Programación genética para el diseño de circuitos lógicos

Eduardo Serna Pérez
Sección Computación CINVESTAV-IPN
México D.F., México.
Email: eserna@computacion.cs.cinvestav.mx

Katya Rodríguez Vázquez
DISCA IIMAS-UNAM
México D.F., México.
Email: katya@uxdea4.iimas.unam.mx

Carlos A. Coello Coello
Sección Computación CINVESTAV-IPN
México D.F., México.
Email: ccoello@cs.cinvestav.mx

***Resumen*—En este artículo proponemos utilizar una técnica de computación evolutiva llamada programación genética para diseñar circuitos lógicos combinatorios (a nivel de compuertas). El proceso se centra en probar y evolucionar expresiones lógicas que finalmente produzcan un diseño lógico válido. En este trabajo se muestran un par de experimentos realizados con circuitos relativamente sencillos con el énfasis de mostrar el alcance de la técnica aquí presentada y observar el grado de complejidad y dimensionalidad del problema de diseño bajo estudio.**

I. Introducción

El diseño de circuitos lógicos combinatorios ha sido un área de investigación muy activa en los últimos años. Este interés es debido a lo complejo que resulta su diseño y especialmente su simplificación. Por años, se ha utilizado el álgebra booleana como la manera más sencilla de elaborar e implementar el diseño de circuitos lógicos. Posteriormente, se desarrollaron diversos métodos de inspección visual y basados en implicantes primos, que fueron dirigidos a la síntesis de circuitos, tales como los mapas de Karnaugh [9] y el método de Quine-McCluskey [15], [12] y algunas implementaciones más actuales como ESPRESSO [3] que utiliza un conjunto de heurísticas para optimizar circuitos combinatorios.

Las técnicas evolutivas han comenzado a incursionar en el diseño de circuitos debido principalmente a su poder exploratorio, ya que permiten explorar simultáneamente diversas regiones de un espacio de diseño y encontrar varias soluciones a problemas con espacios de búsqueda grandes y/o accidentados (a esta área se le denomina *Hardware Evolutivo*).

La programación genética elabora y sintetiza programas y funciones que describen un comportamiento deseado. En tal caso, resulta factible la idea de construir funciones booleanas que representen el funcionamiento de un circuito lógico definido por medio de una tabla de verdad. Con el propósito de mostrar los diseños producidos por nuestro programa, se utilizaron circuitos relativamente pequeños, cuyas soluciones están documentadas en la literatura especializada.

II. Problema a solucionar

El problema que nos interesa resolver consiste en diseñar un circuito digital combinatorio que realice una cierta función lógica (especificada por una tabla de verdad), dado un cierto conjunto de compuertas lógicas disponibles, tratando de utilizar el menor número posible de ellas. Analizaremos la calidad de los circuitos producidos por nuestra técnica con respecto a otros algoritmos. Cualquier implementación que utilice un número menor de compuertas lógicas puede ser considerada como una mejora en el diseño, debido a que el óptimo (bajo esta métrica) para un circuito cualquiera es desconocido.

III. Antecedentes

La computación evolutiva es el término que se utiliza para englobar a todas aquellas técnicas de optimización, búsqueda y aprendizaje de máquina inspiradas en las teorías genéticas y de la evolución que gobiernan la adaptación de las especies en el planeta.

En las técnicas evolutivas se manipula un conjunto de soluciones potenciales en cada iteración, lo que implica un alto grado de paralelismo, pues se explotan a la vez diferentes regiones del espacio de búsqueda. Además, los operadores probabilísticos que utilizan los algoritmos evolutivos evitan que éstos queden atrapados en un óptimo local. La inteligencia artificial considera a las técnicas evolutivas como heurísticas subsimbólicas, debido a que su representación del conocimiento es numérica y no simbólica.

Existen distintas variantes de las técnicas evolutivas, pero se podrían agrupar en 3 principales paradigmas que son: la Programación Evolutiva, la Estrategia Evolutiva y el Algoritmo Genético.

