On the Use of a Population-Based Particle Swarm Optimizer to Design Combinational Logic Circuits

Erika Hernández Luna, Carlos A. Coello Coello
CINVESTAV-IPN
Evolutionary Computation Group
Dpto. de Ing. Elect./Secc. Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco
México, D.F. 07300, MEXICO
eluna@computacion.cs.cinvestav.mx
ccoello@cs.cinvestav.mx

Arturo Hernández Aguirre
Center for Research in Mathematics (CIMAT)
Department of Computer Science
Guanajuato, Gto. 36240, MEXICO
artha@cimat.mx

Abstract

In this paper, we introduce the use of a population-based selection scheme in a particle swarm optimizer used for designing combinational logic circuits. The scheme aims to distribute the search effort in a better way within the particles of the population as to accelerate convergence while improving the robustness of the algorithm. For our study, we compare six PSO-based approaches, combining different encodings (integer and binary) with both single- and multi-objective selection schemes. The comparative study performed indicates that the use of a population-based approach combined with an integer encoding improves both the robustness and quality of results of PSO when designing combinational logic circuits.

1 Introduction

The Particle Swarm Optimization (PSO) algorithm is a biologically-inspired technique originally proposed by Kennedy and Eberhart [7, 8]. PSO has been successfully used as a (nonlinear) optimization technique and has become increasingly popular due to its combination of simplicity (in terms of its implementation), low computational cost and good performance [8]. These are precisely the main motivations that led us to apply PSO for combinational circuit design [4].

The main idea behind PSO is to simulate the movement of a flock of birds seeking food. In this simulation, the behavior of each individual gets affected by both an individual and a social factor. Each individual (or particle) contains its current position in the search space as well as its velocity and the best position found by the individual so far [8]. As many other biologically-inspired heuristics, PSO is a population-based approach that can be defined as $P' = (m(f(P))$, where P is the *population*, which consists of a set of positions in search space, f is the *fitness function*, that returns a vector of values that indicate the goodness of each individual, and m is a manipulation function that generates a new population from the current population. Such a manipulation function is based on the behavioral model of insect colonies [1].

PSO can be seen as a distributed behavioral algorithm that performs (in its more general version) multidimensional search. In the simulation, the behavior of each individual is affected by either the best local (i.e., within a certain neighborhood) or the best global individual. The approach uses a population of potential solutions (called "particles") and a measure of performance similar to the fitness value used with evolutionary algorithms. Also, the adjustments of individuals are analogous to the use of a crossover operator. However, this approach introduces the use of flying potential solutions through hyperspace (used to accelerate convergence). Additionally, PSO allows individuals to benefit from their past experiences [8].

In this paper, we propose the use of a multiobjective optimization technique to design combinational circuits. Our approach (which uses PSO as its optimization engine) is based on some of our previous research in which a population-based genetic algorithm was used to design combinational logic circuits [3]. The proposal consists of handling each of the matches between a solution generated by our PSO approach and the values specified by the truth table as equality constraints. To avoid the dimensionality problems associated with conventional multiobjective optimization techniques (such problems are due to the fact that checking for Pareto dominance is an $O(n^2)$ process), we use a population-based approach similar to the Vector Evaluated Genetic Algorithm (VEGA) [11].

2 Problem Statement

The problem of interest to us consists of designing a circuit that performs a desired function (specified by a truth table), given a certain specified set of available logic gates. The complexity of a logic circuit is a function of the number of gates in the circuit. The complexity of a gate generally is a function of the number of inputs to it. Because a logic circuit is a realization (implementation) of a Boolean function in hardware, reducing the number of literals in the function should reduce the number of inputs to each gate and the number of gates in the circuit—thus reducing the complexity of the circuit. Our overall measure of circuit optimality is the total number of gates used, regardless of their kind. This is approximately proportional to the total part cost of the circuit. Obviously, we perform this analysis for only fully functional circuits.

3 Our Proposed Approach

We used a matrix to represent a circuit as in our previous work [2], as shown in Figure 1. More formally, we can say that any circuit can be represented as a bidimensional array of gates $S_{i,j}$, where j indicates the *level* of a gate, so that those gates closer to the inputs have lower values of j. (Level values are incremented from left to right in Figure 1). For a fixed j, the index i varies with respect to the gates that are "next" to each other in the circuit, but without being necessarily connected. Each matrix element is a gate (there are 5 types of gates: AND, NOT, OR, XOR and WIRE[1]) that receives its 2 inputs from any gate at the previous column as shown in Figure 1. This sort of encoding was originally proposed by Louis [9]. The so-called "cartesian genetic programming" [10] also adopts a similar encoding to the matrix previously described.

