Diseño de Circuitos Lógicos Combinatorios usando Optimización mediante Cúmulos de Partículas

Erika Hernández Luna[1], Carlos A. Coello Coello[1] y Arturo Hernández Aguirre[2]
[1]CINVESTAV-IPN
Depto. Ing. Eléctrica, Sección de Computación
Av. Instituto Politécnico Nacional No. 2508
Col. San Pedro Zacatenco, México, D.F. 07300, MEXICO
ccoello@cs.cinvestav.mx
[2]CIMAT, Area de Computación,Callejón Jalisco s/n
Mineral de Valencia, Guanajuato, Guanajuato 36240, México
artha@cimat.mx

Resumen

En este artículo se presenta una propuesta basada en optimización mediante cúmulos de partículas para el diseño de circuitos lógicos combinatorios a nivel de compuertas. El algoritmo propuesto se valida usando algunos ejemplos de la literatura, y es comparado con un algoritmo genético (con representación entera), así como contra diseñadores humanos que usaron asistentes tradicionales para el diseño de circuitos (p.ej., Mapas de Karnaugh). Los resultados indican que la optimización mediante cúmulos de partículas puede ser una alternativa viable para el diseño de circuitos combinatorios a nivel de compuertas.

Palabras Clave: diseño de circuitos, inteligencia artificial, optimización, optimización mediante cúmulos de partículas, swarm intelligence.

I. Introducción

Kennedy y Eberhart [6] propusieron una técnica llamada “optimización mediante cúmulos de partículas”(*Particle Swarm Optimization* - PSO) que está inspirada en la coreografía de un bandada de aves. La idea de esta técnica es simular el movimiento de un grupo (o una población) de aves que tienen el propósito de encontrar comida. La técnica puede ser vista como un algoritmo con comportamiento distribuído que lleva a cabo (en su versión más general) una búsqueda multidimensional. En la simulación, el comportamiento de cada individuo está afectado ya sea por el mejor local (es decir, dentro de un cierto vecindario) o por el mejor individuo global. La técnica usa el concepto de población y una medida de desempeño similar al valor de aptitud usado en los algoritmos evolutivos. Así mismo, el ajuste de los individuos es análogo al uso del operador de cruza. Sin embargo, esta técnica introduce el uso del sobrevuelo de las soluciones potenciales a través del hiperespacio (usado para acelerar la convergencia) el cual no parece tener un mecanismo análogo en los algoritmos evolutivos tradicionales. Otra diferencia importante es el hecho de que PSO permite que los individuos se beneficien de sus experiencias pasadas mientras que en un algoritmo evolutivo, normalmente la población actual es la única “memoria” usada por los individuos. PSO ha tenido éxito tanto en optimización no lineal continua como en optimización binaria discreta [6, 4, 7, 8].

Hasta donde sabemos, en este artículo se presenta el primer intento de diseñar circuitos lógicos combinatorios usando PSO.

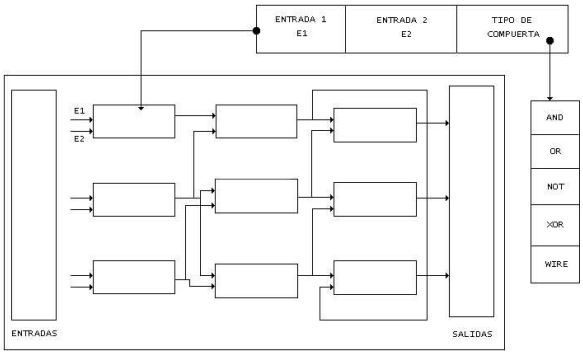

Figura 1: Matriz usada para representar un circuito. Cada compuerta obtiene su entrada de alguna de las compuertas en la columna previa. Nótese la codificación adoptada por cada elemento en la matriz así como el conjunto de compuertas disponibles usadas.

