International Journal of Neural Systems, Vol. 16, No. 3 (2006) 163–177

World Scientific
www.worldscientific.com

EVOLVING DIGITAL CIRCUITS USING HYBRID PARTICLE SWARM OPTIMIZATION AND DIFFERENTIAL EVOLUTION

PHILLIP W. MOORE* and GANESH K. VENAYAGAMOORTHY†
Real-Time Power and Intelligent Systems (RTPIS) Laboratory,
Department of Electrical and Computer Engineering,
University of Missouri — Rolla, 1870 Miner Circle,
Rolla, Missouri 65409
**pwmpn2@umr.edu*
†*gkumar@ieee.org*

This paper presents the evolution of combinational logic circuits by a new hybrid algorithm known as the Differential Evolution Particle Swarm Optimization (DEPSO), formulated from the concepts of a modified particle swarm and differential evolution. The particle swarm in the hybrid algorithm is represented by a discrete 3-integer approach. A hybrid multi-objective fitness function is coined to achieve two goals for the evolution of circuits. The first goal is to evolve combinational logic circuits with 100% functionality, called the *feasible circuits*. The second goal is to *minimize* the number of logic gates needed to realize the feasible circuits. In addition, the paper presents modifications to enhance performance and robustness of particle swarm and evolutionary techniques for discrete optimization problems. Comparison of the performance of the hybrid algorithm to the conventional Karnaugh map and evolvable hardware techniques such as genetic algorithm, modified particle swarm, and differential evolution are presented on a number of case studies. Results show that feasible circuits are always achieved by the DEPSO algorithm unlike with other algorithms and the percentage of best solutions (minimal logic gates) is higher.

Keywords: Combinational logic circuits; differential evolution; evolvable hardware; hybrid algorithms; particle swarm optimization.

1. Introduction

Recently, the demand for digital controllers and instrumentations has increased and reliability and security of hardware platform is important for the homeland security, military, space applications, and many others. Evolvable hardware is a dynamic, automated, and reconfigurable design of hardware. Hardware evolution occurs in order to survive harsh and unknown changes in environmental settings through self-reconfiguration. This rising field applies evolutionary and other related algorithms, similar to the Darwinian process of evolution, to automate, design, and adapt physical, reconfigurable, and morphable structures including: robots, electronic configurations, field programmable gate arrays, and embedded systems.

There are many methods for the design of combinational logic circuits. Methods generally used include Karnaugh maps and Quine-McCluskey's.[1–3] The problem with the human designs (HDs) is that they become cumbersome and problematic when the number of inputs, number of outputs, and complexity of the function increases. The complexity of the combinational circuit depends on the number of gates and number of input/output combinations in the circuit. For real world applications, combinational circuits with minimal area, power consumption, and speed propagation delay are required for cost effective and high speed circuit realization.

The evolutionary design of electronic circuits refers to a self-sufficient process in which a highly efficient circuit may transpire in a population of interacting instances of a logic function.

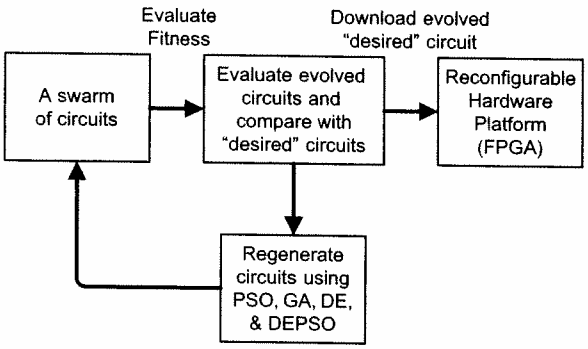

Fig. 1. "Desired" circuit hardware evolution.

Evolutionary hardware design, a process by which electronic circuits evolve, increases the performance and efficiency of combinational circuits. Many papers have reported on the design of combinational circuits using genetic algorithms[4,5] and particle swarm optimization.[6–8] The basic process of hardware evolution is illustrated in Fig. 1. The "desired" circuit refers to the circuit that maps 100 percent exactly the outputs for corresponding input combinations typically given by a truth table for digital circuits. The hardware evolution is carried out until the "desired" circuit is evolved and then downloaded to a reconfigurable hardware platform. This sequential process is commonly referred to as extrinsic evolution.

Particle swarm optimization (PSO), an optimization procedure developed by James Kennedy and Russell Eberhart, is based on the movement patterns of flock of birds and schools of fish.[9–11] The PSO operates in a hyper-dimensional search space containing the solutions for given problems. The PSO uses a population of random particles to explore and exploit a search space. A particle's velocity and position are used to determine its progression throughout the evolution process.

Differential Evolution (DE) is an optimization procedure developed by Rainer Storn and Kenneth Price.[12–14] Differential evolution is an evolutionary process that combines the crossover, selection, and mutation operators from the genetic algorithm (GA) into one step. Two sets of two random parents are used in the crossover, selection, and mutation step to produce the new offspring for the population.

This paper presents the application of a hybrid algorithm called the Differential Evolution Particle Swarm Optimization (DEPSO), based on the combined concepts of a modified particle swarm algorithm and differential evolution, for the evolving combinational logic circuits. In addition, the paper presents modifications made to the canonical PSO to address discrete optimization problems such as evolving combinational logic circuits. To the knowledge of the authors, on average, results are presented to demonstrate that the DEPSO evolves feasible circuits with minimal gates more frequently with a higher fitness other than evolutionary algorithms reported in the literature.

The rest of the paper is organized as follows. Section 2 explains the combinational logic circuit representation. Section 3 describes the PSO, the modified PSO, the DE, and the DEPSO algorithms for circuit evolution. Section 4 presents the results with GA, PSO, DE, and DEPSO algorithms. The discussion and conclusion are given in Secs. 5 and 6 respectively.

2. Combinational Logic Circuit Representation

When using evolutionary methods, circuits are represented by each particle or individual of the population as a possible circuit solution. The circuits are represented using a matrix of size $m \times n$. Figure 2 shows a circuit representation using a 3×3 matrix consisting of 9 cells. Each matrix element accommodates a 2-input gate. The logic gates in the library of gates are represented with numbers one through five. A *wire* is represented by a one, an *AND* gate is represented by a two, an *OR* gate is represented by a three, a *NOT* gate is represented by a four, and an *XOR* gate is represented by a five. The inputs to the gates are from the preceding column. The population of circuits is represented using

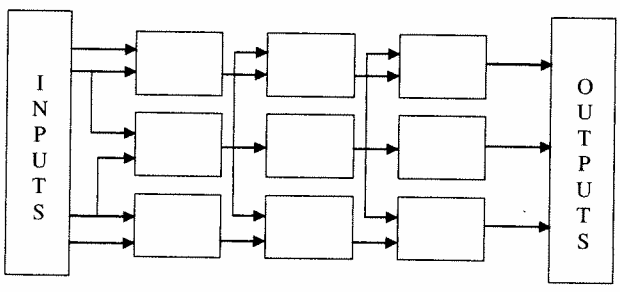

Fig. 2. Circuit Layout for 3×3 with 9 two-input gate cells.