Using the aforementioned matrix, a logic circuit can be encoded using either binary or integer strings that correspond to the type of gate adopted and its inputs. PSO, however, tends to deal with either binary or real-numbers representation. For our comparative study, we will adopt two integer representations: (1) **Integer A** (proposed by Hu et al. [6]), and (2) **Integer B** (proposed by us).

In the PSO algorithm, the individual factor P_{best} refers to the decisions that the individual has made so far and that have worked best (in terms of its performance measure). This value has an impact on its future decisions. Additionally, the social factor N_{best} refers to the decisions that the other individuals (within a certain neighborhood) have made so far and that have worked best for them. This value will also affect the future decisions of the individuals in the given neighborhood.

[1]WIRE basically indicates a null operation, or in other words, the absence of gate, and it is used just to keep regularity in the representation used by our approach that otherwise would have to use variable-length strings.

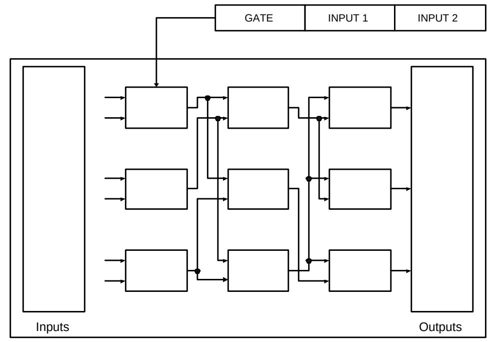


Figure 1. Encoding used for each of the matrix elements that represent a circuit.

Figure 2 shows the pseudocode of the PSO algorithm that we propose for the design of combinational logic circuits. Its main difference with respect to traditional PSO has to do with the update of the position of the particle in each of its dimensions. Both, the **Integer A** and the **Integer B** approaches normalize the velocity of each dimension of the particle in the range 0 to 1, so that we can further determine (in a random way) whether we need to change the current position or not (this is done with the probability given by the velocity). If the change is required, then we copy to the particle the value of N_{best} in the current position. Otherwise, the **Integer A** approach leaves the particle intact. When the change is not required, the **Integer B** approach checks again whether is necessary to change the current position, but now using a probability of $1 - v_d$, where v_d is the current velocity. If the change is required, then we copy to the particle the value of P_{best} in the position that we are updating. Otherwise, we leave the particle intact. These two integer representations are exemplified in Figure 3. As in our previous work [4], we introduce here a mutation operator to our PSO algorithm in order to improve its exploratory power, since this seems necessary when applying this approach to the design of circuits. Furthermore, in this case, the particles try to follow the same characteristics of N_{best} and P_{best} and could get stuck in their current position. Thus, the use of a mutation operator is vital in order to avoid this problem.

4 Use of a Multiobjective Approach

The objective function in our case is defined as in previous work [2]: it is the total number of matches (between the outputs produced by an encoded circuit and the intented val-

Randomly initialize the population of particles, P.
Repeat {
  For each particle i in the population P {
    Compute the fitness of the particle $P[i]$
    If the fitness of $P[i]$ is better than the fitness of the best particle found so far $P_{best}[i]$,
    Then update P_{best} using $P[i]$.
  }
  For each particle i in P {
    Select the particle with the best fitness in the topological neighborhood of $P[i]$
    and update the value of $N_{best}[i]$
  }
  For each particle i in the population P {
    Compute the new velocity for each dimension of the particle
    $\vec{V}[i]_{new} = \vec{V}[i]_{old} + \phi_1(\vec{P}best[i] - \vec{P}[i]) + \phi_2(\vec{N}best[i] - \vec{P}[i])$
    Update the position of the particle $P[i]$
  }
  Apply uniform mutation with a (user given) rate.
} **Until** reaching the stop condition

Figure 2. Pseudocode of the PSO algorithm adopted in this work. Note the addition of a mutation operator.