La Programación Genética es un paradigma que nace del Algoritmo Genético y fue propuesto de manera independiente por Cramer [6] y Koza [11], quienes sugirieron que una estructura de árbol podía ser usada como la representación de un programa en un genoma. En tal caso, los individuos son programas de computadora estructurados jerárquicamente. El tamaño, la forma y el contenido de estos programas de computadora pueden

cambiar dramáticamente durante el proceso de evolución. Los individuos están formados por un conjunto de términos y funciones, que actúan como primitivas que sirven de base para la construcción de programas [2].
El algoritmo básico para la Programación Genética es el siguiente [2]:

1. Generar aleatoriamente una población inicial de programas
2. Mientras que la nueva población no sea completada
   - Evaluar los programas en la población y asignar un valor de aptitud
   - Seleccionar a los individuos con base en su aptitud
   - Aplicar los operadores genéticos de cruza y mutación en los individuos seleccionados
   - Reemplazar la población existente con la nueva población
3. Ciclar hasta cumplir el criterio de terminación
4. Presentar al mejor individuo de la población.

Los individuos son seleccionados generalmente en forma probabilística de acuerdo a su contribución de aptitud con respecto al total de la población.

El operador de *cruza* combina el material genético de dos padres, intercambiando parte de un padre con una parte del otro [1] como se muestra en la Fig. 1. La cruza consiste en [11], [2]:

- Seleccionar dos individuos como padres
- Seleccionar aleatoriamente un subárbol o segmento de instrucciones
- Intercambiar los subárboles o segmentos de código entre los dos padres
- Evitar sustituciones de nodo terminal en el nodo raíz

El operador de *mutación* selecciona un punto al azar en la estructura y reemplaza el subárbol con uno nuevo generado aleatoriamente (Fig. 1).

Por otro lado, el *Hardware Evolutivo* consiste en el diseño de circuitos electrónicos que son evolucionados a través de la simulación del proceso de selección natural [13]. George J. Friedman [7] fue uno de los primeros en aplicar técnicas evolutivas al diseño de circuitos. Friedman propuso un mecanismo para construir, probar y evaluar circuitos en forma automática, utilizando mutaciones aleatorias y procesos de selección.

La idea principal del *hardware evolutivo* consiste en codificar los circuitos en un cromosoma y utilizar un proceso de ensamble y prueba, que unido a un proceso evolutivo nos permita diseñar circuitos de distinto grado de complejidad y así explorar de manera más eficiente el espacio de diseño. Existen diversas formas de diseñar circuitos evolutivos: en línea (directamente evaluados en hardware reconfigurable), o fuera de línea (usando simulaciones)[8]. También existen procesos de evolución dirigidos a *nivel de compuertas* que consisten en emplear sólo compuertas lógicas básicas como pueden ser AND, OR y NOT. Otra alternativa es diseñar a *nivel de funciones*, lo cual consiste básicamente en utilizar compuertas lógicas y a su vez construir con ellas nuevos módulos compuestos, como las ADF's entre otras [13], [8].

Existe trabajo previo de diseño de circuitos combinatorios usando algoritmos genéticos como los de Coello *et al.* [4] que emplean una representación matricial del circuito, otros trabajos reportados en programación genética por Koza [11], Miller *et al.* [13] y Kalganova [8] que utilizan distintas representaciones de los circuitos y otros como Coello *et al.* [5] y Rodríguez *et al.* [16] que aplican técnicas multiobjetivo. Algunas de estas implementaciones por lo general necesitan un mapeo del cromosoma para evaluar un circuito.

[1]Generalmente, los subárboles en cada padre difieren uno de otro en su contenido y/o tamaño

IV. Técnica Propuesta

En nuestra implementación[2] utilizamos cadenas prefijas [10] que codifican funciones booleanas, constituidas por un conjunto de funciones y términos. Un proceso de ensamble y prueba verificará si las cadenas prefijas constituyen un diseño 100 % funcional[3], asignándole un valor de aptitud con respecto a la salida deseada. Posteriormente se añade un proceso dirigido a la minimización de la cadena prefija. Finalmente la simulación de los procesos evolutivos permite que las cadenas prefijas experimenten la transferencia de material genético que constituye una solución parcial al espacio de diseño deseado, obteniendo como resultado una expresión prefija que representa el diseño funcional de un circuito arbitrario.