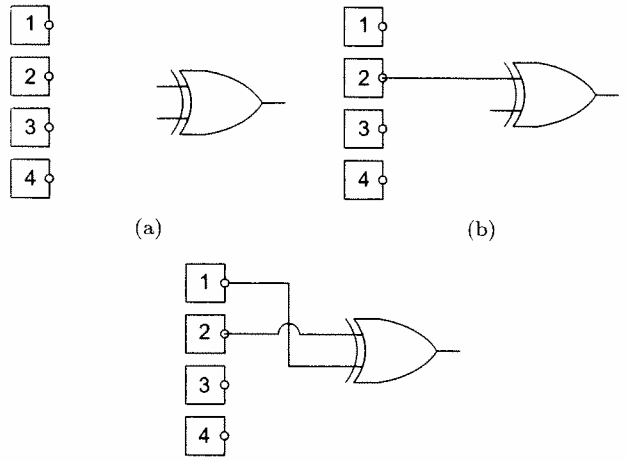

Fig. 3. (a) A fourth dimension value of 1, A(x,x,x,1), represents evolution of a logic gate. (b) A fourth dimension value of 2, A(x,x,x,2), represents the evolution of the first input to the logic gate. (c) A fourth dimension value of 3, A(x,x,x,3), represents the evolution of the second input to the logic gate.

a 4 dimensional matrix "A," of size $i \times m \times n \times o$. The first dimension, i, defines an individual's identity in the population. The second and third dimensions of matrix, "A," represent the row and column where the gate and its indices are located respectively. The fourth dimension, o, of the matrix can take the value one, two, or three. These values represent the gate or index/input, being evolved. For example, the fourth dimension with value one represents a gate and values of two and three represent the first and second index to the gate. An illustration with examples is given in Fig. 3 for the fourth dimension with values one, two, and three respectively.

3. Particle Swarm, Modified Particle Swarm, Differential Evolution, and Hybrid DEPSO

The circuit evolution uses a hybrid multi-objective fitness function given in (1) and (2) when evolving combinational logic circuits involving two tasks.[6,17] The first task is to generate feasible circuits. The second task is to find a feasible circuit with the minimum number of gates. The total fitness function, (3), the sum of the first and second fitness evaluations given by (1) and (2) respectively, is used by the algorithms to evolve the circuits. The following subsections describe the algorithms — particle swarm, the modified particle swarm, the differential evolution, and the DEPSO.

$$Fitness1 = 1 - \left(\frac{number\_correct\_outputs}{total\_number\_of\_outputs}\right) \quad (1)$$

$$Fitness2 = \begin{Bmatrix} m \times n, & not\_feasible\_circuit \\ no\_gates, & feasible\_circuit \end{Bmatrix} \quad (2)$$

$$Fitness = Fitness1 + Fitness2 \quad (3)$$

3.1. *Particle swarm*

In the canonical PSO algorithm, the velocity of a particle i in dimension d is obtained using (4), and the particle's position in dimension d is obtained using (5). The subscript "i" from (5) takes on values 1 to L, where L is the maximum number of particles in the population. Each particle is assigned to a positional point in a D-dimensional space. The first term $v_{id}(t+1)$ is the new velocity for the ith particle. The second term $v_{id}(t)$ is the current velocity for the ith particle. The values for w, c_1, and c_2 represent the inertia weight, cognitive acceleration constant and social acceleration constant respectively. The terms *rand1( )* and *rand2( )* are random numbers between 0 and 1. The random values are different for each particle's dimension. The term $p_i = (p_{i1}, p_{i2}, \ldots, p_{iD})$ is the current personal best fitness for the ith particle. In the canonical PSO there are different types of population topologies. The main two are the star and the ring topology. The star topology allows for each particle to communicate to the swarm's best particle ($g_{i,m,n,o}$). The ring topology allows for each particle to communicate to the best particle in its neighborhood ($l_{i,m,n,o}$). In this paper, the neighborhood version (ring topology) is used, as shown in Fig. 4.

The term $l_i = (l_{i1}, l_{i2}, \ldots, l_{iD})$ is the current neighborhood best fitness for the ith particle. Finally, the term $x_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$ is the position for the ith particle in the population.

$$v_{id}(t+1) = \begin{pmatrix} w \times v_{id}(t) \\ + c_1 \times rand1() * (p_{id}(t) - x_{id}(t)) \\ + c_2 \times rand2() * (l_{id}(t) - x_{id}(t)) \end{pmatrix} \quad (4)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \quad (5)$$

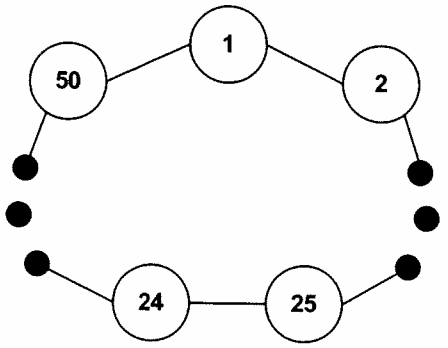

Fig. 4. Ring topology (local best) for PSO.

3.2. *Modified Particle Swarm*

In the modified PSO algorithm, modeled after the PSO algorithm presented in by Coello[6] and used in this paper, each particle is assigned an initial velocity which is updated according to (6).

$$v_{i,m,n,1}(t+1) = round\begin{pmatrix} w \times v_{i,m,n,1}(t) \\ + c_1 \times rand1() \times S_1 \\ + c_2 \times rand2() \times S_2 \end{pmatrix} \quad (6)$$

where $S_1 = p_{i,m,n,1}(t) - x_{i,m,n,1}(t)$ and $S_2 = l_{i,m,n,1}(t) - x_{i,m,n,1}(t)$.

A particle's velocity is ascertained from three influential factors. The first term in (6) consists of the current velocity $v_{i,m,n,1}(t)$ multiplied by a constant w known as the inertia weight. The second term is obtained from the product of c_1, *rand1( )* and S_1(which is $p_{i,m,n,1}(t) - x_{i,m,n,1}(t)$) where c_1 is the cognitive acceleration constant. The third term is obtained from the product of c_2, *rand2( )* and S_2 (which is $l_{i,m,n,1}(t) - x_{i,m,n,l}(t)$) where c_2 is the social acceleration constant. The subscript "i" takes on the values from 1 to L, where L is the maximum number of particles in the population. The subscript "m" takes on the values from 1 to M, where M is the maximum number of rows for the circuit matrix. The subscript "n" takes on values from 1 to N, where N is the maximum number of columns for the circuit matrix. To note, the subscripts "i", "m" and "n" are mentioned in Sec. 2. The current personal best fitness solution for the ith particle is $p_{i,m,n,1}(t)$. The current solution for the ith particle is $x_{i,m,n,1}(t)$. For $p_{i,m,n,1}(t)$ and $x_{i,m,n,1}(t)$, the fourth subscript is set to one (the fourth dimension "o" takes on values from one to three and is described in Sec. 2). The current neighborhood best fitness solution for the ith particle is $l_{i,m,n,1}(t)$. For $l_{i,m,n,1}(t)$, the fourth dimension value is also set to one. The rounded sum of these three terms yields the new velocity of the particle, $v_{i,m,n,1}(t+1)$. The round function is the standard round function which floors the value of the summation of the three terms if it has a decimal value less than 0.5. Otherwise the round function will take the ceiling value of the three term summation.

