ldots, |L|\}$ where $m \subseteq M$. Figure 2 is a representation of the digital logic design problem addressed in this context. In this case one additional constraint is required, which can be stated as $S(i) \neq S(j)\ \forall i \neq j$, i.e., no two elements are assigned to the same location. The cost of a state, *Cost(S)* is a compound cost considering the correctness of the outputs of the solution circuit matching the required function truth table and other metrics such as gate count, power, delay and area. A detailed explanation of the *Cost(S)* function proposed is given later.

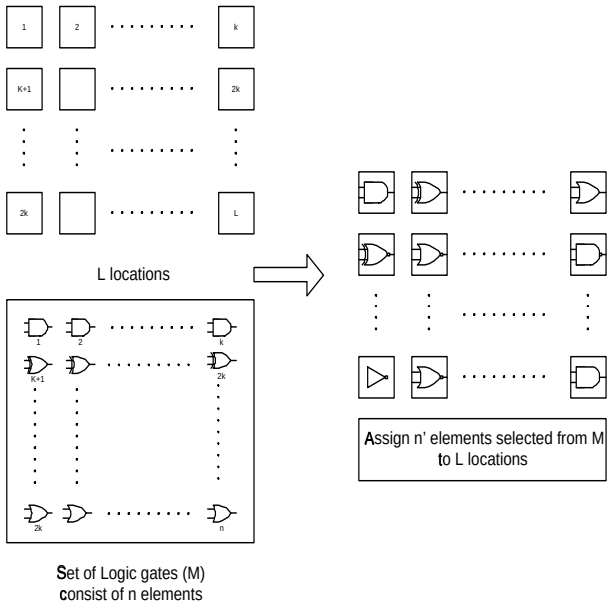

Figure 2: Representation of digital logic design problem.

3.2 Circuit Encoding

In order to represent a digital logic circuit, a two-dimensional matrix is used as depicted in Figure 3. This representation has been adopted by Coello and Miller [3, 6]. This type of data structure is very much similar to the structure of digital circuit. Therefore, the genotype-phenotype mapping becomes an easy task [5].

The size of the matrix is variable and it is relative to n where n is number of inputs of the circuit. An initial value for the size of the matrix is given, which is equal to n [7]. During iterations columns and/or rows can be added or removed.

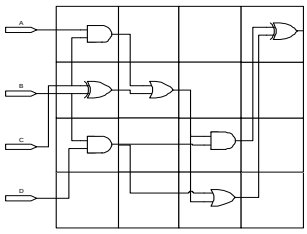

Figure 3: Example of circuit in matrix representation.

Each cell of the matrix is considered to be an individual. The collection of all individuals of the matrix represents a solution. Each cell of the circuit matrix is encoded in a triplet of inputs and gate type, as illustrated in Figure 4. The first two numbers are for the inputs (input1, input2) and the third indicates the gate type. A gate at position (i, j), where i is the column number and j is the row number, can

only be connected to the one at $((i-1), j')$ and j' can be any row of the previous column.

Input 1	Input 2	Gate type

Figure 4: Representation of an individual in matrix.

Table 1 lists different types of gates and their functions, along with their code for the gene encoding.

Code	Type	Function(F)
0	WIRE	a
1	NOT	a'
2	OR	$a + b$
3	AND	$a \cdot b$
4	XOR	$a \oplus b$
5	NOR	$(a + b)'$
6	NAND	$(a \cdot b)'$
7	XNOR	$(a \oplus b)'$

Table 1: Gate types used.

3.3 Proposed Goodness Measures

One major requirement of SimE is evaluating the goodness of each individual i of the population P. There are two types of cost measures or goodness measures associated with this problem. These two types are:

- ***functional cost measure*** where the correctness of the obtained logic circuit in matching the truth table of the required function is considered.
- ***optimization cost measure*** where the extent of the optimality of the logic circuit synthesized is taken into consideration. Such measures of optimality are power, delay and area.