ues defined by the user in the truth table). For each match, we increase the value of the objective function by one. If the encoded circuit is feasible (i.e., it matches the truth table completely), then we add one (the so-called "bonus") for each WIRE present in the solution. Note however, that in this case, we use a multiobjective approach to assign fitness. The main idea behind our proposed approach is to use a population-based multiobjective optimization technique similar to VEGA [11] to handle each of the outputs of a circuit as an objective. In other words, we would have an optimization problem with m equality constraints, where m is the number of values (i.e., outputs) of the truth table that we aim to match. So, for example, a circuit with 3 inputs and a single output, would have $m = 2^3 = 8$ values to match. At each generation, the population is split into $m + 1$ sub-populations, where m is defined as indicated before (we have to add one to consider also the objective function). Each sub-population optimizes a separate constraint (in this case, an output of the circuit). Therefore, the main mission of each sub-population is to match its corresponding output with the value indicated by the user in the truth table. The main issue here is how to handle the different situations that could arise. Our proposal is the following:

if $o_j(\mathbf{X}) \neq t_j$	**then**	fitness($\mathbf{X}$) = 0
else if $v \neq 0$ AND $x \in R$	**then**	fitness = $-v$
else		fitness = $f(\mathbf{X})$

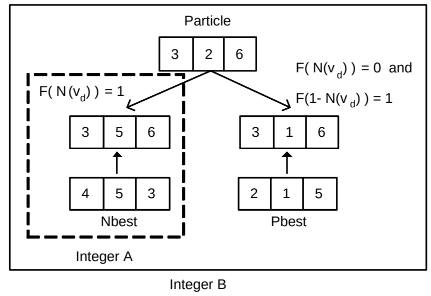


Figure 3. Example of the two integer representations used for our PSO algorithm.

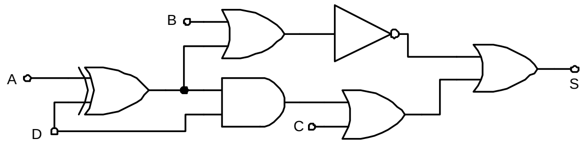
where $o_j(\mathbf{X})$ refers to the value of output j for the encoded circuit $\mathbf{X}$; t_j is the value specified for output j in the truth table; v is the number of outputs that are not matched by the circuit $\mathbf{X}$ ($\leq m$); and R is the subpopulation whose objective is to match all the output values from the truth table. Finally, $f(\mathbf{X})$ is the fitness function defined as:

$$f(\mathbf{X}) = \begin{cases} h(\mathbf{X}) & \text{if } \mathbf{X} \text{ is infeasible} \\ h(\mathbf{X}) + w(\mathbf{X}) & \text{otherwise} \end{cases} \quad (1)$$

In this equation, $h(\mathbf{X})$ refers to the number of matches between the circuit $\mathbf{X}$ and the values defined in the truth table, and $w(\mathbf{X})$ is the number of WIREs in the circuit $\mathbf{X}$. As can be seen, the scheme adopted in this work is slightly different from the one used by our MGA reported in [3]. The main reason for adopting this approach is that in our experiments, it produced more competitive results, improving in most cases the results obtained with our single-objective PSO, as we will see in the next section.

5 Comparison of Results

The truth tables used to validate our PSO approach were taken from the specialized literature. In our experimental study, we compared the following approaches: a binary multiobjective PSO approach (BMPSO), a PSO approach using an integer A encoding (EAMPSO), a PSO approach using an integer B encoding (EBPSO), a binary single-objective PSO (BPSO), a single-objective PSO approach using integer A encoding (EAPSO), a single-objective PSO approach using integer B encoding (EBPSO) and the multi-objective genetic algorithm for circuit design (MGA) [3]. For each of the examples shown, we performed 20 independent runs, and the available set of gates considered was the following: AND, OR, NOT, XOR and WIRE. We used a



$$S = (B + (D \oplus A))' + (C \oplus D(D \oplus A))$$

Figure 4. Diagram and boolean expression corresponding to the best solution found for example 1.

matrix of size 5×5 in all cases, except for the second example for which a 6×6 matrix was adopted. The parameters adopted by both BPSO and BMPSO were the following: $\phi_1 = \phi_2 = 0.8$, $V_{max} = 3.0$, mutation rate $P_m = 0.1$ and neighborhood size $= 3$. EAPSO, EAMPSO, EBPSO and EBMPSO used: $\phi_1 = \phi_2 = 0.2$, $V_{max} = 0.4$, $P_m = 0.1$ and neighborhood size $= 3$. The MGA used $P_m = 0.00667$ and a crossover rate $= 0.5$ (as suggested in [3]).