In the velocity equation for the modified PSO given in (6), the velocity's fourth dimension is set to one as well as the x_{id}, p_{id} and l_{id}. Once the velocity is calculated using (6), the velocity values with $o = 1$ is copied to velocity values for $o = 2$ and $o = 3$, using (7) and (8). The velocity update equation in (6) updates the velocity for all of the gates for each particle. When (7) and (8) are applied, the velocities of the first and second index to a gate will have the same value as the velocity of that gate. Not updating the velocities for gate indices should decrease the search capability in theory but it should also decrease the amount of computational time required on the calculations per iteration. Through the case studies in this paper, it is observed that search takes lesser number of iterations. Fortunately, reducing the computational time for velocity calculations did not impair the results of the algorithm, but instead, aided it.

$$v_{i,m,n,2}(t+1) = v_{i,m,n,1}(t+1) \quad (7)$$

$$v_{i,m,n,3}(t+1) = v_{i,m,n,1}(t+1) \quad (8)$$

Taking the rounded value for the velocity is performed in order to obtain discrete values for the PSO. Limiting the PSO to the discrete (integer) domain will confine the velocity ability to change continuously as with continuous PSO algorithms. In order to compensate for this tradeoff, the numbers representing the gates in the gate library is set to a range from zero to a number higher than the number of gates in the library (four times more). In modifying the gate representation in such a way, the velocity will be able to accelerate a particle over more values as a way of imitating a continuous particle swarm. The gate representation values are then condensed, using a modular function, during evaluation in order to produce numbers ranging from zero to four that represent the five gates in the gate library. The modular function takes the modular value of a gate representation number by the number of gates in the gate library. This yields numbers in the range

of zero to four enabling a clear identification of the gate representation number to each of the five logic gates.

A copy of the velocity is then normalized to a value between 0 and 1. In order to obtain a normalized velocity, a copy of the velocity is first constrained to values no larger than 4 and no smaller than -4 by setting minimum and maximum constraints on the velocity values. Using the constraints, the absolute value of the velocity value divided by the maximum velocity will yield the normalized velocity value. The reasons for using a normalized velocity is to model the canonical discrete PSO after the binary PSO in order to form an integer encoded PSO. For these reasons, the velocity is condensed into values from 0 to 1 to make a probability threshold. The normalized velocity is then used to update the positions of the particles as given in (9) based on a comparison of the normalized velocity and a flip function.[6] The flip function is a basic coin-flip percentage function; the percent of times that a coin lands on heads within 10 flips is the output of the flip function. The first and second flip functions in (9) have the same functionality. The first flip function output is compared to the normalized velocity. If the normalized velocity is less than the first flip function output, then the current particle's position will update to become $l_{i,m,n,o}$. Otherwise, the second flip function is called. The second flip function is then compared to one minus the normalized velocity. If this value is less than the output from the second flip function, then the current particle's position will update to become $p_{i,m,n,o}$. Otherwise, the current particle's position will remain the same.

$$x_{i,m,n,o}(t+1) = \begin{cases} l_{i,m,n,o}(t) & N\big(v_{i,m.n,o(t+1)}\big) < FF_1 \\ p_{i,m,n,o}(t) & \big[1 - N\big(v_{i,m,n,o(t+1)}\big)\big] < FF_2 \\ x_{i,m,n,o}(t) & otherwise \end{cases} \tag{9}$$

where $N(v_{i,m,n,1}(t+1))$ is the normalized velocity, FF_1 is the first flip-function and FF_2 is the second flip-function.

After the position updates, PSO particles go through a mutating function. This function mutates the particles at a given rate. The mutating function keeps the position of the particles from prematurely converging to $p_{i.m.n,o}$ and $l_{i,m,n,o}$ solutions. The subscript "o" takes on values from one to three. An explanation of this dimension is given in Sec. 2. Solutions from the PSO algorithm without the mutation function for this problem studied in this paper were not reaching optimal, but instead were prematurely converging within one to two iterations. The cause of this premature convergence comes from the nature of the positional update equation. The positional update equation can only change the values of a particle to either $p_{i,m,n,o}$ or $l_{i,m,n,o}$ values. This means that the particles of the population are trapped within a very small search space consisting of all combinations of the particles' initialization values. Over the iterations in the particle swarm algorithm, the search space decreases until it has reached the smallest search space size. The smallest size search space is when the particle's $p_{i,m,n,o}$ and $x_{i,m,n,o}$ values have all reached the $l_{i,m,n,o}$ solution. Once this has happened, then the velocity update and position update is meaningless. In order to discontinue the retraction of the search space, a continuous mutation function must continue to change particle's positions at a given mutation rate. A continuous mutation function will ensure that the search space stays large enough for the PSO algorithm to effectively search for optimal solutions. The mutation

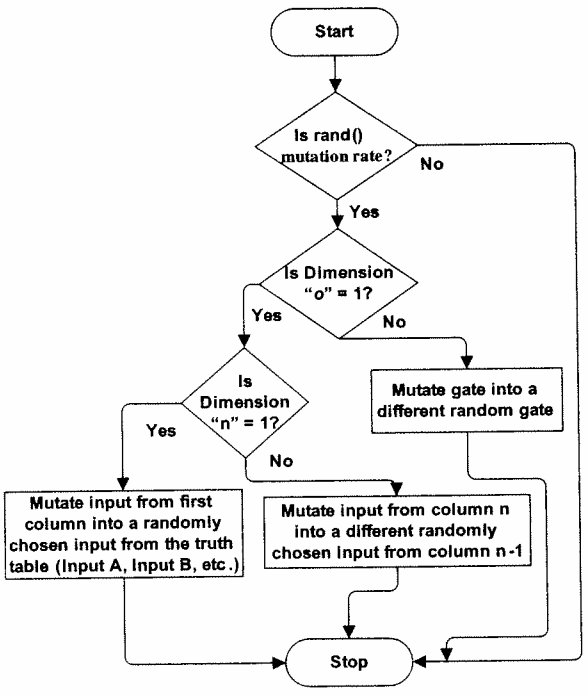

Fig. 5. Flowchart for the mutation operator.

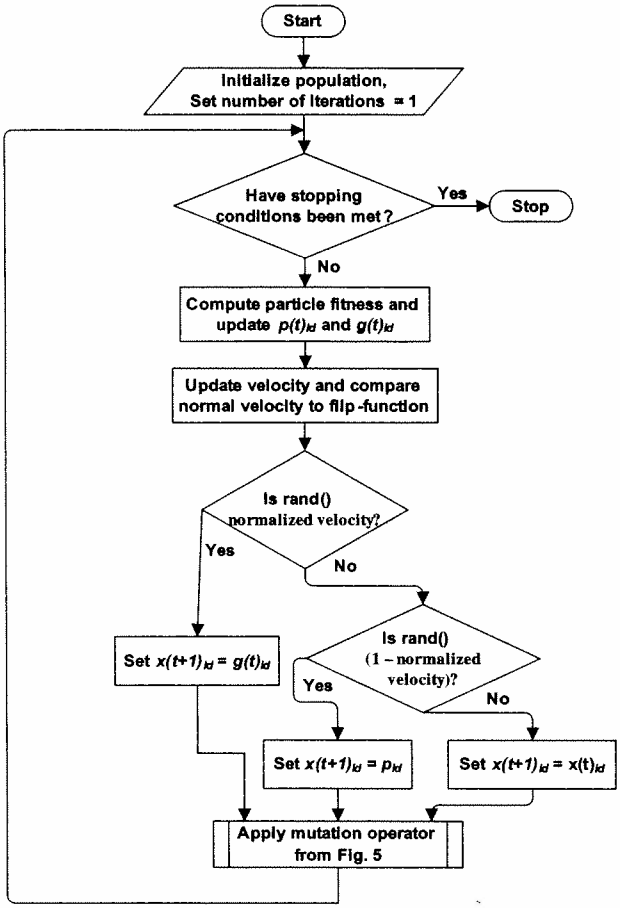

Fig. 6. Flowchart for the modified particle swarm algorithm.