A. Representación de los individuos

El genoma de cada individuo tiene una distribución de cadenas prefijas sintácticamente válida constituída por un conjunto de funciones y términos:

"Salida0" genoma0: &X^|!ZWY
"Salida1" genoma1: &!XY

El conjunto de *funciones* está formado por caracteres que representan el comportamiento de una función lógica determinada y el conjunto de *términos* son caracteres de la *A* a la *Z* que tienen aridad cero y que serán los valores booleanos de entrada para cada función (Fig. 2).

Compuerta	Símbolo	Aridad
NOT	!	1
AND	&	2
XOR	^	2
OR	|	2

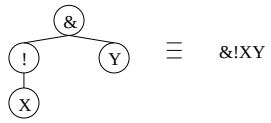

Fig. 2. Conjunto de funciones utilizadas en la representación de circuitos lógicos en PG Prefija. A la derecha se muestra una expresión prefija representada en un árbol.

El número de nodos tipo función y término que componen una expresión prefija varía según la complejidad del circuito, no excediendo de 2^7 (128) nodos para los ejemplos presentados, ya que con circuitos de una complejidad mayor, dicha restricción en cuanto la número máximo de nodos puede cambiar.

B. Evaluador de expresiones

Las cadenas son evaluadas por un *intérprete de genomas* (parser). El intérprete lee las expresiones carácter por carácter (de derecha a izquierda) y las evalúa según su significado (ver Fig. 3). En tal caso:

[2]La implementación completa del programa se realizó en C++.

[3]Un circuito funcional es aquel que cumple con todas las salidas de la tabla de verdad.

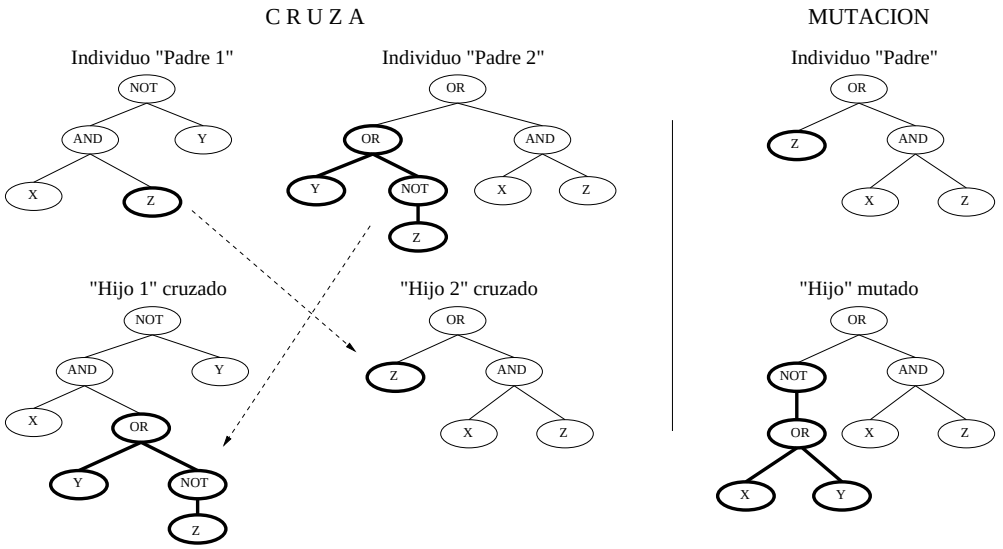

Fig. 1. Mecanismo de cruza de dos estructuras arbóreas que generan 2 nuevos hijos y el mecanismo de mutación que modifica una parte del árbol.

1. Si el carácter extraído es una letra, se le asignará un valor del tipo tabla de verdad[4]. Este valor es insertado en una pila tipo tabla de verdad. Cada nivel en la pila es de 2^n posiciones, donde n es el número de variables de entrada para el circuito.
2. En caso de un operador lógico, los valores son extraídos de la pila, evaluados por el operador lógico y el resultado es depositado nuevamente en la parte superior de la pila.