5.1 Example 1

Our first example has 4 inputs and 1 output, as shown in Table 1. The additional parameters adopted are shown in Table 2. Note that we attempted to perform the same number of fitness function evaluations with all the approaches compared. In Table 3, we show a comparison of the results of all the approaches adopted. The best solution found for this example has 6 gates and is graphically shown in Figure 4. Note that both BMPSO and EBMPSO were able to find a circuit that uses one gate less than their single-objective counterparts (i.e., BPSO and EBPSO). Nevertheless, the average fitness of both BMPSO and EBMPSO were lower than the values of their single-objective counterparts. Also note that although EAMPSO was not able to improve the solutions obtained by EAPSO, its percentage of feasible circuits increased from 65% to 85%. Also, the average fitness of EAMPSO was 30.25 compared to the 26.75 value produced by EAPSO. In this example, the MGA did not perform too well when compared with any of our PSO versions. Its percetange of feasible circuits was low (35%) and it was not able to find the solution with only 6 gates produced by some of the PSO approaches. Another interesting fact was that EBPSO had the best average fitness (31.2), but was not able to produce circuits with 6 gates. EAMPSO, in contrast, had the second best average fitness (30.25), but was able to find circuits with only 6 gates 5% of the time. Thus, EAMPSO can be considered as the best overall performer in this example.

D	C	B	A	S
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

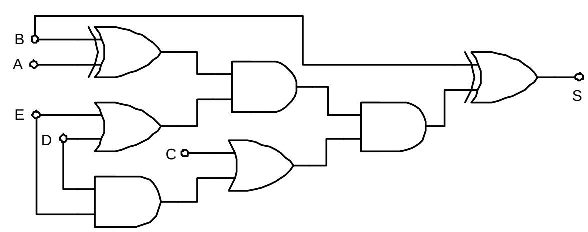
Table 1. Truth table for example 1.

Technique	Population size	Iterations	FFE
MPSO	68	1,471	100,028
PSO	50	2,000	100,000
MGA	170	600	102,000

Table 2. Parameters adopted for example 1. FFE = Fitness function evaluations.

approach	gates b.s.	freq. b.s.	feas. circs.	avg.# gates	avg. fitn.	Std. dev.
BMPSO	8	5%	20%	22.8	18.2	6.622
EAMPSO	6	5%	85%	10.75	30.25	6.680
EBMPSO	6	5%	75%	12.75	28.25	7.953
BPSO	9	15%	45%	19.1	21.9	7.887
EAPSO	6	5%	65%	14.25	26.75	8.902
EBPSO	7	30%	90%	9.8	31.2	5.616
MGA	7	15%	35%	19.95	21.05	8.929

Table 3. Comparison of the results obtained by our multiobjective versions of PSO, our single-objective PSO versions, MGA and a human designer for the first example. b.s.=best solution.



$$S = (((E + D)(B \oplus A))(C + (ED))) \oplus B$$

Figure 5. Diagram and boolean expression corresponding to the best solution found for example 2.

E	D	C	B	A	S
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	1	1

Table 4. Truth table for example 2.

5.2 Example 2

Our second example has 5 inputs and 1 output, as shown in Table 4. The additional parameters adopted are shown in Table 5. In Table 6, we show a comparison of the results of all the approaches adopted. The best solution found for this example has 7 gates and is graphically shown in Figure 5. In this case, none of the binary versions of PSO was able to produce feasible circuits, which exemplifies the usefulness of adopting integer encodings in PSO. There were mixed results for the other approaches. Both EAMPSO and EAPSO found the best solution with the same frequency (15%), but EAMPSO found feasible circuits 40% of the time (versus 35% of EAPSO). In terms of average fitness both EAMPSO and EAPSO had similar results (41.7 vs. 40.45). Thus, we can conclude that EAMPSO was the best overall performer in this example. Interestingly, EBPSO had both the highest average fitness (41.9) and the highest percentage of feasible circuits (45%), but was not able to find a circuit with 7 gates. The MGA was able to find circuits with 7 gates, but both its percentage of feasible circuits (20%) and its average fitness (36) were low in comparison with the multi-objective PSO approaches.