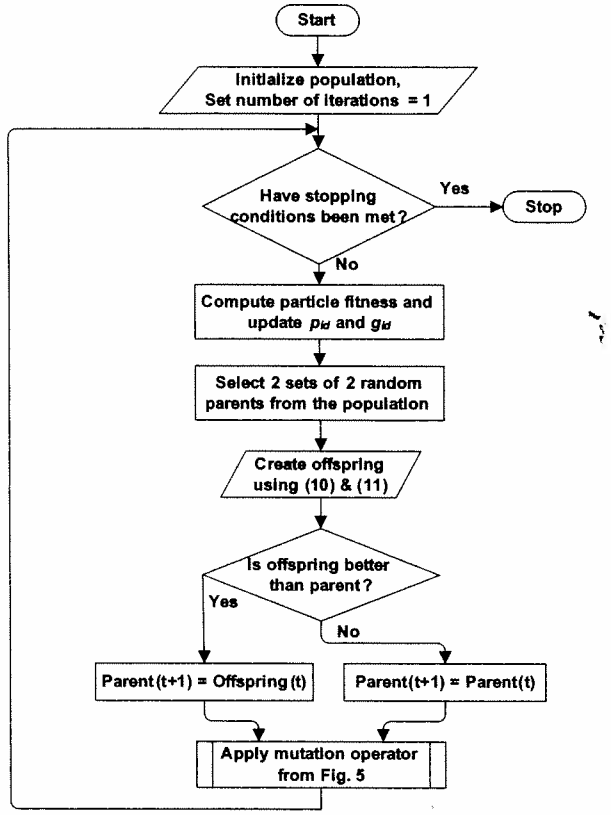

Fig. 7. Flowchart for the differential evolution algorithm.

rate used in this study is 10%. A general flowchart for the mutation operator and the modified particle swarm algorithm is given in Figs. 5 and 6, respectively. In Fig. 5, when dimension $o = 1$, then the object sent into the mutation function is a gate. Otherwise the object is either the first or second input to a gate. In Fig. 5, when dimension $n = 1$, then the gates and inputs, sent into the mutation function, is from the first column.

3.3. *Differential evolution*

The DE algorithm evolves individuals/particles in order to increase the convergence to minimal gate solution.[15,16] The DE utilizes a particle's best position p_{id} and a mutation operator to evolve the particle into an offspring, given by (10). The mutation operator selects two sets of two parents from the population and performs a mutation on the parents as shown in (11). For every particle $P_k(k = 1, 2, \ldots, L)$, where L is the maximum number of particles, four other random parents P_1, P_2, P_3, and P_4 are selected from the population and mutated to produce an offspring. The offspring is compared to the parent and the best-fit particle is kept for the next generation as in (12) (P_i is the parent of the ith particle, and p_i is the personal best position of the ith particle). The fitness is then updated so that the PSO can use the new updated p_{id} and l_{id}. The flowchart for the differential evolution algorithm is given in Fig. 7.

$$Offspring(t) = p_i(t) + \varsigma_\Delta(t) \tag{10}$$

$$\varsigma_\Delta(t) = \frac{\big(P_1(t) - P_2(t)\big) + \big(P_3(t) - P_4(t)\big)}{2} \tag{11}$$

$$P_i(t+1) = \begin{cases} P_i(t), & fit(P_i(t)) \le fit(Offs(t)) \\ Offs(t), & fit(P_i(t)) > fit(Offs(t)) \end{cases} \tag{12}$$

where *Offs* and *fit* means *Offspring* (10) and *Fitness* (3) respectively.

3.4. *Hybrid DEPSO*

Modified particle swarm optimization and differential evolution combine together to form a hybrid algorithm called the DEPSO.[15] The hybrid algorithm evolves combinational logic circuits to produce feasible minimal gate circuits. The modified PSO algorithm resembles the 3-integer representation developed by Coello Coello.[6] The DE algorithm[16] randomly selects two sets of two parents from the population to evolve. A mutation operator, that mutates at a given mutation rate, diversifies the population in order to decrease the population convergence to the p_{id} and l_{id} solutions. The DEPSO algorithm flowchart is shown in Fig. 8.[18–20] The sequences of operation in the hybrid algorithm are — first update the fitness, p_{id}, and l_{id} values, evolve the population with the PSO algorithm, evolve with the DE algorithm, and then apply the mutation operator shown in Fig. 5. The equations for the DEPSO process use a combination of (6) through (12) to evolve combinational logic circuits.

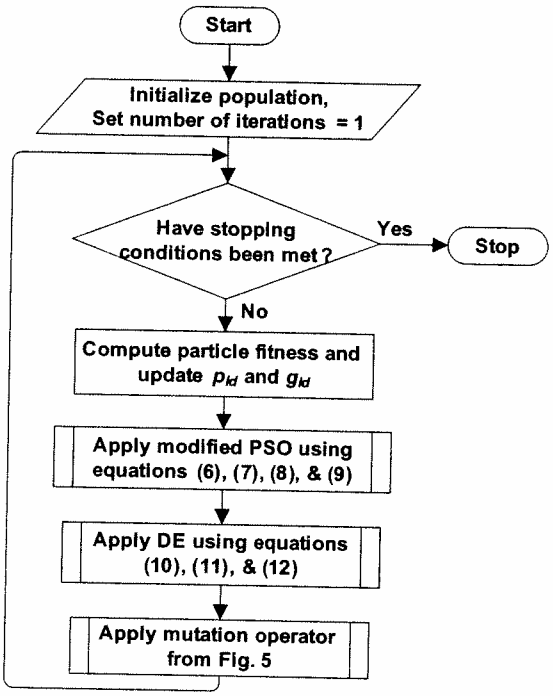

Fig. 8. Flowchart for the DEPSO algorithm.

4. Case Studies and Results

Six case studies (Tables 1 and 2) are carried out using the PSO, DE and DEPSO. Results are compared with those obtained by the HD and the GA algorithm reported in literature.[6,21] The results shown in Tables 3 through 7 were averaged over 20 test runs. The last case study results given in Table 8, was averaged over 10 test runs. All six case studies had used a population size of 50, PSO parameters of $w = 1$, $c_1 = 0.2$, and $c_2 = 0.2$, and a 10% mutation rate. The PSO has a ring topology with a neighborhood size of 3. The neighborhood topology is depicted in Fig. 4 with each numbered node referring to a particle.

With the parameter of w equal to one, the inertia weight will have no effect on the velocity update equation. Values of w, c_1, and c_2 are taken from the values used in the similar PSO algorithm.[6] Case studies 1 and 2 used a circuit size of 4×4, case studies 3 through 5 used a circuit size of 5×5, and case study 6 used a circuit size of 6×6. The circuits evolved from the GA, PSO, and DEPSO are given in Figs. 9 through 23. The following subsections describe the details of each case study.

When analyzing the results of the six case studies, the algorithm that did the best is the algorithm that evolved the least number of gates for the case study, on average. The problem is in minimizing the number of gates for a circuit. Therefore, whichever algorithm can do this better on average is the algorithm that has performed better. In the table of results for each case study, a standard deviation for the average number of gates and average number of iterations have been added to more accurately portray the values obtained from the DE, PSO, and DEPSO.