Este procedimiento continúa hasta que se haya terminado de evaluar toda la expresión. El resultado lógico está compuesto por una variable del tipo tabla de verdad de 2^n posiciones.

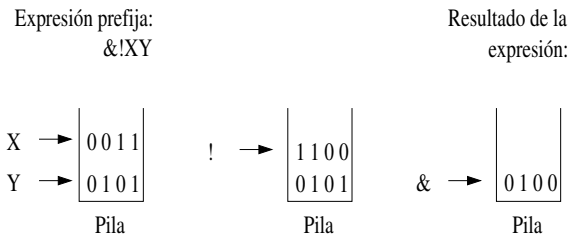

Fig. 3. Funcionamiento del intérprete de genomas. Nótese cómo lee carácter por carácter y cómo va solucionando el resultado de la expresión lógica.

C. Función de aptitud

El resultado lógico de la expresión es comparado bit a bit con la expresión de salida deseada del circuito, a fin de determinar su valor de aptitud. A mayor número de ocurrencias o aciertos con respecto de la salida deseada, mayor será la aptitud del individuo. La función de aptitud es mostrada a continuación:

$$aptitud(i) = aciertos(i)/2^n$$

La aptitud del *i-ésimo* individuo es igual al número de aciertos con cada uno de los bits de salida del circuito deseado. El total de aciertos es dividido entre 2 elevado a la n variables de entrada del circuito (la aptitud varía de 0 a 1.0).

[4]El tipo tabla de verdad es un vector de 2^n posiciones compuestas por ceros y unos, que corresponderán a las combinaciones binarias que se hayan establecido para esa variable en especial

D. Operadores genéticos

La **reproducción** es implementada utilizando un método de selección proporcional llamado Selección Universal Estocástica. Esta técnica se usa debido a que buscamos minimizar la mala distribución de los individuos en la población en función de sus valores esperados.

El operador de **cruza** consiste en seleccionar dos individuos. Después, se generan puntos de cruza al azar dentro de cada cadena cromosómica de manera independiente y se determina el tamaño de cada una de las subcadenas prefijas que serán intercambiadas. Si alguna subcadena a ser insertada provoca un crecimiento excesivo (128 nodos), se repite el proceso hasta que se logre la inserción de una subcadena de tamaño apropiado. Finalmente, se construyen los nuevos genomas intercambiando las subcadenas de cada individuo.

El operador de **mutación** nos permite explorar algunos puntos del espacio de diseño. En la mutación seleccionamos un individuo y escogemos un punto dentro de la cadena. Determinamos el tamaño de la subcadena prefija que será removida de la cadena y generamos aleatoriamente una nueva expresión prefija que será insertada en su lugar.

Finalmente se realiza el **elitismo** que asegure la transferencia genética del mejor individuo en esa población. En este caso se considera un doble elitismo: primero, el de aptitud más alta y segundo, en el caso de que la aptitud más alta sea común en más de un individuo, se tomará aquel con menor número de nodos.

V. Experimentos

Se utilizaron varios circuitos lógicos de distinto grado de complejidad para probar la técnica propuesta, denominada PG Prefija [14]. Para el propósito de este artículo se seleccionaron 2 ejemplos que ilustran la calidad de los circuitos producidos. Los resultados generados son comparados con aquellos obtenidos por otras técnicas evolutivas como son el Algoritmo Genético Binario (BGA) [4] y el Algoritmo Genético Multiobjetivo (MGA) [5], debido a que resultan bastante competitivos en lo que a diseño a nivel de compuertas se refiere. También se

PG Prefija	BGA
$F0 = Y \oplus W$ $F1 = YW \oplus (X \oplus Z)$ $F2 = YW(X \oplus Z) + XZ$	$F0 = W \oplus Y$ $F1 = (Z \oplus X) \oplus WY$ $F2 = ZX + WY(Z \oplus X)$
7 compuertas 3 ANDs, 1 OR, 3 XORs	7 compuertas 3 ANDs, 1 OR, 3 XORs
Diseñador Humano	
$F0 = W \oplus Y$ $F1 = (Z \oplus X)Y' + ((Z \oplus X) \oplus W)Y$ $F2 = ZX + WY(Z + X)$	
12 compuertas 5 ANDs, 3 ORs, 3 XORs, 1 NOT	