II. Representación

Se utilizó la misma representación adoptada en trabajos previos [2, 1] para codificar los circuitos. Dicha representación es mostrada en la Fig.1. Esta matriz

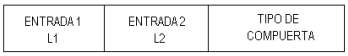

Figura 2: Codificación de cada uno de los elementos de la matriz que representa un circuito.

está codificada como una cadena de longitud fija de bits o enteros de 0 a $N-1$, donde N se refiere al número total de filas permitidas en la matriz (al que llamaremos alfabeto de cardinalidad n). Nuestros resultados serán comparados con respecto a un algoritmo genético (AG) que utiliza un alfabeto de cardinalidad n, ya que se ha encontrado que esta versión del algoritmo produce de manera consistente mejores resultados que su contraparte binaria [2].

Formalmente, podemos decir que cualquier circuito puede ser representado como un arreglo bidimensional de compuertas $S_{i,j}$, donde j indica el *nivel* de la compuerta, por lo que aquellas compuertas cercanas a las entradas tienen valores más pequeños de j. (Los valores de los niveles son incrementados de izquierda a derecha en la Fig.1). Para un valor fijo de j, el índice i varía con respecto a las compuertas que están una a continuación de la otra en el circuito, pero no necesariemante conectadas. Cada elemento de la matriz es una compuerta (existen 5 tipos de compuertas: AND, NOT, OR, XOR y WIRE[1]) que recibe sus dos entradas desde cualquier compuerta en la columna previa como se muestra en la Fig.1. A pesar de que nuestra implementación permite compuertas con más entradas y éstas a su vez pueden venir de cualquier nivel previo en el circuito, nos limitamos a compuertas de 2 entradas las cuales se restringieron para que vinieran solamente desde un nivel previo. Esta restricción podría eliminarse, pero se adoptó para permitir una comparación más justa con nuestra técnica previa basada en un AG.

Una cadena cromosómica codifica la matriz mostrada en la Fig.1 usando tercias en donde los primeros 2 elementos se refieren a cada una de las entradas usadas y el tercero es la compuerta correspondiente del conjunto de compuertas disponibles.

La representación de matriz adoptada en este trabajo fue propuesta originalmente por Louis [10, 9], quien aplicó su técnica a un sumador de 2 bits y al problema de verificación de paridad (para $n = 4, 5, 6$). Esta representación ha sido adoptada también por Miller et al. [11, 12] con algunas diferencias. Por ejemplo, las restricciones con respecto a la fuente de ciertas entradas a ser alimentadas en un elemento de la matriz, varían en cada uno de los tres enfoques: Louis [9] tiene restricciones duras, Miller et al. [11] no tiene restricciones y nosotros tenemos restricciones ligeras . La codificación también es diferente en todos los casos. Louis [9] sólo codifica la información referente a una entrada y el tipo de compuerta a ser usada en cada posición de la matriz. El también usa una representación binaria. En nuestro caso, hemos usado tanto el alfabeto de cardinalidad n como un alfabeto binario y codificamos la compuerta que va a ser colocada en cada localidad de la matriz además de sus dos entradas. Miller et al. [11] codifican una expresión Booleana completa usando un entero simple. Esta representación es más compacta, pero tiene el problema de requerir que la mutación tome el lugar de la cruza para introducir suficiente diversidad en la población, de manera que el algoritmo evolutivo pueda acercarse a la región factible.

Finalmente, la última diferencia entre los tres enfoques antes mencionados es referente a la función de aptitud. Louis [9] simplemente maximiza el número de las salidas producidas por el circuito y que son iguales a las indicadas en la tabla de verdad. Nosotros usamos una función de aptitud que trabaja en dos etapas: en la primera, se maximiza el número de aciertos con respecto a la tabla de verdad (como en el caso de Louis) y en la segunda, una vez que se ha encontrado una solución factible se maximiza el número de WIREs en el circuito. Haciendo esto, se puede optimizar el circuito en términos del número de compuertas que usa. Miller et al. [11] hicieron algo similar a Louis hasta hace poco tiempo (en trabajo más reciente introdujeron una función de aptitud de dos etapas como la adoptada por nosotros [5]).

Por lo tanto, podemos decir que nuestro objetivo es producir un diseño completamente funcional (o sea, uno que produzca todas las salidas esperadas para cualquier combinación de entradas de acuerdo a la tabla de verdad dada para el problema) que maximice el número de WIREs (o sea, que minimice el número total de compuertas).

