Expresiones Prefijas para el Diseño de Circuitos Lógicos utilizando Programación Genética

Eduardo Serna Pérez[1], Katya Rodríguez Vázquez[2] y Carlos A. Coello Coello[3]

[1]Sección Computación CINVESTAV-IPN México D.F., México.
eserna@computacion.cs.cinvestav.mx
[2]DISCA IIMAS-UNAM México D.F., México.
katya@uxdea4.iimas.unam.mx
[3]Sección Computación CINVESTAV-IPN México D.F., México.
ccoello@cs.cinvestav.mx

Resumen En este artículo proponemos el uso de la programación genética con una representación lineal prefija y un doble elitismo que permita diseñar (a nivel de compuertas), probar y evolucionar expresiones lógicas que finalmente produzcan un circuito lógico. En la implementación propuesta se realizan una serie de experimentos con distintos tipos de circuitos, los cuales varían en grado de complejidad y dimensionalidad.

1 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 [6] y el método de Quine-McCluskey [12,9].

Las técnicas evolutivas han comenzado a incursionar en el diseño de circuitos debido principalmente a su poder exploratorio, ya que permiten evaluar diversas regiones de un espacio de diseño y encontrar varias soluciones a problemas con un buen desempeño (*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. Con el propósito de demostrar el desempeño de los diseños producidos por nuestro programa, se utilizaron circuitos lógicos relativamente pequeños, cuyas soluciones están documentadas en la literatura especializada.

2 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 mínimo óptimo para un circuito cualquiera es desconocido. La diferencia de esta técnica con otras radica en la representación de los individuos y en la forma en cómo son evaluadas estas expresiones.

3 Antecedentes

El *Hardware Evolutivo* consiste en el diseño de circuitos electrónicos que son evolucionados a través de la simulación del proceso de selección natural [10]. George J. Friedman [4] fue uno de los primeros en aplicar técnicas evolutivas a la robótica. Friedman propuso un mecanismo para construir, probar y evaluar circuitos en forma automática, utilizando mutaciones aleatorias y procesos de selección.

La idea principal 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 linea (directamente evaluados en hardware reconfigurable), o fuera de linea (usando simulaciones)[5]. 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. Y otro dirigido a *nivel de funciones* que consiste básicamente en utilizar compuertas lógicas y a su vez con ellas construir nuevos módulos compuestos, como las ADF's entre otras [10,5].

Existe trabajo previo de diseño de circuitos combinatorios usando algoritmos genéticos como los de Coello *et al.* [2,3] que emplean una representación matricial del circuito y otros trabajos reportados en programación genética por Koza [8], Miller *et al.* [10] y Kalganova [5] que utilizan distintas representaciones de los circuitos. Algunas de estas técnicas por lo general necesitan un mapeo del cromosoma para evaluar un circuito.

4 Técnica Propuesta

En nuestra implementación [1] utilizamos cadenas prefijas [7] 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, 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

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

solución parcial al espacio de diseño deseado, obteniendo como resultado una expresión prefija que representa el diseño funcional de un circuito arbitrario.

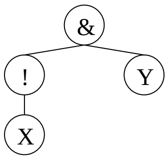
4.1 Representación de los individuos

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

genoma1: &X^|!ZWY genoma2: !Y

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 (Figura 1).

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



≡ &!XY

Figura1. Conjunto de funciones utilizadas en la representación de circuitos lógicos en PG Prefija. A la derecha 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.

4.2 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 Figura 2). En tal caso:

1. Si el carácter extraído es una letra, se le asignará un valor del tipo tabla de verdad[2]. 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.

[2] 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

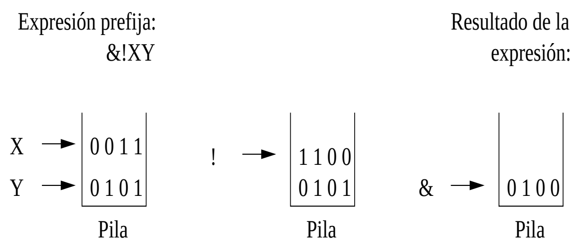


Figura2. Funcionamiento del intérprete de genomas. Nótese cómo lee carácter por carácter y como va solucionando el resultado de la expresión lógica.

4.3 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 patitud varia de 0 a 1.0).