TABLA I

COMPARACIÓN DE LAS MEJORES SOLUCIONES OBTENIDAS POR UN ALGORITMO GENÉTICO BINARIO (BGA), PROGRAMACIÓN GENÉTICA PREFIJA (PG PREFIJA) Y MAPAS DE KARNAUGH PARA EL PRIMER EJEMPLO: UN SUMADOR DE 2 BITS.

PG Prefija
$F0 = ((Z \oplus X) + Y') \oplus ((Z \oplus X) + W)$ $F1 = ((W + ((Z \oplus X) + Y')) \oplus (X' + Z)')'$ $F2 = ((Y'W \oplus Z) + (Z \oplus X)) \oplus X$
15 compuertas
1 AND, 5 ORs, 5 XORs 4 NOTs
Diseñador Humano
$F0 = (Z \oplus X)'(W \oplus Y)'$ $F1 = Z'X + (Z \oplus X)'(W'Y)$ $F2 = (F0 + F1)'$
13 compuertas
4 ANDs, 2 ORs, 2 XORs, 5 NOTs
MGA
$F0 = ((W \oplus Y) + (Z \oplus X))'$ $F1 = F2 \oplus ((W \oplus Y) + (Z \oplus X)$ $F2 = ((W \oplus Y) + (Z \oplus X))$ $(((Z \oplus X) + (Z \oplus W)) \oplus X)$
9 compuertas 2 ANDs, 3 ORs, 3 XORs 2 NOTs

TABLA II

COMPARACIÓN DE LAS MEJORES SOLUCIONES OBTENIDAS POR MGA, LA PG PREFIJA Y DOS DISEÑADORES HUMANOS PARA EL TERCER EJEMPLO: UN COMPARADOR CON 4 ENTRADAS 3 SALIDAS.

comparan los resultados con algunas técnicas tradicionales como los Mapas de Karnaugh y el método de Quine-McCluskey.

Los párametros de tamaño de población y número de generaciones varían de acuerdo a la complejidad y dimensionalidad de cada problema.

A. Ejemplo 1

El primer ejemplo consiste en un sumador de 2 bits que consta de 4 entradas y 3 salidas, con la tabla de verdad que se muestra en la Fig. 4. Se realizaron 20 corridas con un tamaño total de población de 500 individuos y un número máximo de 800 generaciones. El 80 % de las corridas produjo circuitos funcionales y el 20 % de las ocasiones arrojó soluciones óptimas[5] con 7 compuertas. La aptitud promedio de las 20 corridas nos da como resultado una media de 0.8309, con una desviación estándar de 0.0135 y la mediana en 0.8335.

La mejor solución ocurrió en la corrida 12, alcanzando una solución funcional en la generación 19 con 25 compuertas. El óptimo (con 7 compuertas) fue hallado en la generación 45. El diagrama lógico del circuito óptimo es mostrado en la Fig. 4.

La comparación de resultados se muestra en la Tabla I. El Diseñador Humano, utilizando mapas de Karnaugh y álgebra booleana para simplificar el circuito, produjo una solución con 12 compuertas. El Algoritmo Genético Binario logra, al igual que la Programación Genética Prefija, una solución con 7 compuertas. Observamos que las salidas de ambos circuitos son exactamente las mismas. Ambas técnicas evolutivas utilizaron el mismo número de iteraciones (400,000) para encontrar la solución óptima. Nótese que el algoritmo propuesto obtuvo una solución con menos compuertas que el diseñador humano.

B. Ejemplo 2

El segundo ejemplo consiste en un comparador de 4 entradas y 3 salidas, con la tabla de verdad que se muestra en la Fig. 5. Se realizaron 20 corridas con un tamaño total de población de 700 individuos y un número máximo de 2000 generaciones. Encontramos en el 60 % de las corridas circuitos funcionales y sólo en una ocasión se encontró una solución de 15 compuertas. La aptitud promedio de las 20 corridas nos da como resultado una media de 0.8226, con una desviación estándar de 0.0117 y la mediana en 0.8250.