The results for average fitness/average number of gates and for the average number of iterations are not given for the GA comparisons. The reason for this is due to the fitness function of the GA and evaluation methods the authors used for their GA.[6] The fitness function for the GA maximized the number of wires in the circuit rather than minimizing the number of gates. The difference in fitness evaluations from the DE and DEPSO makes the average fitness evaluation of the GA a different type of evaluation. Therefore, these results cannot be compared. The authors of the GA evaluated the number of iterations for each case study as a maximum set amount. This causes their average number of iterations to be equal to their maximum set amount. This would cause an unfair

comparison to the GA and therefore the values of average number of iterations have been left out for the GA. The GA values for maximum number of iterations can be obtained from the paper written on the GA.[6]

There are limitations when applying evolutionary algorithms to solve combinatorial circuit evolution of large scale circuits. As the size of circuits to evolve increase, the computational time for algorithms also increase. To represent a large scale circuit, the circuit structure size, from Sec. 2, must be increased. Increasing the circuit structure size will increase the search space. An increase in circuit structure size alone will increase computational time. To properly explore an expanded search space, there must be more iterations and particles in the population to make up the difference in search space sizes. Further, the increase in the number of particles and iterations increases the computational time.

4.1. *Case study 1*

The evolved combinational logic circuit for this simple example has three inputs and one output (the truth table is given in Table 1 with the output variable Y). Table 3 shows the results of circuit evolution with the different algorithms. "Min Gates" represents the minimum number of logic gates required to realize the truth table function (feasible circuit). These three algorithms were able to find the least amount of gates 100% of the time. This shows that the PSO portion of the code is able to obtain the solution on its own without any help, as well as the DE.

When the DEPSO evolves the circuit for the same truth table, the DE is unable to enhance the results of the PSO. This is due to the fact that the PSO has already converged to the solution as effectively and

Table 1. Truth table for case studies 1 and 2.

A	B	C	Y	Z
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

Table 2. Truth table for case studies 3 through 6.

A	B	C	D	W	X	Y	Z1	Z2
0	0	0	0	0	0	1	1	0
0	0	0	1	1	0	1	1	0
0	0	1	0	1	1	0	1	0
0	0	1	1	0	1	0	0	0
0	1	0	0	1	0	1	1	0
0	1	0	1	1	0	1	1	0
0	1	1	0	0	0	1	0	0
0	1	1	1	1	0	1	0	0
1	0	0	0	1	1	0	1	0
1	0	0	1	0	1	0	0	0
1	0	1	0	1	0	1	0	0
1	0	1	1	1	1	0	0	1
1	1	0	0	0	1	0	0	0
1	1	0	1	1	1	0	0	0
1	1	1	0	1	1	0	0	1
1	1	1	1	0	0	1	0	1

Table 3. Results from human design (HD), PSO, DE and hybrid DEPSO for case study 1.

Case Study 1				
Algorithm	HD	PSO	DE	DEPSO
Min gates	3	3	3	3
Equation	$\overline{B \otimes (A \otimes C)}$	$\overline{A \otimes (B \otimes C)}$	$(A \otimes C) \otimes \overline{B}$	$(\overline{A \otimes C}) \otimes B$
Types of gates	2 XOR, 1 NOT	2 XOR, 1 NOT	2 XOR, 1 NOT	2 XOR, 1 NOT
Percent of best solution	—	100%	100%	100%
Feasible circuits	—	100%	100%	100%
Avg fitness & # of gates	—	3.00 ± 0.00	3.00 ± 0.00	3.00 ± 0.00
Num of iterations	—	3.25 ± 2.79	5.60 ± 4.30	5.05 ± 3.33

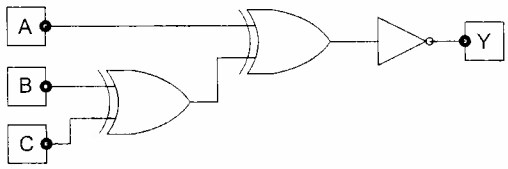

Fig. 9. Circuit design of case study 1 from PSO.

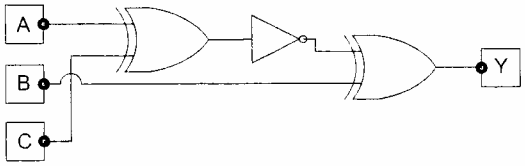

Fig. 10. Circuit design of case study 1 from DEPSO.

efficiently as possible, leaving no room for the DE to improve the solution. The PSO, DE, and DEPSO have all obtained unique solutions for this case study. The best logic circuits evolved using the PSO and the DEPSO algorithms are shown in Figs. 9 and 10, respectively.

4.2. *Case study 2*

The evolved combinational logic circuit for this simple example contains three inputs and one output (the truth table is given in Table 1 with the output variable Z). Table 4 shows the results of circuit evolution with the different algorithms. The results of comparing the PSO, DE, and DEPSO show the same results as the first case study. The only difference is in the performance of the DE. The DE is unable to reach the optimal gate circuit solution 100% of the time, unlike the PSO and DEPSO.

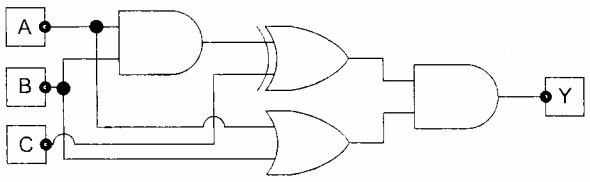

Fig. 11. Circuit design of case study 2 from PSO.

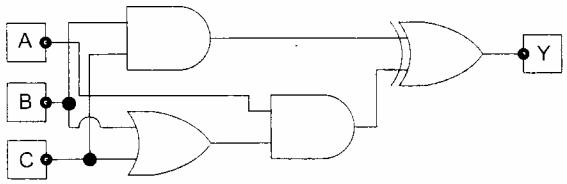

Fig. 12. Circuit design of case study 2 from DEPSO.

The complexities in reaching the best solution have hindered the DE's ability to perform as well as the PSO. This shows that the PSO and DEPSO are able to outperform the DE for this simple circuit example. The PSO, DE, and DEPSO have all obtained optimal circuit solutions unique to this case study. The minimum gate circuit solution for the PSO and the DEPSO are shown in Figs. 11 and 12, respectively.

4.3. *Case study 3*

The third case study is more complex than the first two case studies. This example has four inputs and one output with all possible inputs and outputs (variable W) given in Table 2. Table 5 shows the results of circuit evolution with the different algorithms. This

Table 4. Results from HD, PSO, DE and hybrid DEPSO for case study 2.

	Case Study 2			
Algorithm	HD	PSO	DE	DEPSO
Min gates	5	4	4	4
Equation	$C(A \oslash B)$ $+ B(A \otimes C)$	$(A + B)$ $\times ((A \bullet B) \oslash C)$	$[C \bullet (A - B)]$ $\oslash (A \bullet B)$	$(A \bullet (B + C))$ $\oslash (BC)$
Types of Gates	2 XOR, 2 AND, 1 OR	2 AND, 1 OR, 1 XOR	1 XOR, 2 AND, 1 OR	1 XOR, 2 AND, 1 OR
Percent of best solution		100%	65%	100%
Feasible circuits	—	100%	100%	100%
Avg fitness & # of gates		4.00 ± 0.00	4.35	4.00 ± 0.00
Num of iterations		228.10 ± 200.39	297.20	228.30 ± 116.50

Table 5. Results from HD, GA and PSO for case study 3.