Technique	Population size	Iterations	FFE
MPSO	99	20,000	1,980,000
PSO	50	39,600	1,980,000
MGA	330	6,000	1,980,000

Table 5. Parameters adopted for example 2. FFE = Fitness function evaluations.

approach	gates b.s.	freq. b.s.	feas. circs.	avg.# gates	avg. fitn.	std. dev.
BMPSO	*	0%	0%	*	29.8	0.410
EAMPSO	7	15%	40%	26.3	41.7	14.543
EBMPSO	7	5%	20%	32.25	35.75	11.461
BPSO	*	0%	0%	*	29.9	0.308
EAPSO	7	15%	35%	27.55	40.45	14.529
EBPSO	8	20%	45%	26.1	41.9	13.619
MGA	8	5%	20%	38	36	13.322

Table 6. Comparison of the results obtained by our multiobjective versions of PSO, our single-objective PSO versions, MGA and a human designer for the second example. b.s.=best solution.

5.3 Example 3

Our third example has 4 inputs and 2 outputs as shown in Table 7. The additional parameters adopted are shown in Table 8. In Table 9, we show a comparison of the results of all the approaches adopted. The best solution found for this example has 7 gates and is graphically shown in Figure 6. In this case, BPSO produced considerably better results than its multi-objective counterpart (BMPSO) both in terms of average fitness (46.95 vs. 38.60) and in terms of percentage of feasible circuits produced (95% vs. 50%). EAMPSO, however, was able to considerably improve the results produced by its single-objective counterpart (EAPSO) also in

D	C	B	A	S_0	S_1
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	1
1	1	1	1	0	1

Table 7. Truth table for example 3.

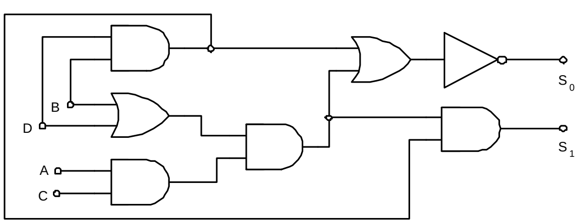
Technique	Population size	Iterations	FFE
MPSO	99	2,000	198,000
PSO	50	4,000	200,000
MGA	330	610	201,300

Table 8. Parameters adopted for example 3. FFE = Fitness function evaluations.

approach	gates b.s.	freq. b.s.	feas. circs.	avg. # gates	avg. fitn.	std. dev.
BMPSO	7	10%	50%	18.4	38.6	8.210
EAMPSO	7	65%	100%	7.75	49.25	1.333
EBMPSO	7	90%	100%	7.15	49.85	0.489
BPSO	7	30%	95%	10.05	46.95	4.330
EAPSO	7	40%	70%	13.45	43.55	8.530
EBPSO	7	60%	100%	7.75	49.25	1.160
MGA	7	25%	75%	13.4	43.6	8.090

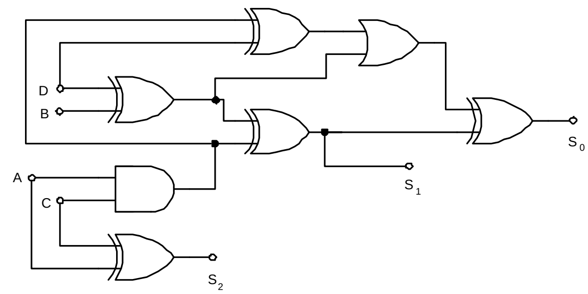
Table 9. Comparison of the results obtained by our multiobjective versions of PSO, our single-objective PSO versions, MGA and a human designer for the third example. b.s.=best solution.

terms of both average fitness (49.25 vs. 43.55) and percentage of feasible circuits produced (100% vs. 70%). Note that both EBMPSO and EBPSO were able to find feasible circuits in all their runs and had similar average fitnesses (49.85 vs. 49.25), but the former converged more often to the best solution found (90% vs. 60%). In fact, EBMPSO was the best overall performer in this example. Again, the MGA had a poor performance with respect to the PSO-based multi-objective approaches (EAMPSO and EBMPSO), although it had a better average fitness than both BMPSO and EAPSO and was also able to find the circuit of 7 gates generated by the PSO-based approaches.



$$S_0 = ((CA)(B+D) + BD)'$$
$$S_1 = (CA)(B+D)(BD)$$

Figure 6. Diagram and boolean expression corresponding to the best solution found for example 3.



$$S_0 = (AC \oplus (B \oplus D)) \oplus ((D \oplus AC) + (B \oplus D))$$
$$S_1 = AC \oplus (B \oplus D);\ S_2 = C \oplus A$$

Figure 7. Diagram and boolean expression corresponding to the best solution found for example 4.