4.4 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 de genoma 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.

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.

5 Experimentos

Se utilizaron varios circuitos lógicos de distinto grado de complejidad para probar la técnica propuesta, denominada Programación Genética Prefija (PG Prefija) [11]. Para el propósito de este artículo se seleccionaron 3 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) [2] y el Algoritmo Genético Multiobjetivo (MGA) [3], debido a que resultan bastante competitivos en lo que a diseño a nivel de compuertas se refiere. También se 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 varian de acuerdo a la complejidad y dimensionalidad de cada problema.

5.1 Ejemplo 1

El primer ejemplo, un circuito de 4 entradas y 1 salida. La Figura 3 muestra la configuración de la tabla de verdad. Se utilizó un tamaño total de población de 400 individuos y un máximo de 1000 generaciones. Obteniendo en un 80% de las corridas circuitos funcionales y el 20% de las ocasiones se encontraron soluciones robustas con 6 compuertas. La aptitud promedio de la población en las 20 corridas nos da como resultado una media de 0.8583 y una desviación estándar de 0.0082.

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

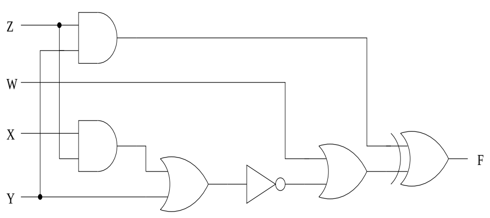


Figura3. Tabla de verdad y diagrama lógico del circuito funcional de menor tamaño encontrado por la PG Prefija para el primer ejemplo.

La mejor solución obtenida alcanzó un resultado funcional (aptitud de 1.0) en la generación 28 con 29 compuertas y encuentra uno de 6 compuertas en la generación 147. El diagrama lógico de esta solución es mostrado en la Figura 3.

El resultado (Tabla 1) es comparado con el obtenido por el Diseñador Humano que utiliza mapas de Karnaugh y álgebra booleana, encontrando una solución de 8 compuertas. El BGA encontró un circuito con 7 compuertas, generando circuitos funcionales el 40% de las veces en 20 corridas aleatorias. La PG Prefija encontró también circuitos funcionales de 7 compuertas en el 20% de las veces en 20 corridas, además del 20% logrado con circuitos de 6 compuertas.

Diseñador Humano	**BGA**
$F = (Y' + Z \oplus W)((ZX)' + W \oplus Y$	$F = (ZX + (W + Y))' + (ZY \oplus W)$
8 compuertas	7 compuertas
2 ANDs, 2 ORs, 2 XORs, 2 NOTs	2 AND, 3 ORs, 1 XOR, 1 NOT
PG Prefija	
$F = (YZ) \oplus (Y + (XZ))' + W$	
6 compuertas 2 ANDs, 2 ORs, 1 XORs, 1 NOTs	

Tabla1. Comparación de las mejores soluciones obtenidas por la PG Prefija, el Diseñador Humano y el BGA para el circuito del primer ejemplo.

5.2 Ejemplo 2

El segundo ejemplo consiste en un circuito de 5 entradas y 1 salida. La Figura 4 muestra su tabla de verdad. Se utilizó un tamaño total de población de 500 individuos y un número máximo de 1400 generaciones. El 75% de las corridas produjo circuitos funcinales y el 20% de las ocasiones arrojó soluciones robustas con 7 compuertas. La aptitud promedio de la población en las 20 corridas nos da como resultado una media de 0.8122 y una desviación estándar de 0.0178.

La mejor solución fue un circuito funcional en la generación 300 con 60 compuertas y encontrando uno con 7 compuertas en la generación 324. El diagrama lógico del circuito funcional de menor tamaño es mostrado en la Figura 4.

En la Tabla 2 se encuentra un comparativo de los resultados obtenidos. El Diseñador Humano utiliza mapas de Karnaugh y álgebra booleana, generando un circuito con 12 compuertas. Como observamos, nunca hace uso de la compuerta XOR ya que es dificil de visualizar, y que resulta de gran utilidad al momento de simplificar. El MGA encuentra una solución con 10 compuertas utilizando una población de 650 individuos y 2000 generaciones. La PG Prefija obtiene un circuito de menor tamaño usando tan solo 7 compuertas y utilizando un número menor de iteraciones, debido a la representación prefija.