III. Técnica Propuesta

La motivación principal de usar optimización mediante cúmulos de partículas (PSO) para diseñar circuitos combinatorios es que se ha encontrado que este algoritmo es muy eficiente en una gran variedad de tareas [8]. Sin embargo, la mayoría de los usos exitosos de PSO reportados en la literatura trabajan con una representación de números reales y en este caso, usaremos una codificación binaria. A pesar de que una representación con números reales es posible (de hecho, estamos trabajando actualmente en tal versión del algoritmo), los resultados preliminares reportados en este artículo fueron encontrados con una representación binaria que trabaja razonablemente bien. El uso de números reales para representar un circuito requiere de un mapeo del genotipo al fenotipo más sofis-

[1]WIRE básicamente indica una operación nula, o en otras palabras, la ausencia de compuerta y es sólo para mantener la regularidad en la representación usada, ya que de otra forma se habrían tenido que usar cadenas de longitud variable.

1. For i = 1 to M (M = tamaño de la población)
   Iniciar $P[i]$ aleatoriamente
   (P es la población de partículas)
   Inicializa $V[i] = 0$
   (V = velocidad de cada partícula)
   Evaluar $P[i]$
   $GBEST$ = Mejor partícula encontrada $P[i]$
2. Final del For
3. For i = 1 to M
   $PBESTS[i] = P[i]$
   (Inicializa la "memoria"de cada partícula)
4. Fin del For
5. Repetir
   For i = 1 to M
   $V[i] = V[i-1] + \Phi_1 \times (PBESTS[i] - P[i])$
   $+\Phi_2 \times (PBESTS[GBEST] - P[i])$
   (Calcula la velocidad V[i] para cada partícula)
   (Φ_1 y Φ_2 son los límites superiores
   de números aleatorios positivos
   con distribución uniforme)
   $POP[i] = P[i] + V[i]$
   Si una partícula se sale de
   hipercubo definido
   entonces es reintegrada dentro
   de los límites
   Evaluar $P[i]$
   Si la nueva posición es mejor entonces
   $PBESTS[i] = P[i]$
   $GBEST$ = La mejor partícula
   encontrada en $P[i]$
   Fin del For
6. Hasta que se encuentra la condición de paro

Figura 3: Pseudocódigo del algoritmo de optimización mediante cúmulos de partículas

ticado. A continuación, describiremos los detalles de nuestra implementación.

El algoritmo general del PSO binario se muestra en la Fig.3. En PSO existen dos tipos de información disponible para las partículas de manera que puedan tomar la mejor decisión de hacia dónde se moverán. Una de éstas es su propia experiencia en la búsqueda (es decir, una partícula ha pasado a través de varios estados y "sabe" cuál de ellos ha sido el mejor). Adicionalmente, la partícula también sabe acerca del desempeño de las partículas en su vecindario (o sea, sabe cuales han sido los mejores estados que sus vecinos han alcanzado hasta el momento). Estas dos piezas de información corresponden al aprendizaje individual y la transmisión cultural, respectivamente [8].

Matemáticamente hablando, la versión binaria del PSO está definida de manera que la probabilidad de que un individuo seleccione el cero o el uno (o sea, falso o verdadero) es [7]:

(1) $P(x_{id}(t) = 1) = f(x_{id}(t-1), v_{id}(t-1), p_{id}, p_{gd})$

Cuadro I: Tabla de verdad para el circuito del primer ejemplo.

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

donde:

- $P(x_{id}(t) = 1)$ es la probabilidad de que el individuo i seleccione 1 para el bit en la posición d de la cadena binaria.
- $x_{id}(t)$ es el estado actual de la cadena en la posición d del individuo i.
- t se refiere a la iteración actual.
- $v_{id}(t-1)$ es la velocidad de la partícula y esta definida como la medida de la predisposición del individuo o la probabilidad actual para seleccionar 1.
- p_{id} es el mejor estado encontrado hasta el momento.
- p_{gd} es el mejor estado encontrado hasta el momento dentro del vecindario.