	Case Study 3				
Algorithm	HD	DE	*GA	PSO	DEPSO
Min gates	9	9	6	6	6
Equation	$[\overline{(A+C)} \otimes \overline{(B+D)}] + [(A \bullet C) \otimes (B \bullet D)]$	$\{[(B \otimes D) + (A \otimes C)] \otimes [(B \otimes C) + (B \otimes D)]\} + \{[(B \otimes D) \bullet (A \otimes C)] \otimes [(B \otimes C) + (B \otimes D)]\}$	$(C \otimes D) \bullet (B \otimes C) + [(B \otimes A) \otimes (C \otimes D)]$	$[(A \otimes D) \otimes (B \otimes C)] \bullet [(A \otimes B) + (B \otimes C)]$	$[(B \otimes C) \bullet (A \otimes B)] + [(B \otimes C) \otimes (C \otimes D)]$
Types of gates	2 XOR, 2 AND, 3 OR, 2 NOT	5 XOR, 1 AND, 3 OR —	4 XOR, 1 AND, 1 OR	4 XOR, 1 OR, 1AND	4 XOR, 1 AND, 1 OR
Percent of best solution	—	5%	30%	100%	100%
Feasible circuits	—	15%	90%	100%	100%
Avg Fitness & # of gates	—	22.94	Not available	6.00 ± 0.00	6.00 ± 0.00
Num of iterations	—	875.95	Not available	878.85 ± 504.23	442.60 ± 220.36

Note: * The results for the GA algorithm were obtained from Coello.[6] The fitness evaluation of GA is a maximizing function, thus a comparison on average fitness against a minimizing fitness function is not possible. The average number of iterations taken by the GA algorithm is not represented.

is the first case study where DEPSO improves over the DE and the PSO. This is also the first case study in which DE diverges from finding the minimum gate circuit solution. The DEPSO and PSO find the best solution 100% of the time, but DEPSO is able to find the solution nearly half the amount of time of the PSO. When the DE is added to the PSO, the DEPSO is able to reach maximum efficiency and effectiveness to solve the problem in around half the number of iterations of the PSO. The DE algorithm is unable to find the minimum gate circuit solution on its own, but it is able to help the PSO algorithm perform better. As the complexity of the circuits increase, the DE is unable to keep up with the performances of the PSO and the DEPSO. For this reason, the DE has been left out of the comparisons for the next three case studies.

For hardware evolution on combinational logic circuits, the DE is excellent for aiding other algorithms, but not for evolving hardware on its own. Since there are no results for average fitness value or number of iterations, the GA must be compared with respect to percent of feasible circuits. If the values are the same for percent of feasible circuits, then the GA must be compared with respect to percent of best solutions. This will be the means of evaluation for all of the GA comparisons in this case study and the case studies that follow. In this case study, the GA was unable to obtain the 100% convergence of feasible circuits. The GA was only able to outperform the DE algorithm in this case study. The PSO and DEPSO were able to find different solutions to the minimum gate circuit solution, whereas the DE was unable to find the best solution.

The minimum gate circuit design for the GA, PSO, and DEPSO are shown in Figs. 13, 14, and 15, respectively.

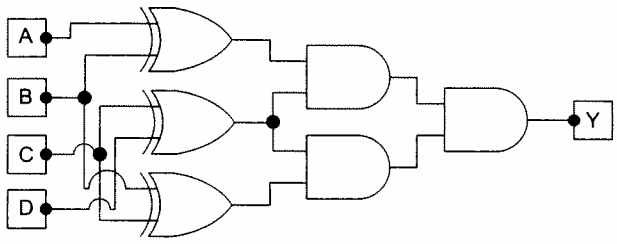

Fig. 13. Circuit design of case study 3 from GA.

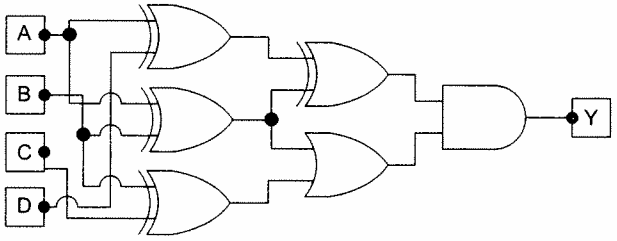

Fig. 14. Circuit design of case study 3 from PSO.

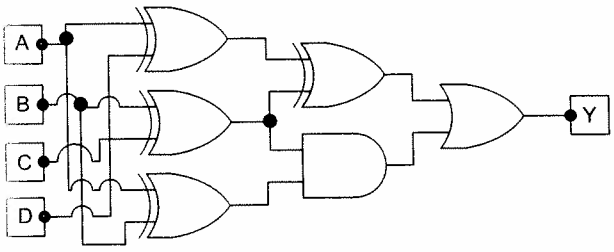

Fig. 15. Circuit design of case study 3 from DEPSO.

4.4. *Case study 4*

This example was another complex circuit that involved 4 inputs and 1 output (variable X). The truth table is shown in Table 2. Table 6 shows the results of circuit evolution with the different algorithms. This is the first case study showing a direct improvement. In the previous case studies, the PSO was able maintain the minimal average number of gates along with the DEPSO. With this more complex circuit, the PSO was unable to achieve the minimal average number of gates along with the DEPSO. The DEPSO, however, was able to obtain a lower average number of gates than the PSO. Not only did the DEPSO have a lower average number of gates, but the DEPSO also had a fewer number of iterations in which it reached its best solutions. The minimum number of average gates is equal to five for this case study even though the standard deviation yields results that could be below the minimum number of gates found for the respective algorithm. Therefore there are no results that can be obtained below the value of the minimum number of gates obtained for each algorithm. This will be true for each additional case study as well. The GA was unable to do as well as the PSO and DEPSO algorithms in this case study as well. The GA was only able to obtain the best solution of five gates 70% of the time. The results of the GA,[6] PSO, and DEPSO yield two different circuit designs of the minimum gate circuit solution. The circuits evolved are shown in Figs. 16, 17 and 18 respectively.

4.5. *Case study 5*

This case study is another complex circuit with four inputs and one output (variable Y). The truth table for this case study is given in Table 2. Table 7 shows the results of circuit evolution with the different algorithms. The results of case study 5 comply with the results from the previous case study. In this case the PSO was unable to obtain the minimal average number of gates, but the DEPSO was able to. Again the DEPSO's performance is shown to be better than the PSO. Not only did the DEPSO have a lower average number of gates, but it also found its best solutions in a lower number of iterations than that of the PSO on average. The PSO and DEPSO have obtained unique solutions in evolving hardware to obtain a minimum gate circuit solution. The minimum gate circuit

Table 6. Results from HD and GA for case study 4.

Case Study 4				
Algorithm	HD	*GA	PSO	DEPSO
Min gates	11	5	5	5
Equation	$[(\overline{B} \bullet C) \bullet (\overline{A} + D)]$ $+ [(\overline{A} + D + \overline{B}) + (\overline{A} + C)]$	$(C \bullet B)$ $\otimes [(D \bullet A) + (B \otimes D)]$	$[(A \bullet D) + (A \otimes c)]$ $\otimes (B \bullet C)$	$[(A \otimes C) + (C \bullet D)]$ $\otimes (B \bullet C)$
Types of gates	2 AND, 5 OR, 4 NOT	2 XOR, 2 AND, 1 OR	2 XOR, 1 OR, 2 AND	2 XOR, 2 AND, 1 OR
Percent of best solution	—	10%	60%	90%
Feasible circuits	—	70%	100%	100%
Avg fitness & # of gates	—	Not available	5.90 ± 1.25	5.10 ± 0.31
Num of iterations	—	Not available	1032.40 ± 411.27	956.55 ± 592.19

Note: *The results for the GA algorithm were obtained from Coello.[6] The fitness evaluation of GA is a maximizing function, thus a comparison on average fitness against a minimizing fitness function is not possible. The average number of iterations taken by the GA algorithm is not represented.