La mejor solución obtenida ocurrió en la corrida 12, alcanzando una solución funcional en la generación 247 con 146 compuertas y encontrando una solución con 15 compuertas en la generación 586. El diagrama lógico del circuito de 15 compuertas es mostrado en la Fig. 5.

El resultado es comparado con el El Diseñador Humano 2 utiliza el método de Quine-McCluskey obteniendo 13 compuertas (Tabla II). Es claro el nivel de eficiencia que logra el MGA encontrando una solución de 9 compuertas con un alto grado de reutilización de componentes del circuito. La PG Prefija sólo logra llegar a una solución de 15 compuertas, con un grado muy bajo de reutilización de compuertas. Esta solución es inferior a la producida por el diseñador humano.

VI. CONCLUSIONES

Hemos mostrado que la técnica propuesta de PG para el diseño de circuitos lógicos logra producir diseños funcionales que resultan aceptables, pudiéndose encontrar soluciones lo bastante robustas como para ser consideradas como buenas. Asimismo, en ocasiones llegan a soluciones similares a las obtenidas por otras técnicas evolutivas o por diseñadores humanos usando los mapas de Karnaugh y el método de Quine-McCluskey.

Desafortunadamente los parámetros de tamaño de la población y número de generaciones van ligados a los problemas de dimensionalidad y complejidad del circuito que se desee encontrar. La versión actual del algoritmo tiene dificultades para

[5] Esta noción de "óptimo" se refiere a la mejor solución conocida para estos problemas.

Z	W	X	Y	F0	F1	F2
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	0
1	0	0	1	1	1	0
1	0	1	0	0	0	1
1	0	1	1	1	0	1
1	1	0	0	1	1	0
1	1	0	1	0	0	1
1	1	1	0	1	0	1
1	1	1	1	0	1	1

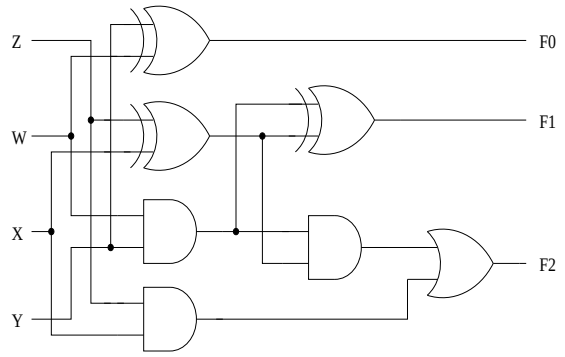

Fig. 4. Tabla de verdad y diagrama lógico del circuito funcional de menor tamaño encontrado por la PG Prefija para el primer ejemplo: un sumador de 2 bits.

Z	W	X	Y	F0	F1	F2
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	1	0	0

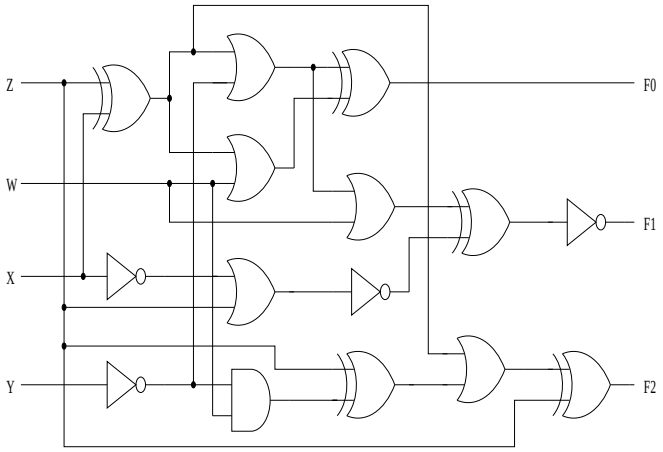

Fig. 5. Tabla de verdad y diagrama lógico del circuito funcional de menor tamaño encontrado por la PG Prefija para el segundo ejemplo: un comparador con 4 entradas 3 salidas.

resolver algunos circuitos de varias salidas, no así para los de una salida. Esto se debe principalmente a problemas ligados con la representación adoptada, que se encuentra aún en proceso de experimentación y depuración.