A pesar de que el principal ajuste en la expresión usada por el PSO puede ser vista como una forma de mutación, encontramos que su poder exploratorio no era suficiente para el diseño de circuitos. Por ello, agregamos un operador de mutación uniforme como el usado en los algoritmos genéticos tradicionales. Este operador, sin embargo, fue aplicado solamente a un cierto porcentaje de la población (éste es un parámetro definido por el usuario). De nuestros experimentos, determinamos que un valor entre 1 % y 3 % era apropiado para usarse como el porcentaje de la población sujeto a la mutación.

IV. Resultados

Usamos algunos ejemplos tomados de la literatura para probar nuestra implementación. Los resultados fueron comparados con aquellos obtenidos por dos diseñadores humanos y por un algoritmo genético con una representación de cardinalidad n (ver [2] para detalles).

Cuadro II: Comparación de resultados entre el algoritmo PSO, el AG de cardinalidad n (NGA) y dos diseñadores humanos para el circuito del primer ejemplo.

NGA	Diseñador Humano 1
$F = Z(X+Y) \oplus XY$	$F = Z(X \oplus Y) + Y(X \oplus Z)$
4 compuertas	5 compuertas
2 ANDs, 1 OR, 1 XOR	2 ANDs, 1 OR, 2 XORs
PSO	**Diseñador Humano 2**
$F = ((Y+Z)X) \oplus YZ$	$F = X'YZ + X(Y \oplus Z)$
4 compuertas	6 compuertas
2 ANDs, 1 OR, 1 XOR	3 ANDs, 1 OR, 1 XOR, 1 NOT

A. Ejemplo 1

Nuestro primer ejemplo tiene 3 entradas y 1 salida y su tabla de verdad se muestra en el Cuadro I. En este caso, la matriz usada fue de tamaño 5×5, y la longitud de cada cadena que representa un circuito fue de 75. Ya que se podía seleccionar entre 5 compuertas en cada posición de la matriz, entonces el espacio de búsqueda intrínseco (o sea, el tamaño máximo permitido como consecuencia de la representación usada) es l donde l se refiere a lo longitud requerida para representar un circuito (l= 75 en nuestro caso). Por ende, el tamaño del espacio de búsqueda intrínseco es $5^{75} \approx 2.6 \times 10^{52}$. La aptitud fue calculada de la siguiente manera: 8 (número de salidas correctas que deben obtenerse para obtener un circuito factible) + 5×5 (tamaño de la matriz) - número de compuertas usadas (o sea, diferentes de WIRE). Una aptitud de 29 (el mejor valor producido por el circuito) significa que el circuito es factible (de otra forma, su aptitud no podría ser más grande que 8), y tiene 4 compuertas (es decir, 21 WIREs), porque 8 + (25-4) = 8 +21 = 29.

La representación gráfica del mejor circuito producido por PSO se muestra en la Fig.4. Nuestro algoritmo PSO encontró esta solución usando los siguientes parámetros[2]: 90 partículas, 300 iteraciones, (es decir, se requirieron 27,000 evaluaciones de la función objetivo), $\phi 1 = \phi 2$ =0.8, V_{max} =3.0, $P_m = 1\,\%$. Los parámetros usados para el NGA fueron los siguientes: porcentaje de cruza = 50 %, porcentaje de mutación = 0.5/75 $\times$ 100 = 0.22 %, tamaño de la población = 90, número máximo de generaciones = 300.

La comparación de los resultados producidos por el PSO, el NGA y los dos diseñadores humanos se muestran en el Cuadro II. En este caso, el diseñador humano 1 usó mapas de Karnaugh además de identidades del álgebra Booleana para simplificar el circuito. El diseñador humano 2 usó el procedimiento de Quine-McCluskey.

PSO produjo circuitos factibles el 100 % de las veces y encontró el óptimo en 7 de las 20 corridas (o sea, 35 % de las veces), alcanzándose una aptitud de 28 en todas las corridas. La aptitud promedio de las 20 corridas hechas fue 28.35 con una desviación estándar de 0.49. La representación gráfica de la mejor solución encontrada por el PSO se muestra en la Fig.4.