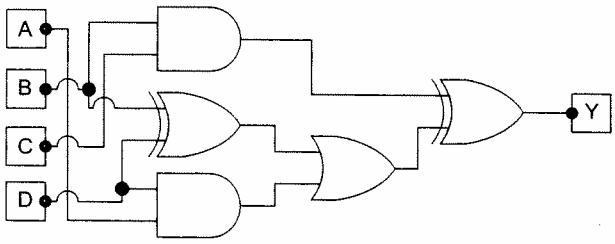

Fig. 16. Circuit design of case study 4 from GA.

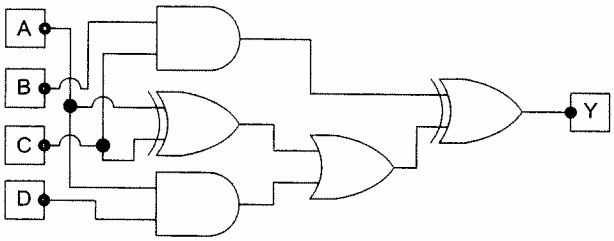

Fig. 17. Circuit design of case study 4 from PSO.

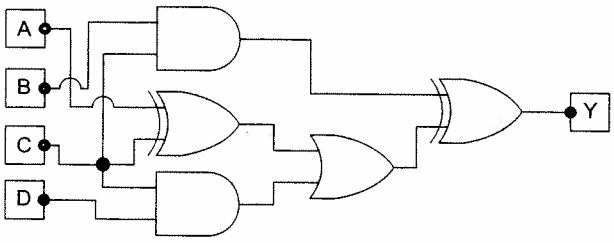

Fig. 18. Circuit design of case study 4 from DEPSO.

evolved using the PSO and DEPSO algorithms are shown in Figs. 19 and 20 respectively.

4.6. *Case study 6*

In this example, the complexity of the circuit increased to four inputs and two outputs (variables Z1 and Z2). The truth table for this case study is given in Table 2. Table 8 shows the results of circuit evolution with the different algorithms. Again, the DEPSO was able to obtain a lower average number of gates than the PSO. As the circuit complexity increases, the DEPSO is able to increase its performance over the PSO. In previous case studies, the PSO was able to keep its average number of gates to within one gate to that of the DEPSO. In this case study, the DEPSO has circuit designs of, on average, two gates less than that of the PSO. The DEPSO again has less number of iterations, on average, to reach the minimal gate solutions than that of the PSO. The GA, PSO, and the DEPSO all obtained unique optimal solutions as shown in Figs. 21, 22, and 23 respectively.

5. Discussion

Usually, the velocity is encoded with continuous numbers and is analyzed discretely in the canonical discrete PSO algorithm. In this paper, the velocity is encoded with discrete numbers and is also analyzed discretely. The first reason for doing is for the future implementation of the PSO based algorithm on a digital processor. In order to implement

Table 7. Results from HD, PSO, and hybrid DEPSO for case study 5.

	Case Study 5		
Algorithm	HD	PSO	DEPSO
Min Gates	12	6	6
Equation	$\overline{[(A \bullet \overline{C}) + (A \bullet B \bullet \overline{D})]} \bullet [(B + \overline{C} + \overline{D}) \bullet (A + B + \overline{C})]$	$[\overline{C} + (B \otimes D)] \otimes [(C \bullet D) + A]$	$[(A \otimes C) + (C \bullet D)] \otimes \overline{(B \bullet C)}$
Types of gates	5 AND, 4 OR, 3 NOT	2 XOR, 2 OR, 1 AND, 1 NOT	2 XOR, 2 AND, 1 OR, 1 NOT
Percent of best solution	—	70%	100%
Feasible circuits	—	100%	100%
Avg fitness & # of gates	—	6.55 ±1.00	6 ± 0
Num of iterations		1187.00 ± 520.32	891.05 ± 530.16

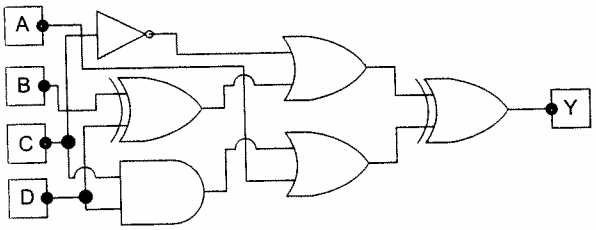

Fig. 19. Circuit design for case study 5 from PSO.

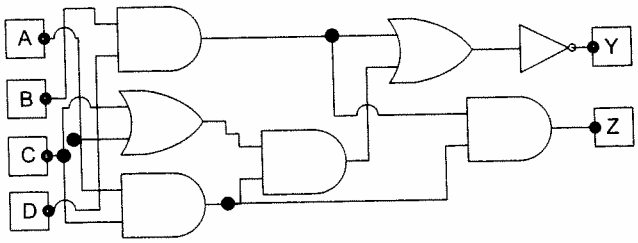

Fig. 21. Circuit design for case study 6 from GA.

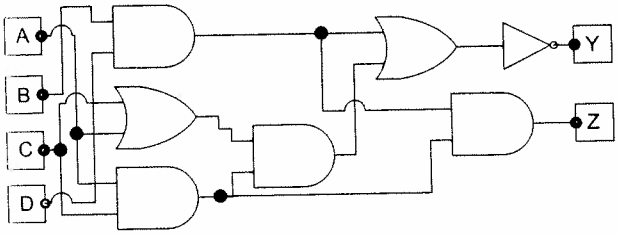

Fig. 20. Circuit design for case study 5 from DEPSO.

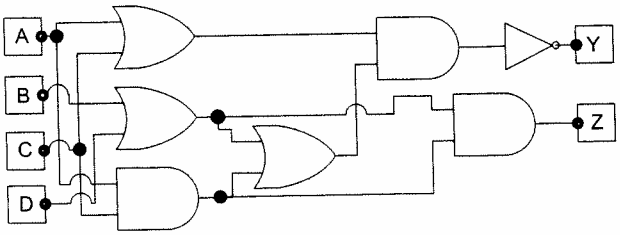

Fig. 22. Circuit design for case study 6 from PSO.

the PSO on a digital processor, it is cost efficient to use purely discrete numbers. Floating point encoded values that are stored on a digital processor takes up unnecessary memory space in memory to represent floating point values.

The second reason for encoding the velocity to be purely discrete is to include *acceleration*, which is not as noticeable in continuous encoded velocity update equations. The a*cceleration* is the sudden increase and decrease in a velocity that acts as an

Table 8. Results from HD and GA for case study 6.

	Case Study 6			
Algorithm	HD	*GA	PSO	DEPSO
Min Gates	8	7	7	7
Equation First Output	$[(\overline{B+D}) \bullet (A+C)] + \overline{(A+C)}$	$\overline{(C \bullet A) \bullet [(B+D)+(B \bullet D)]}$	$\overline{(A+C)} \bullet \overline{[(B+D)+(A \bullet C)]}$	$\overline{[(B+D) \bullet (A+C)]} + \overline{(A \bullet C)}$
Second Output	$(A \bullet C) \bullet (B+D)$	$(C \bullet A) \bullet (B+D) \bullet (B \bullet D)$	$(B+D) \bullet (A \bullet C)$	$[(B+D) \bullet (A+C)] \bullet (A \bullet C)$
Types of gates	3 AND, 3 OR, 2 NOT	4 AND, 2 OR, 1 NOT	3 OR, 3 AND, 1 NOT	3 AND, 3 OR, 1 NOT
Percent of best solution	—	25%	40%	20%
Feasible circuits	—	75%	95%	100%
Avg fitness & # of gates	—	Not available	10.50 ± 9.07	8.60 ± 1.26
Num of iterations	—	Not available	3612.80 ± 1066.10	3607.00 ± 1175.50

Note: *The results for the GA algorithm were obtained from Coello.[6] The fitness evaluation of GA is a maximizing function, thus a comparison on average fitness against a minimizing fitness function is not possible. The average number of iterations taken by the GA algorithm is not represented.