5.4 Example 4

Our fourth example has 4 inputs and 3 outputs as shown in Table 10. The additional parameters adopted are shown in Table 11. In Table 12, we show a comparison of the results of all the approaches adopted. The best solution found for this example has 7 gates and is graphically shown in Figure 7. In this case, none of the binary versions of PSO was able to generate feasible circuits. Note that the performance of EAPSO was better than that of EAMPSO both in terms of average fitness (55.85 vs. 53.30) and in terms of frequency with which the best solution was found (10% vs. 5%). However, EBMPSO had a slightly better performance than EBPSO both in terms of average fitness (58.90 vs. 58.75) and in terms of the frequency with which the best solution was found (35% vs. 15%). Nevertheless, EBPSO had a slightly better percentage of feasible circuits found than EBMPSO (70% vs. 65%). Although marginally, we conclude that EBMPSO was the best overall performer in this example. The MGA was not able to generate circuits with 7 gates, but it found feasible circuits more consistently than most of the PSO-based approaches.

5.5 Example 5

Our fifth example has 4 inputs and 4 outputs, as shown in Table 13. The additional parameters adopted are shown in Table 14. In Table 15, we show a comparison of the results of all the approaches adopted. The best solution found for this example has 7 gates and is graphically shown in Figure 8. In this case, none of the binary versions of PSO was able to produce feasible circuits. The performance of EAMPSO was considerably better than that of its single-objective counterpart (EAPSO) both in terms of frequency of the best solution found (30% vs. 10%) as in terms of

D	**C**	**B**	**A**	S_0	S_1	S_2
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

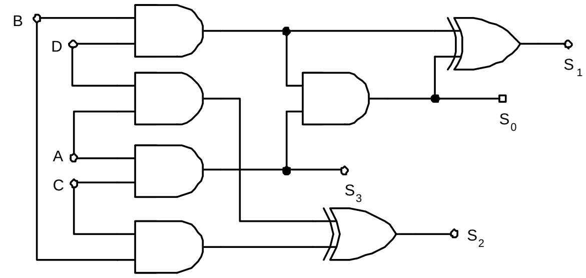
Table 10. Truth table for example 4.

Technique	**Population size**	**Iterations**	**FFE**
MPSO	147	5,000	735,000
PSO	50	14,700	735,000
MGA	490	1,500	735,000

Table 11. Parameters adopted for example 4. FFE = Fitness function evaluations.

approach	**gates b.s.**	**freq. b.s.**	**feas. circs.**	**avg.# gates**	**avg. fitn.**	**std. dev.**
BMPSO	*	0%	0%	*	44.5	1.100
EAMPSO	7	5%	45%	19.7	53.3	8.053
EBMPSO	7	35%	65%	14.1	58.9	8.985
BPSO	*	0%	0%	*	45.65	1.089
EAPSO	7	10%	55%	17.15	55.85	8.610
EBPSO	7	15%	70%	14.25	58.75	8.123
MGA	8	10%	70%	15.9	57.1	7.490

Table 12. Comparison of the results obtained by our multiobjective versions of PSO, our single-objective PSO versions, MGA and a human designer for the fourth example. b.s.=best solution.



$$S_0 = (CA)(DB);\ S_3 = CA$$
$$S_1 = DB \oplus (CA)(DB);\ S_2 = DA \oplus BC$$

Figure 8. Diagram and boolean expression corresponding to the best solution found for example 5.