[2]Estos parámetros fueron derivados empíricamente

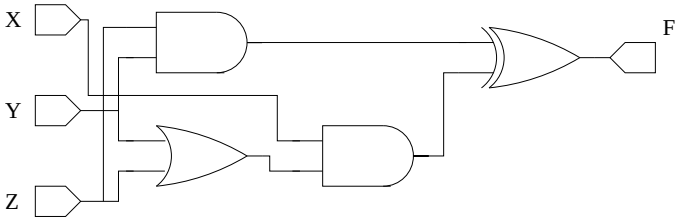

Figura 4: Representación gráfica del mejor circuito encontrado por nuestro algoritmo PSO para el primer ejemplo.

Cuadro III: Tabla de verdad para el circuito del segundo ejemplo.

Z	W	X	Y	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
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	0

Por otro lado, la mejor solución encontrada por el NGA usando el mismo tamaño de población tiene también una aptitud de 29 (o sea, un circuito con 4 compuertas), pero aparece sólo el 10 % de las veces. También, 20 % de las veces se encontró una solución no factible. La aptitud promedio de estas 20 corridas fue 21.4 con una desviación estándar de 8.438009244. En este ejemplo, nuestro algoritmo PSO mostró un mejor desempeño (en promedio) que el NGA.

B. Ejemplo 2

Nuestro segundo ejemplo tiene 4 entradas y una sola salida,como se muestra en el Cuadro III. En este caso, la matriz usada fue también de tamaño 5×5. Nuestro algoritmo PSO usó los siguientes parámetros: 200 partículas, 1000 iteraciones (es decir, se necesitaron 200,000 evaluaciones de la función objetivo),

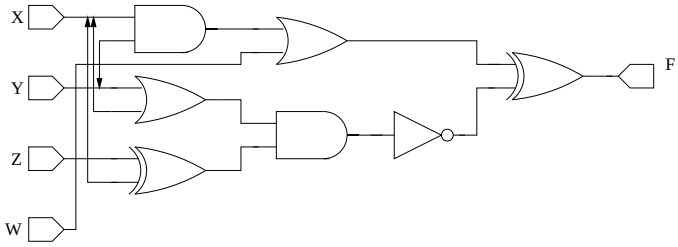

Figura 5: Representación gráfica del mejor circuito encontrado por PSO para el segundo ejemplo.

Cuadro IV: Comparación de resultados entre PSO, un AG de cardinalidad n (NGA), un diseñador humano y el enfoque de Sasao para el circuito del segundo ejemplo.

NGA
$F = (WYX' \oplus ((W + Y) \oplus Z \oplus (X + Y + Z)))'$
10 compuertas
2 ANDs, 3 ORs, 3 XORs, 2 NOTs
Diseñador Humano 1
$F = ((Z'X) \oplus (Y'W')) + ((X'Y)(Z \oplus W'))$
11 compuertas
4 ANDs, 1 OR, 2 XORs, 4 NOTs
PSO
$F = (XY + W) \oplus ((Z \oplus X)(X + Y))'$
7 compuertas
2 ANDs, 2 ORs, 2 XORs, 1 NOT
Sasao
$F = X' \oplus Y'W' \oplus XY'Z' \oplus X'Y'W$
12 compuertas
3 XORs, 5 ANDs, 4 NOTs

Cuadro V: Tabla de verdad para el circuito del tercer ejemplo.

A	**B**	**C**	**D**	**F_1**	**F_2**	**F_3**
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

$\phi 1 = \phi 2$ =0.8,V_{max} =3.0, P_m = 3 %. Los parámetros usados por el NGA fueron los siguientes: porcentaje de cruza = 50 %, porcentaje de mutación = 0.22 %, tamaño de la población = 200, número máximo de generaciones = 1000.

La comparación de los resultados producidos por PSO, el AG de cardinalidad n (NGA), un diseñador humano (usando mapas de Karnaugh), y el enfoque de Sasao [13] son mostrados en el Cuadro IV. Sasao ha usado este circuito para ilustrar la técnica de simplificación basada en el uso de ANDs y XORs. Esta solución usa, sin embargo, más compuertas que la solución producida por nuestra técnica.

Nuestro algoritmo encontró una solución con un valor de aptitud de 34 (es decir, un circuito con 7 compuertas) 20 % de las veces y se encontraron circuitos factibles 67 % de las veces. La aptitud promedio de las 20 corridas realizadas fue de 29.35, con una desviación estándar de 7.4. La representación gráfica de la mejor solución encontrada se muestra en la Fig.5.