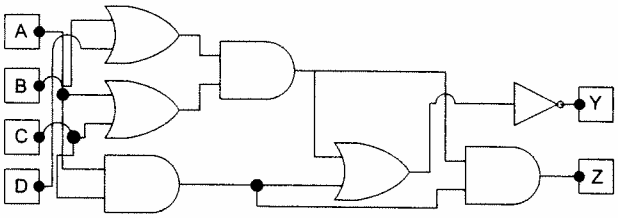

Fig. 23. Circuit design for case study 6 from DEPSO.

expansion of what the velocity of a particle can represent. The velocity value for a particle can either represent the floor or ceil values of the original velocity value. The decimal value of the original velocity is used as a means of determining whether or not the velocity shall represent the floor or ceil of the original velocity value.

A tradeoff for including acceleration, using a purely discrete velocity by means of the round function, in the velocity of a PSO is that the velocity jumps may skip over better solutions. To compensate for this tradeoff, as previously explained, the numbers representing the five gates of the gate library has been increased to twenty. Increasing the numbers representing the gates from the gate library will bring the gate representation, for the five gates, closer to a continuous representation. Compensating for the trade allows for the velocity to expand what the velocity of a particle can represent while at the same time limiting velocity jumps over better solutions.

6. Conclusion

This paper has presented a new hybrid algorithm called the Differential Evolution Particle Swarm Optimization (DEPSO) based on the combined concepts of a modified PSO and DE for evolving combinational logic circuits. The paper has also presented modifications necessary to enhance performance and robustness on discrete optimization problems such as evolvable hardware when applying swarm and evolutionary techniques.

The DE and modified PSO algorithms performed very well on the simple digital circuits. However, the performance of the DE algorithm rapidly declines as the complexity of the circuit function increases slightly. The DE and PSO have complementary characteristics that enable them to enhance their combined performance in the DEPSO algorithm; the DE is able to help more of the population reach the minimum gate solution and the PSO is able to help small numbers of the population excel at reaching their minimal gate circuit solution.

Increasing the number of inputs and outputs in a circuit increases its complexity. When it comes to more complex circuits, the PSO and the DE algorithms do not perform well individually. The combination of DE and PSO is able to evolve complex minimal gate circuit solutions more effectively. Overall, DEPSO has two main advantages over the other algorithms: i) The results in feasible circuits are always 100%, and ii) The average fitness and number of gates is always lower.

It is interesting to note that the evolutionary strategies in process of evolving combinational circuits using PSO, DE, and DEPSO arrive at many unique solutions for a given function unlike when using the conventional textbook methods such as Karnaugh maps. This means that such techniques have potential applications for fault tolerant design based on redundancy.

Acknowledgment

The authors gratefully acknowledge the support from the National Science Foundation CAREER grant ECS # 0348221.

References

1. M. Karnaugh, The map method for synthesis of combinational logic circuits, *AIEE Trans.* **72**(9) (1953) 593–599.
2. W. V. Quine, A way to simplify truth functions, *American Mathematical Monthly* **62**(9) (1955) 627–631.
3. E. J. McCluskey, Minimization of Boolean functions, *Bell Systems Technical Journal* **35**(5) (1956) 1417–1444.
4. C. A. Coello, A. D. Christiansen and A. A. Hernández, Automated design of combinational logic circuits using genetic algorithms, in *Proc. Intl. Conf. on Artificial Neural Nets and Genetic Algorithms*, (University of East Anglia, Norwich, England), (1997) pp. 335–338.
5. J. F. Miller, D. Job and V. K. Vassiley, Principles in the evolutionary design of digital circuits, *Journal of Genetic Programming and Evolvable Machines* **1**(1) (2000) 8–35.
6. C. A. Coello, E. H. Luna and A. H. Aguirre, A comparative study of encodings to design combinational logic circuits using particle swarm optimization, in *NASA/DoD Conference on Evolvable Hardware*, (Seattle, Washington) (2004) pp. 71–78.

7. V. G. Gudise and G. K. Venayagamoorthy, Evolving digital circuits using particle swarm, in *Proc. of the INNS-IEEE Intl. Joint Conf. on Neural Networks*, (Portland, OR) (2003) pp. 468–472.
8. E. H. Luna, C. A. Coello and A. H. Aguirre, On the use of a population-based particle swarm optimizer to design combinational logic circuits, in *NASA/DoD Conference*, (Seattle, WA, USA) (2004) pp. 183–190.
9. R. C. Eberhart and J. Kennedy, A new optimizer using particles swarm theory, in *Proc. Sixth Intl. Symposium on Micro Machine and Human Science*, (Nagoya, Japan) (IEEE Service Center, Piscataway, NJ) (1995) pp. 39–43.
10. X. Hu, R. C. Eberhart and Y. Shi, Engineering optimization with particle swarm, in *IEEE Swarm Intelligence Symposium*, (Indianapolis, IN, USA) (2003) pp. 53–57.
11. J. Kennedy and R. C. Eberhart, A discrete binary version of the particle swarm algorithm, in *IEEE Conf. on Systems, Man and Cybernetics*, (Orlando, FL, USA) **5** (1997) 4104–4108.
12. R. Storn and K. Price, Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces, *Technical Report*, (Berkley, California) (1995) TR-95-012.
13. V. Feoktistov and S. Janaqi, Generalization of the strategies in differential evolution, in *IEEE 18th Intl. Parallel and Distributed Processing Symposium (IPDPS'04)* - Workshop 6, (Sante Fe, New Mexico) (2004) p. 165.
14. K. V. Price, *An Introduction to Differential Evolution, New Ideas in Optimization.* (McGraw-Hill, London, UK) ISBN 007-709506-5 pp. 79–108.
15. W. Zhang and X. Xie, DEPSO: Hybrid particle swarm with differential evolution operator, in *IEEE Intl. Conf. on Systems, Man and Cybernetics*, **4** (2003) 3816–3821.
16. J. Lampinen, Differential evolution handling integer and discrete valued variables, in *Proc. of MENDEL 2001, 7th Intl. Conf. on Soft Computing*, (Brno, Czech Republic) (2001) 50–57.
17. C. A. Coello and A. H. Aguirre, Design of combinational logic circuits through an evolutionary multiobjective optimization approach, *Artificial Intelligence for Engineering, Design, Analysis and Manufacture* **16**(1) (2002) 39–53.
18. P. W. Moore and G. K. Venayagamoorthy, Evolving digital circuits using hybrid particle swarm optimization and differential evolution, in *Conf. on Neuro-Computing and Evolving Intelligence*, (Auckland, New Zealand) (2004) pp. 71–73.
19. V. G. Gudise and G. K. Venayagamoorthy, Evolving digital circuits using particle swarm, in *Proc. of the INNS-IEEE Intl. Joint Conf. on Neural Networks*, (Portland, OR) (2003) pp. 468–472.
20. V. G. Gudise and G. K. Venayagamoorthy, FPGA placement and routing using particle swarm optimization, in *IEEE Computer Society Annual Symposium on VLSI Emerging Trends in VLSI Systems Design*, (Lafayette, Louisiana) (2004) pp. 307–308.
21. C. A. Coello, A. H. Aguirre and B. P. Buckles, Evolutionary multiobjective design of combinational logic circuits, in *The Second NASA/DoD Workshop on Evolvable Hardware*, (Palo Alto, California) (2000) pp. 161–170.