5.3 Ejemplo 3

El tercer ejemplo consiste en un multiplicador de 2 bits que consta de 4 entradas y 4 salidas, con la tabla de verdad que se muestra en la Figura 5. Se utilizó un tamaño total de población de 300 individuos y un máximo de 400 generaciones. Encontramos en el 100% de las corridas circuitos funcionales y el 15% de las ocasiones se produjeron soluciones robustas con 7 compuertas. La aptitud promedio de la población en las 20 corridas nos da como resultado una media de 0.7741 y una desviación estándar de 0.0136.

V	Z	W	X	Y	F
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

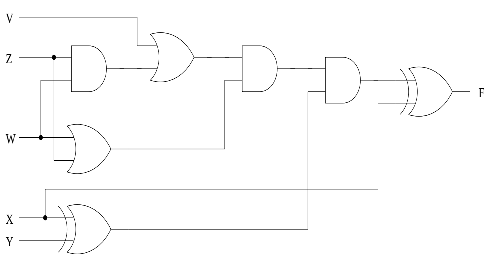


Figura4. Tabla de verdad y diagrama lógico del circuito funcional de menor tamaño encontrado por la PG Prefija para el segundo ejemplo.

Diseñador Humano
$F = X((Z+W)' + V'(ZW)') + Y(V(Z+W) + ZW)$
12 compuertas 5 ANDs, 4 ORs, 3 NOTs
MGA
$F = ((V+W)(V+Z)(X \oplus Y)(Z+W) \oplus X')'$
10 compuertas 3 ANDs, 3 ORs, 2 XORs, 2 NOTs
PG Prefija
$F = (V+WZ)(X \oplus Y)(Z+W) \oplus X$
7 compuertas 3 ANDs, 2 ORs, 2 XORs

Tabla2. Comparación de las mejores soluciones obtenidas por la PG Prefija, el Diseñador Humano y el MGA para el circuito del segundo ejemplo.

Z	W	X	Y	F0	F1	F2	F3
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	1	0	0	0
0	1	1	0	0	1	0	0
0	1	1	1	1	1	0	0
1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0
1	0	1	0	0	0	1	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	1	1	0	0
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

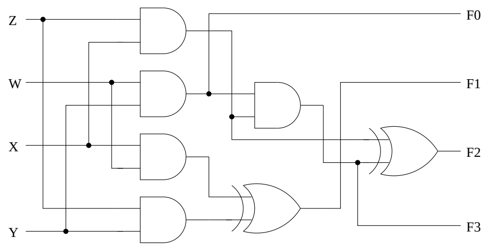


Figura5. Tabla de verdad y diagrama lógico del mejor circuito producido por la PG Prefija para el tercer ejemplo: el multiplicador de 2 bits.

La mejor solución fue un circuito funcional en la generación 38 con 14 compuertas y encontrando un circuito de menor tamaño en la generación 280 con 7 compuertas. El diagrama lógico de este circuito es mostrado en la Figura 5.

En la Tabla 3 mostramos los resultados obtenidos por PG Prefija y otras técnicas. El Diseñador Humano utiliza el método de Quine-McCluskey, generando una solución de 8 compuertas. Observamos que el MGA encuentra una solución óptima con 7 compuertas después de 325,000 evaluaciones. La PG Prefija utilizó 120,000 iteraciones para encontrar la solución de 7 compuertas que es igual a la obtenida por el MGA.

PG Prefija	**MGA**	**Diseñador Humano**
$F0 = WY$ $F1 = XW \oplus ZY$ $F2 = (WYXZ) \oplus XZ$ $F3 = WYXZ$	$F0 = WY$ $F1 = WX \oplus ZY$ $F2 = ZX \oplus (WYZX)$ $F3 = WYZX$	$F0 = WY$ $F1 = WX \oplus ZY$ $F2 = ZX(WY)'$ $F3 = WYZX$
7 compuertas 5 ANDs, 2 XORs	7 compuertas 5 ANDs, 2 XORs	8 compuertas 6 ANDs, 1 XOR, 1 NOT

Tabla3. Comparación de las mejores soluciones obtenidas por la PG Prefija, el Diseñador Humano y el MGA para el tercer ejemplo: el multiplicador de 2 bits.