La mejor solución que el NGA pudo encontrar usando el mismo tamaño de población tuvo una aptitud de 31 (o sea, un circuito con 10 compuertas), y apareció sólo el 20 % de las veces. Así mismo, 65 % del tiempo la mejor solución encontrada por el NGA no fue factible. La aptitud promedio de las 20 corridas efectuadas fue de 20.25, con una desviación estándar de 7.68. En este ejemplo, nuestro algoritmo de PSO mostró un mejor desempeño que el NGA.

Cuadro VI: Comparación de resultados entre nuestro algoritmo PSO, el AG de cardinalidad n (NGA) y un diseñador humano para el circuito del tercer ejemplo.

NGA
$F_1 = B \oplus D$
$F_2 = (A \oplus C) \oplus BD$
$F_3 = AC + BD(A \oplus C)$
7 compuertas
2 ANDs, 1 OR, 4 XORs
Diseñador Humano 1
$F_1 = A \oplus D$
$F_2 = (A \oplus C)D' + ((A \oplus C) \oplus B)D$
$F_3 = AC + BD(A + C)$
12 compuertas
5 ANDs, 3 ORs, 3 XORs, 1 NOT
PSO
$F_1 = B \oplus D$
$F_2 = (BD) \oplus (A \oplus C)$
$F_3 = (AC) + ((BD)(A \oplus C))$
7 compuertas
3 XORs, 3 ANDs, 1 OR

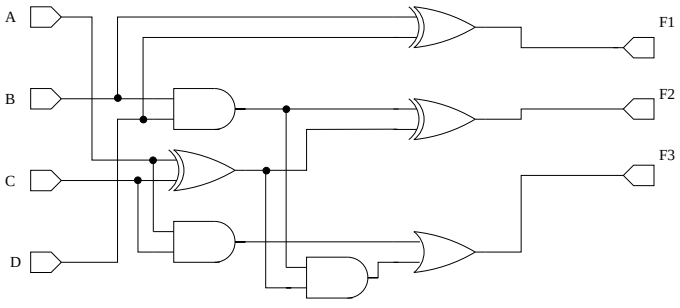

Figura 6: Representación gráfica del mejor circuito encontrado por nuestro algoritmo PSO para el tercer ejemplo.

C. Ejemplo 3

El tercer ejemplo es un sumador de dos bits (4 entradas y 3 salidas) y su tabla de verdad se muestra en el Cuadro V. La matriz usada en este caso fue nuevamente de 5×5. Nuestro algoritmo de PSO usó los siguientes parámetros: 300 partículas, 2000 iteraciones (o sea, se requirieron 600,000 evaluaciones de la función objetivo), $\phi 1 = \phi 2$= 0.8, V_{max} =3.0,P_m = 1 %. Los parámetros usados por el NGA fueron los siguientes: porcentaje de cruza = 50 %, porcentaje de mutación = 0.22 %, tamaño de la población = 300, número máximo de generaciones = 2000.

Las comparaciones de los resultados producidos por el PSO, el AG de cardinalidad n (NGA) y un diseñador humano (usando los mapas de Karnaugh) son mostrados en el Cuadro VI.

Nuestro algoritmo PSO encontró una solución con aptitud de 66 (o sea, un circuito con 7 compuertas) pero solamente una vez en las 20 corridas desarrolladas. Solamente el 20 % de las veces se encontraron circuitos factibles. La aptitud promedio de las 20 corridas desarrolladas fue de 48.85, con una desviación estándar de 6.82. La representación gráfica de la mejor solución encontrada por nuestro algoritmo PSO es mostrada en la Fig.6

La mejor solución que pudo encontrar el NGA usando el mismo número de evaluaciones de la función de aptitud tuvo una aptitud de 66 (o sea, un circuito factible con 7 compuertas). Esta solución apareció también sólo una vez en las 20 corridas.Sin embargo, el NGA pudo encontrar soluciones factibles el 75 % de las veces. La aptitud promedio de estas 20 corridas fue 58.2, con una desviación estándar de 7.17. Es claro que en este ejemplo el NGA tuvo un mejor desempeño (en promedio) que nuestro algoritmo PSO.