Also, there are two types of ***functional cost measures*** and these are:

- *intrinsic functional cost measure* denoted by g_i
- *extrinsic functional cost measure* denoted by G_i

The *intrinsic functional cost measure* is related to the goodness of cell i in matching the objective truth table. While, the *extrinsic functional cost measure* is related to the effect coming from other cells having cell i as input for them. Both *Selection Function* and *Allocation Function* will use these *cost measures* in order to guide the search to reach to an optimal circuit. The selection function considers the extrinsic functional cost measure. On the other hand, the allocation function considers the intrinsic functional cost measure in evaluating the mutated cells after allocation in order to accept a certain move.

The formulation of the ***goodness measure*** has a great impact on the performance of the SimE algorithm and the quality of the solution obtained. In this paper, two proposed ***functional goodness measures*** are discussed in detail. The first *goodness measure* is called **Minterm Based Goodness** while the second one is called **Multilevel Logic Based Goodness**. These are explained below.

The ***Minterm Based Goodness Measure*** relies on the truth table of the required function to be implemented. Each cell i in the solution matrix is evaluated based on the number of correct outputs being generated of the required truth table. The following is a formulation of the ***Minterm Based Goodness Measure***:

$$g_i = O_i/C_i$$

where O_i is an estimate of optimal cost (all minterms generated), C_i is the actual cost of cell i. By letting, ρ_i = number of matching minterms of the i^{th} cell and n = number of inputs of the required circuit, then the Intrinsic Functional Minterm Based Goodness Measure is

$$g_i = \rho_i / 2^n$$

The Extrinsic Functional Minterm Based Goodness is computed using the procedure shown in Figure 5.

```
/*g_i: Intrinsic Functional Goodness of cell i */
/*g_j: Intrinsic Functional Goodness of the input cell of cell i */
/*G_i: Extrinsic Functional Goodness of cell i */
/*G_j: Extrinsic Functional Goodness of the input cell of cell i */
/*n number of inputs, n^2 number of cells in the matrix */
  For i = n^2 downto 1 Do  /* scan all cells from last to first */
    For All js cells in the previous columns of i Do
      If g_i > g_j or G_i > g_j Then

        If  g_j < g_i Then G_j = (g_i + g_j)/2
        Else If g_j < G_i Then G_j = (G_i + g_j)/2
        Else If gate type of i is inverter Then
          g_j = g_i and G_j = G_i
        EndIf
      EndIf
    EndFor
  EndFor
```

Figure 5: Extrinsic goodness measure calculation.

This goodness measure has been used to implement a version of the SimE algorithm denoted by SimE-G1.

The ***Multilevel Logic Based Goodness Measure*** is based on the assumption that the higher the level of a gate in a multilevel logic circuit, the more minterms are covered at the output of that gate. Therefore, the ***goodness*** of a gate is affected by the number of minterms covered at its output and the level where the gate is located. Figure 6 illustrates this assumption. Since the number of inputs of the circuit is

4, there are 16 patterns that should be generated at the output correctly. Initially, these patterns are distributed among the levels of the circuit evenly and progressively. Also, it is assumed that the initial number of levels is 4 since there are 4 columns in the search matrix. Therefore, a logic gate located at the second level should cover 8 patterns while a logic gate located at the first level should cover 4 patterns.

In general, for n inputs (2^n patterns) circuit, to have a goodness of 1 at a cell in level i, there should be $\lceil (2^n/n)i \rceil$ correct patterns produced at this cell. Thus, the *multilevel logic goodness measure* is formulated as follows:

$$g_i = \frac{\rho}{\lceil 2^n/n \rceil j}$$

where g_i is the *goodness* of cell i, j is the level number or column number, n is the number of inputs of the required circuit and ρ is the number of matching patterns at the output of cell i compared to the intended truth table.