6 Conclusiones

Hemos mostrado que la técnica propuesta de programación genética 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 superar a otras técnicas evolutivas, aunque en la mayoria de los casos nuestra técnica evolutiva basada en programación genética ha producido circuitos funcionales más compactos (en términos del número de compuertas) que los generados por diseñadores humanos usando los mapas de Karnaugh y el método de Quine-McCluskey, e incluso ha superado a otros algoritmos evolutivos diseñados para este dominio específico.

La creatividad, principalmente expresada en los circuitos de una salida, resulta muy interesante ya que se generan soluciones robustas de distinto grado de complejidad, número y tipo de componentes; es decir, se obtienen circuitos robustos que cumplen satisfactoriamente con una salida en particular con el mismo número de compuertas, pero de distinto tipo o distinta configuración. Esta pequeña evidencia, nos hace pensar en lo complicado y vasto que resulta ser el espacio de diseño, ya que no se puede asegurar que se tenga una solución única lo bastante robusta.

Además, se observó el potencial de la PG Prefija para "descubrir" nuevas equivalencias algebraicas, permitiendo la exploración de nuevos patrones de diseño. Se realizó un análisis de los resultados obtenidos y se encontraron algunas nuevas equivalencias que podrían ser muy útiles en la simplificación por otro sistema adicional, como uno basado en casos.

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. También encontramos que los circuitos de multiples salidas resultan muy complicados de solucionar por nuestra técnica, debido principalmente a su representación actual, que se encuentra en proceso de experimentación y depuración. Además el proceso de doble elitismo nos permite preservar buenos individuos y dirigir la busqueda.

7 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 circuitos de múltiples salidas más eficientes y poder obtener una mayor reutilización y conexión de código "útil" entre las salidas de los circuitos. En este sentido sería algo similar a utilizar pilas dinámicas, en lugar de las pilas estáticas que actualmente se emplean o quizás una adaptación del nodo raíz. También creemos conveniente agregar algún mecanismo que permita la creación de módulos reutilizables. Similar a las ADF's de Koza [8] y los MA de Angeline [1].

Finalmente se pretende desarrollar una nueva función de aptitud más robusta y versátil que permita emplear técnicas multiobjetivo, con el fin de poder llevar a cabo la evolución del circuito tomando en cuenta la funcionalidad, el número de compuertas, las reconexiones y la utilización de módulos. Esto nos podría llevar a un plano más ambicioso que sería poder crear circuitos a nivel de funciones. Además creemos importante tomar en cuenta aspectos más técnicos del diseño de circuitos como problemas en el retardo de señales, tolerancia a fallas, entre otras. Además pretendemos llegar a una mayor escalabilidad del problema.

8 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 primer autor agradece a CONACyT la beca otorgada para cursar la Maestría en Inteligencia Artificial de la UV-LANIA, a la Familia Salmerón Serna por las facilidades brindadas y a Nareli por su ayuda y suguerencias. El tercer autor agradece el apoyo parcial del CINVESTAV a través del proyecto JIRA'2001/08 y de CONACyT a través del proyecto NSF-CONACyT 32999-A.

Referencias

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. 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.
3. 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.
4. George J. Friedman. Selective feedback computers for engineering synthesis and nervous system analogy. Master's thesis, University of California at Los Angeles, February 1956.
5. Tatiana G. Kalganova. *Evolvable Hardware Design of Combinational Logic Circuits*. PhD thesis, Napier University, Edinburgh, Scotland, 2000.
6. M. Karnaugh. A map method for synthesis of combinational logic circuits. *Transactions of the AIEE, Communications and Electronics*, 72(I):593–599, November 1953.
7. 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.
8. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts, 1992.
9. E. J. McCluskey. Minimization of boolean functions. *Bell Systems Technical Journal*, 35(6):1417–1444, November 1956.
10. 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.
11. Eduardo Serna Pérez. Diseño de circuitos lógicos combinatorios utilizando programación genética. Master's thesis, Maestría en Inteligencia Artificial, Universidad Veracruzana, Sebastian Camacho 5, Xalapa, Veracruz México, Febrero 2001.
12. W. V. Quine. A way to simplify truth functions. *American Mathematical Monthly*, 62(9):627–631, November 1955.