VII. Trabajos Futuros

Estamos trabajando en la inclusión de más funciones lógicas que nos permitan experimentar con nuevos componentes como los multiplexores, entre otros. Analizamos también la opción de implementar una nueva representación de los individuos que permita diseñar más fácilmente circuitos de múltiples salidas más eficientes y poder obtener una mayor reutilización y conexión de código "útil" entre las salidas de los circuitos buscando prescindir de las técnicas actuales para ello [11], [1].

VIII. Agradecimientos

El primer y segundo autor agradecen el apoyo al Consejo Nacional de Ciencia y Tecnología (CONACyT) a través del proyecto CONACyT No. J34900-A. El tercer autor agradece el apoyo del CONACyT a través del proyecto NSF-CONACyT 32999-A.

References

[1] Peter. J. Angeline. Genetic Programming and Emergent Inteligence. In Jr. Kenneth E. Kinnear, editor, *Advances in Genetic Programming*, chapter 4, pages 75–98. The MIT Press, Cambridge, Massachusetts, 1994.

[2] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming An Introduction*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1998. On the Automatic Evolution of Computer Programs and Its Applications.

[3] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1984.

[4] Carlos A. Coello, Christiansen Alan D., 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, ICANNGA'97*, pages 335–338, Norwich, England, April 1997. University of East Anglia.

[5] Carlos A. Coello Coello, Arturo Hernández A., 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.

[6] Nichael L. Cramer. A representation for the adaptive generation of simple sequential programs. In J.J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, pages 183–187, Pittsburgh, PA., 1985. Carnegie-Mellon University.

[7] George J. Friedman. Selective feedback computers for engineering synthesis and nervous system analogy. Master's thesis, University of California at Los Angeles, February 1956.

[8] Tatiana G. Kalganova. *Evolvable Hardware Design of Combinational Logic Circuits*. PhD thesis, Napier University, Edinburgh, Scotland, 2000.

[9] M. Karnaugh. A map method for synthesis of combinational logic circuits. *Transactions of the AIEE, Communications and Electronics*, 72(I):593–599, November 1953.

[10] Mike J. Keith and Martin C. Martin. Genetic Programming in C++: Implementation issues. In Jr. Kenneth E. Kinnear, editor, *Advances in Genetic Programming*, chapter 13, pages 285–310. The MIT Press, Cambridge, Massachusetts, 1994.

[11] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts, 1992.

[12] E. J. McCluskey. Minimization of boolean functions. *Bell Systems Technical Journal*, 35(6):1417–1444, November 1956.

[13] Julian F. Miller, Dominic Job, and Vesselin K. Vassilev. Principles in the evolutionary design of digital circuits-part I. *Genetic Programing and Evolvaable Machines*, 1(1/2):7–35, April 2000.

[14] Eduardo Serna Pérez, Katya Rodríguez Vázquez, and Carlos Coello Coello. Expresiones prefijas para el diseño de circuitos logicos utilizando programación génetica. In Carlos Zozaya, Marcelo Mejía, Pablo Noriega, and Alfredo Sánchez, editors, *Memorias del 3er Encuentro Internacional de la Computacion (ENC'01)*, volume 1, pages 95–104, Universidad Autonoma de Aguascalientes, Septiembre 2001.

[15] W. V. Quine. A way to simplify truth functions. *American Mathematical Monthly*, 62(9):627–631, November 1955.

[16] Katya Rodríguez-Vázquez and Peter J. Fleming. Functionality and Optimality in Circuit Design: A Genetic Programming Approach. In *Proceedings of the Third International Symposium on Adaptive Systems—Evolutionary Computation and Probabilistic Graphical Models*, pages 23–28, Havana, Cuba, March 19–23 2001. Institute of Cybernetics, Mathematics and Physics.