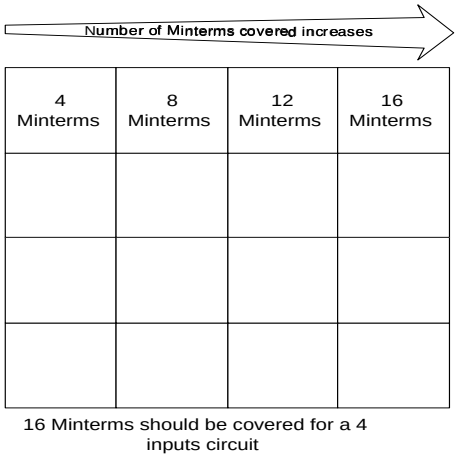

Figure 6: Multilevel logic goodness assumption.

Several scenarios result from this assumption, and these are:

- if a cell i at level j produces more than $((2^n/n)j)$;
- if none of the cells located at level j can produce $((2^n/n)j)$ nor any cell at level n can produce 2^n patterns; and
- intrinsic and extrinsic complications.

First, if a cell i at level j produces more than $((2^n/n)j)$, then $\rho_i > ((2^n/n)j) \Rightarrow g_i > 1$. The number of levels that the SimE algorithm is searching should be decreased by a factor relative to the number of patterns produced by the cell i. The number of levels is now $l = \lceil (2^n/\rho_i) \rceil$. Also, the number of patterns that should be covered at each level will be changed accordingly. A cell i at level j should cover $\rho_i + 1$ patterns. At $j + 1$ level, there should be

$$(\rho_i + 2) + \left\lceil \frac{2^n - (\rho_i + 1)}{l} \right\rceil$$

patterns covered. Then, the maximum number of patterns at level $i > j$ is

$$(\rho_i + 2) + \left\lceil \frac{2^n - (\rho_i + 1)}{l} \right\rceil (i - j)$$

Eventually, our goodness measure also should be changed accordingly to become

$$g_i = \rho/m$$

where m is the new maximum number of patterns that gate i should generate at level j.

The second case indicates that the number of levels is less than what it should be. Therefore, the number of levels has to be increased. Also, the ranges of the maximum number of patterns will change if the number of levels increases. The increase in number of levels is done gradually, i.e., one by one. If l is the number of levels, then the number of pattern coverage for each level is $\lfloor 2^n/l \rfloor$. Therefore, the *goodness* measure for cell i located at level j will be changed to:

$$g_i = \frac{\rho}{\lfloor 2^n/l \rfloor j} \quad \text{if } j < l$$

$$g_i = \frac{\rho}{2^n} \quad \text{if } j = l$$

The third complication is similar to the one in the first goodness measure. A cell j with low goodness measure might feed cell i that has high goodness. Therefore, cell j will have high probability of being selected and mutated which might disturb the goodness of cell i. Therefore, the same approach as in the previous one in computing the intrinsic and extrinsic goodnesses is considered. In this case the *intrinsic goodness measure* is computed by using the multilevel logic based goodness measure equations for g_i.

This goodness measure has been used to implement a version of the SimE algorithm denoted by SimE-G2.

In order to optimize the circuit for other measures such as power, delay and area, a working circuit has to be first obtained. This means that the *functional goodness measure* should be equal to 1. Also, a *global optimization goodness cost function* is proposed. After allocation, the solution is evaluated and if there is an improvement in the global cost function, the solution is made as permanent. The *global optimization goodness cost function* is denoted by G_O and is computed using the procedure in Figure 7. The goodness of the solution considering the j^{th} objective is computed as follows:

Circuit	Coello [4]			SimE-G1			SimE-G2		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
circuit1	12393.00	5.05	4.38	12393.00	5.05	4.38	12393.00	5.05	4.38
circuit2	21870.00	6.18	6.61	15451.60	5.26	6.77	13150.93	5.21	5.32
circuit3	19926.00	4.34	5.15	10843.10	3.00	4.35	10745.45	3.12	3.96
circuit4	1458.00	0.005	0.66	1458.00	0.005	0.66	1458.00	0.005	0.66
circuit5	27945.00	8.76	7.89	13412.44	6.67	6.29	11723.95	5.61	5.14
majority	21141.00	7.53	6.07	14029.51	4.88	4.41	13977.51	4.50	5.12
xor8	32805.00	9.53	11.64	20880.23	6.02	9.78	20655.00	5.90	9.32
xor9	35266.00	11.34	13.79	23814.00	9.57	10.65	23814.00	9.57	10.65

Table 2: Comparison with Coello [4] technique in terms of area, delay and power.

/*g_i: Intrinsic Functional Goodness of cell i */
/*G_O: Global Optimization Goodness of P */
/*P: the current population (solution) */
/*k: number of objectives for optimization */
/*g_j: the goodness of the solution considering the j^{th} objective */
/*w_i: a weight for functional cost */
/*w_j: a weight for optimization cost */

If $g_i < 1$ **Then**
  $G_O = w_i g_i$ $[0, w_i]$
ElseIf $g_i = 1$ **Then**
  $G_O = w_i g_i + \sum_{j=1}^{k} w_j g_j$ $[w_i, 1]$
EndIf

Figure 7: Global optimization goodness.

$$g_j = \frac{1}{1+\delta}$$

and

$$\delta = \frac{N}{N_{max}}$$

N is the current cost of the objective function (power, area and delay) and N_{max} is the maximum acceptable value for the given constraint.

Linear aggregation function is used for the evaluation of all objectives of the solution. It maps all objectives into a single metric that indicates how good the solution is. SimE works on two solutions, the current solution and the candidate (mutant) solution at each iteration. Therefore, neither is better than the other on all objectives. Since SimE works on two solutions during iteration, Pareto ranking is not applicable due to the lack of the ranking set. A linear aggregation function will be an efficient and fast choice compared to Pareto ranking.

4 Results and Comparison

Several circuits of different degrees of complexity have been used to test the proposed approach. The two proposed algorithm, SimE-G1 and SimE-G2 are compared to Genetic Algorithm (GA) approach similar to that reported in [3, 4]. Table 2 shows the quality of solutions in terms of area, delay and power for GA, SimE-G1 and SimE-G2. In Table 2, circuit1 to circuit5 are randomly generated circuits while the last three circuits are selected from ISCAS'85 benchmark. Table 3 shows the average execution time for the given test cases. It is obvious that Coello's approach is slower compared to SimE-G1 and SimE-G2 in time.

Circuit	Coello [4](s)	SimE-G1 (s)	SimE-G2 (s)
circuit1	91.66	12.52	10.85
circuit2	102.32	15.09	11.01
circuit3	155.78	20.14	14.07
circuit4	275.10	20.50	13.10
circuit5	266.36	21.25	15.87
majority	6290.32	224.71	119.62
xor8	7430.01	221.10	120.34
xor9	10856.55	320.92	273.63

Table 3: Comparison with Coello's [4] technique in terms of execution time.

The tables show that there are significant improvements in terms of area, delay, power and execution time for most cases. For the purpose of this paper, two examples were chosen to illustrate this approach.

Since the Genetic Algorithm (GA) reported in [3, 4] considers only functional fitness and number of gates as cost function, modification in cost function has to be made in order to include power, delay and area optimization in the cost function instead of number of gates used. The population size used in all circuits for GA is 1000. Two-point crossover is used as in [3, 4] with crossover rate 0.95. The mutation rate is initialized to 0.01.