D	**C**	**B**	**A**	S_0	S_1	S_2	S_3
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

Table 13. Truth table for example 5.

the percentage of feasible circuits found (80% vs. 35%). EBMPSO had also a better performance than its single-objective counterpart (EBPSO) both in terms of frequency of the best solution found (25% vs. 15%) as in terms of the percentage of feasible circuits found (75% vs. 35%). In this case, the MGA performed better than any of the PSO-based approaches, producing the highest average fitness (80.4) with the lowest number of fitness function evaluations. Thus, the MGA was the best overall performer in this example.

6 Conclusions and Future Work

In this paper, we have introduced a population-based PSO approach to design combinational logic circuits. Six PSO-based algorithms were compared (using both single- and multi-objective schemes and different encodings). Also, a population-based genetic algorithm (MGA) was included in the comparison. The results indicate that the

Technique	Population size	Iterations	FFE
MPSO	195	5,000	975,000
PSO	50	19,500	975,000
MGA	650	500	325,000

Table 14. Parameters adopted for example 5. FFE = Fitness function evaluations.

approach	gates b.s.	freq. b.s.	feas. circs.	avg. # gates	avg. fitn.	std. dev.
BMPSO	*	0%	0%	*	60.35	0.7452
EAMPSO	7	30%	80%	11.8	77.2	7.7432
EBMPSO	7	25%	75%	13.15	75.85	8.0934
BPSO	*	0%	0%	*	60.75	0.6387
EAPSO	7	10%	35%	21.2	67.8	8.9713
EBPSO	7	15%	35%	22.05	66.95	8.64
MGA	7	15%	100%	8.6	80.4	1.14

Table 15. Comparison of the results obtained by our multiobjective versions of PSO, our single-objective PSO versions, MGA and a human designer for the fifth example. b.s.=best solution.

population-based PSO approaches proposed perform better than the MGA. Also, within the six PSO-based techniques compared, those adopting both multi-objective selection schemes and an **Integer B** encoding [4] were the best overall performers. From the results, it can be clearly seen that the use of binary PSO is not a good choice when designing combinational logic circuits, since in some cases this sort of encoding was not able to even reach the feasible region. Also, the results seem to suggest that PSO is a better search engine than a genetic algorithm when adopting the population-based selection scheme described in this paper.

As part of our future work, we are interested in exploring alternative encodings (e.g., graphs and trees) that have not been used so far in particle swarm optimizers. We are also interested in studying some alternative multi-objective selection schemes (e.g., Pareto ranking) in the context of combinational circuit design using PSO [5].

Acknowledgements

The first author acknowledges support from CONACyT through a scholarship to pursue graduate studies at the Computer Science Section of the Electrical Engineering Department at CINVESTAV-IPN. The second author gratefully acknowledges support from CONACyT through project 42435-Y. The third author acknowledges support from CONACyT project no. 42523.

References

[1] P. J. Angeline. Evolutionary optimization versus particle swarm optimization: philosophy and performance differences. In W. D. Porto V.W., Saravanan N. and E. A.E., editors, *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 611–618. Springer, 1998.

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

[3] C. A. Coello Coello and A. Hernández Aguirre. Design of combinational logic circuits through an evolutionary multiobjective optimization approach. *Artificial Intelligence for Engineering, Design, Analysis and Manufacture*, 16(1):39–53, January 2002.

[4] C. A. Coello Coello, E. Hernández Luna, and A. Hernández Aguirre. Use of particle swarm optimization to design combinational logic circuits. In P. C. H. Andy M. Tyrell and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. 5th International Conference, ICES 2003*, pages 398–409, Trondheim, Norway, 2003. Springer, Lecture Notes in Computer Science Vol. 2606.

[5] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, May 2002. ISBN 0-3064-6762-3.

[6] X. Hu, R. C. Eberhart, and Y. Shi. Swarm intelligence for permutation optimization: a case study on n-queens problem. In *Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, pages 243–246, Indianapolis, Indiana, USA., 2003.

[7] J. Kennedy and R. C. Eberhart. Particle Swarm Optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages 1942–1948, Piscataway, New Jersey, 1995. IEEE Service Center.

[8] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, California, 2001.

[9] S. J. Louis. *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Department of Computer Science, Indiana University, August 1993.

[10] J. F. Miller, D. Job, and V. K. Vassilev. Principles in the Evolutionary Design of Digital Circuits—Part I. *Genetic Programming and Evolvable Machines*, 1(1/2):7–35, April 2000.

[11] J. D. Schaffer. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum, 1985.