V. Conclusiones y Perspectivas

Hemos presentado la primera propuesta formal del uso del algoritmo de optimización mediante cúmulos de partículas para diseñar circuitos lógicos combinatorios. La técnica presentada parece prometedora, ya que produce resultados competitivos (y en algunos casos mejores) con respecto al algoritmo genético de cardinalidad n (excepto por el último ejemplo) y consistentemente mejora las soluciones encontradas por los diseñadores humanos. En realidad, nuestros experimentos[3] indican que nuestro PSO es muy competitivo con circuitos de una sola salida, pero que la técnica no es tan robusta cuando se trata con circuitos de varias salidas.

Una de las limitaciones actuales de nuestro enfoque es el poder exploratorio del algoritmo el cual no es

[3]Existen algunos ejemplos más disponibles que no fueron incluidos debido a limitaciones de espacio.

tan bueno como nosotros esperábamos. Esto es más evidente en circuitos con más de una salida como el sumador de dos bits en el cual, nuestro algoritmo de PSO tuvo un desempeño muy pobre con respecto al desempeño del NGA (en promedio).

Como parte de nuestro trabajo futuro, planeamos introducir una técnica poblacional como la propuesta en [3] para mejorar las capacidades de búsqueda de nuestro algoritmo. También estamos trabajando en una representación indirecta que nos permita usar números reales para representar circuitos. Nuestra hipótesis es que PSO tendrá un mejor desempeño si podemos utilizar una representación con números reales ya que hay evidencia de este comportamiento positivo en la literatura [8].

Agradecimientos

El primer autor agradece el apoyo de CONACyT a través de una beca para realizar los estudios de posgrado en Ciencias de la Computación en la Sección de Computación del departamento de Ingeniería Eléctrica del CINVESTAV-IPN. El segundo autor agradece el apoyo de CONACyT a través del proyecto NSF-CONACyT número 32999-A. El tercer autor agradece el apoyo de CONACyT a través del proyecto número I-39324-A.

Referencias

[1] Carlos A. Coello Coello, Alan D. Christiansen, and Arturo Hernández Aguirre. Automated Design of Combinational Logic Circuits using Genetic Algorithms. In D. G. Smith, N. C. Steele, and R. F. Albrecht, editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 335–338. Springer-Verlag, University of East Anglia, England, April 1997.

[2] Carlos A. Coello Coello, Alan D. Christiansen, and Arturo 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] Carlos A. Coello Coello, Arturo Hernández Aguirre, and Bill P. Buckles. Evolutionary Multiobjective Design of Combinational Logic Circuits. In Jason Lohn, Adrian Stoica, Didier Keymeulen, and Silvano Colombano, editors, *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware*, pages 161–170, Los Alamitos, California, July 2000. IEEE Computer Society.

[4] Russell C. Eberhart and Yuhui Shi. Comparison between Genetic Algorithms and Particle Swarm Optimization. In V. W. Porto, N. Saravanan, D. Waagen, and A.E. Eibe, editors, *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 611–619. Springer-Verlag, March 1998.

[5] Tatiana Kalganova and Julian Miller. Evolving more efficient digital circuits by allowing circuit layout and multi-objective fitness. In Adrian Stoica, Didier Keymeulen, and Jason Lohn, editors, *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*, pages 54–63, Los Alamitos, California, 1999. IEEE Computer Society Press.

[6] James Kennedy and Russell 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.

[7] James Kennedy and Russell C. Eberhart. A Discrete Binary Version of the Particle Swarm Algorithm. In *Proceedings of the 1997 IEEE Conference on Systems, Man, and Cybernetics*, pages 4104–4109, Piscataway, New Jersey, 1997. IEEE Service Center.

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

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

[10] Sushil J. Louis and Gregory J. Rawlins. Using Genetic Algorithms to Design Structures. Technical Report 326, Computer Science Department, Indiana University, Bloomington, Indiana, February 1991.

[11] J. F. Miller, P. Thomson, and T. Fogarty. Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, pages 105–131. Morgan Kaufmann, Chichester, England, 1998.

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

[13] Tsutomu Sasao, editor. *Logic Synthesis and Optimization*. Kluwer Academic Press, 1993.