Each experiment is run 50 times and the average cost function (fitness) for GA and average goodness for SimE for the 50 runs is reported. Also, the best solution obtained

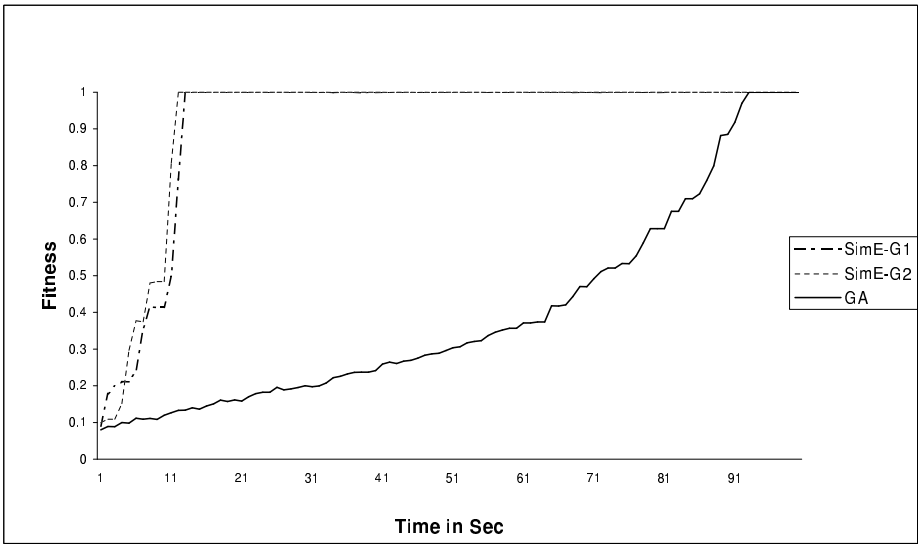

Figure 8: Average global cost function for Example 1.

Algorithm	Resulting Function	Gate Types
GA	$F = ((A \oplus B) \oplus AD) + (C + (A \oplus D))'$	1 AND, 2 ORs, 3 XORs, 1 NOT
SimE-G1	$F = A'B + A((BD)' + C'D)$	4 ANDs, 2 ORs, 3 NOTs
SimE-G2	$F = A'B + A((BD)' + C'D)$	3 ANDs, 2 ORs, 2 NOTs, 1 NAND

Table 4: Results produced by GA, SimE-G1 and SimE-G2 for Example 1.

in the 50 runs is reported for comparison in power, delay and area. The library used for the evaluation of power, delay and area is the MOSIS 0.25μ CMOS cell library.

Example 1

The first example has 4 inputs and one output where F is 1000111111100101 (circuit 2). It is noticed that SimE-G1 and SimE-G2 outperformed GA since more cells are assigned to higher goodness locations. As a result, the global average goodness increased resulting in converging into better solutions in lesser number of iterations compared to GA. The results for global cost function is shown on Figure 8. Also, Table 4 shows the best resulting circuit outcome from running SimE-G1, SimE-G2 and GA.

Example 2

The second example is a 9 inputs odd parity circuit (xor9). SimE-G1 and SimE-G2 produced results in almost 300 seconds while GA could not generate a working circuit in the first 500 seconds. The results for global cost function is shown on Figure 9.

5 Discussion and Conclusions

All circuits experimented with showed that SimE-G1 and SimE-G2 have better convergence than GA in time and quality of circuit. The reason is that SimE algorithm uses less memory and less computation time. SimE works on one solution only while GA works on a large number of solutions (population size). The number of crossovers and mutations that GA will perform in every iteration is large which will require more CPU time. On the other hand, SimE requires less CPU time because it uses the ***goodness measure*** to guide the algorithm through the search space. ***Goodness measure*** is used during selection and allocation, therefore, SimE algorithm will assign more cells to better goodness locations in every iteration. The more knowledge about the problem incorporated into the ***goodness measure***, the better the performance of SimE algorithm in terms of CPU time and quality of solution. On the other hand, GA evaluates the solution after crossover and mutation. Therefore, crossover and mutation is done blindly and most of the time the solution space is explored randomly.

We have introduced a technique to design combinational logic circuit using Simulated Evolution Algorithm (SimE). Also, two goodness measures have been proposed. The pro-

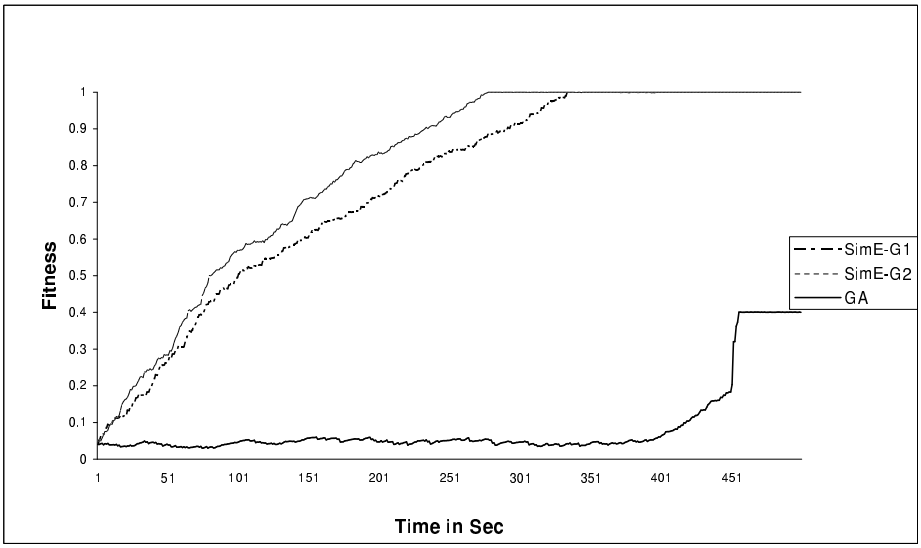

Figure 9: Average global cost function for Example 2.

posed algorithm SimE-G1 and SimE-G2 are compared with GA. Both SimE-G1 and SimE-G2 showed better results in all the cases.

Acknowledgment

The authors would like to acknowledge the support received from King Fahd University of Petroleum & Minerals under project entitled “Iterative Heuristics for the Design of Combinational Logic Cicuits”.

Bibliography

[1] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*. Wiley-IEEE Press, 2000.

[2] Sadiq M. Sait and Habib Youssef. *VLSI Physical Design Automation: Theory and Practice*. World Scientific Press, 1999.

[3] C. A. Coello, A. D. Christiansen, and A. H. Aguirre. Use of Evolutionary Techniques to Automate the Design of Combinational Circuits. *International Journal of Smart Engineering System Design, Elsevier Science*, 2(4):299–314, June 2000.

[4] C. A. Coello, A. D. Christiansen, and A. H. Aguirre. Towards Automated Evolutionary Design of Combinational Circuits. *Computers and Electrical Engineering, Pergamon Press*, 27(1):1–28, Jan. 2001.

[5] J. F. Miller, T. Fogarty, and P Thomson. Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science, John Wiley and Sons, Chichester*, pages 105–131, 1998.

[6] J. F. Miller, D. Job, and Vassilev V. K. Principles in the Evolutionary Design of Digital Circuits - Part I. *Journal of Genetic Programming and Evolvable Machines*, 1(1):8–35, 2000.

[7] J. F. Miller and P. Thomson. Discovering Novel Digital Circuits Using Evolutionary Techniques. *IEE Colloquium on Evolvable Systems, Savoy Place, London*, March 1998.

[8] J. F. Miller and Peter Thomson. A Developmental Method for Growing Graphs and Circuits. *GECCO’03*, 1(1):8–35, 2003.

[9] R. M. Kling and P. Banerjee. ESP: A New Standard Cell Placement Package using Simulated Evolution. *Proceeding of 24th Design Automation Conference*, pp. 60–66, 1987.

[10] Y. Saab and V. Rao. Stochastic Evolution: A Fast Effective Heuristic for Some Generic Layout Problems. *27th ACM/IEEE Design Automation Conference*, pp. 26–31, 1990.