Plataforma en Java para el Diseño de Circuitos Lógicos Combinatorios, Experimentando con Diferentes Operadores Genéticos.

Javier Cruz Pérez
Universidad Tecnológica de la Mixteca
jcruz@nuyoo.utm.mx,

Hilda Caballero Barbosa,
Universdad Tecnológica de la Mixteca
hilda@mixteco.utm.mx,

Carlos A. Coello Coello
Centro de Investigación y de Estudios Avanzados del I.P.N.
ccoello@cs.cinvestav.mx

Abstract

En el presente trabajo se muestran los avances obtenidos al aplicar un paradigma de la computación evolutiva, el Algoritmo Genético (AG), en el diseño de Circuitos Lógicos Combinatorios (CLC), con el propósito de minimizar el número de compuertas utilizadas. Los resultados mostrados en este documento son parte del trabajo desarrollado para obtener el título de Ingeniero en Computación, en el cual se pretende realizar una herramienta académica para visualizar el comportamiento de un AG en una aplicación específica.

1 Introducción

El diseño de los CLC inicialmente se resolvía por medio del álgebra booleana [8, 15], pero la minimización de éste depende de la capacidad de reconocer teoremas y postulados, siendo tedioso y propenso a errores [11, 12, 15]. Para incrementar la velocidad de reducir expresiones algebraicas, disminuyendo el grado de error, aparecieron otras técnicas, como los mapas de Karnaugh (sólo es posible utilizar este método hasta con seis variables [11, 12]), o el método de Quine-McCluskey, el cual además puede programarse [8, 11, 15]. Sin embargo, la complejidad de este problema es tal que ha permitido buscar soluciones por medio de otras técnicas. Una de éstas son los AGs, los cuales se basan en emular el proceso evolutivo para así hallar la solución óptima.

1.1 El Algoritmo Genético

Una definición formal de AG propuesta por John Koza [10] es la siguiente:
"El algoritmo genético es un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos (típicamente cadenas de caracteres de longitud fija que se ajustan al modelo de las cadenas de cromosomas), cada uno de los cuales se asocia con una aptitud, en una población nueva (es decir, la siguiente generación) usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y sobrevivencia del más apto y tras haberse presentado de forma natural una serie de operaciones genéticas (notablemente la recombinación sexual)". Los AGs ofrecen como ventaja la exploración eficiente en un amplio espacio de búsqueda, debido a sus operadores genéticos.

El esquema básico del funcionamiento de un AG simple es el siguiente [1]:

- Generar una población inicial de cromosomas (generalmente se efectúa de manera aleatoria).
- Calcular la aptitud de cada individuo.
- Seleccionar a los mejores individuos con base en su aptitud.
- Aplicar los operadores genéticos de cruza y mutación para generar la siguiente población.
- Reemplazar la población actual con la obtenida en la selección.
- Ciclar hasta que cierta condición se satisfaga.

1.2 Elementos básicos de un AG

Algunos de los *parámetros* del AG son: el tamaño de la población, la probabilidad de cruza y mutación, el número máximo de generaciones, entre otros.

Uno de los elementos básicos de un AG es la *representación* de los individuos o posibles soluciones. La representación tradicional es la binaria, en la cual un individuo o cromosoma es una cadena de la forma $\{b_1, b_2, \ldots b_m\}$, donde $b_1, b_2, \ldots b_m$, se denominan alelos (ya sean ceros o unos). Sin embargo, este método no mapea adecuadamente el espacio de búsqueda cuando se trata de números adyacentes en dicho espacio, por lo que los Códigos de Gray son una representación alternativa a este tipo de problemas.

Para *seleccionar* a los mejores individuos existen varios métodos, pero pueden clasificarse en:

- *Selección Proporcional*, propuestos por Holland [3, 7, 8, 15]. Los individuos se eligen en forma estocástica de acuerdo a su contribución de aptitud con respecto al total de la población. Existen diversos métodos, siendo los más comunes: Selección de Ruleta [3, 8, 15], Universal Estocástica, Sobrante Estocástico y Muestreo Determinístico [3, 5, 8, 15].

- *Selección Mediante Torneo*, propuesta por Wetzel [16]. Los individuos se redistribuyen de forma aleatoria y después compiten regularmente en parejas para ser seleccionados con base en comparaciones directas, siendo elegido aquél de aptitud más alta [3, 8, 15].

- *Selección de estado Uniforme*, propuesta por Whitley [17]. Este método es utilizado en los AGs no generacionales, en donde sólo son reemplazados algunos individuos en cada generación (los menos aptos).

La implementación de los operadores genéticos depende de la representación. Al optar por la representación binaria, se mencionarán únicamente los operadores de *cruza* y *mutación* más comunes de ésta.

Para realizar la *cruza* existen tres técnicas [3, 8, 15]:

- *Cruza de un punto*, en ella se selecciona un punto de manera aleatoria dentro del cromosoma para cada pareja de padres, y a partir del cual se intercambian los materiales genéticos, para generar nuevos descendientes [10].

- *Cruza de dos puntos*, es similar a la anterior, sólo que en vez de generarse un punto de cruza son dos [9].

- *Cruza Uniforme*, la cruza se realiza en *n* puntos, pero la diferencia con las anteriores radica en que nunca se fijan previamente los puntos de cruza [3, 8, 15]. Cabe mencionar que este operador no es muy utilizado.

La *mutación* es el operador genético que evita estancarse en óptimos locales, ya que permite realizar pequeños saltos en el espacio de búsqueda [3, 8, 15]. Por lo general es un operador Not, el cual es llamado mutación uniforme [8, 15]. También se tiene la mutación aleatoria, la cual consiste en seleccionar aleatoriamente un cero o un uno cada vez que haya sido verdadera la probabilidad de mutar, para después cambiar al gen por ese valor. El inconveniente de esta última técnica es que puede elegir un valor igual al que se pretende mutar y por tal motivo no lo modifica.

2 Trabajos Previos en el Diseño de CLCs

El diseño de los CLC ya ha sido explorado por medio de los AGs. Se encuentra el trabajo del Dr. Carlos A. Coello Coello [2], cuyo programa está hecho en C sobre la plataforma UNIX, y el del Maestro Eduardo Islas Pérez [8], quien utiliza además el razonamiento basado en casos para darle al programa memoria. También se encuentra el trabajo del Maestro Eduardo Serna Pérez [15], utilizando la Programación Genética Prefija.

Los programas del Dr. Coello y del Maestro Islas son de relativa facilidad en su uso (la inserción de los datos es por medio de un archivo o desde la línea de comandos). Lo que es complicado es la interpretación de los resultados, así como su transportación a otras plataformas, además no exploran el espacio de búsqueda más que con un solo tipo de operador de cruza y mutación, y utilizan sólo un operador de selección.

Se pretende que este trabajo permita una interpretación más sencilla, ya que la salida será gráfica y como el nombre lo sugiere, se realizará en Java para aprovechar las ventajas de este lenguaje de programación, puesto que se desea que el programa trabaje en Internet.

3 Desarrollo

3.1 ¿Por qué Java?

Para que el sistema trabaje en Internet es necesario que pueda ejecutarse en cualquier plataforma. Java se creó con la mente puesta en Internet, así que nos permite generar programas para trabajar en ella, con la certeza de que serán robustos y seguros. Además, Java necesita de un intérprete para ejecutar los programas, ya que los programas se compilan a bytecodes y después éstos son interpretados por la máquina virtual java (JVM). La interpretación implica que los programas sean dinámicos [4, 13, 14]. Esta característica de java resulta conveniente, puesto que permite trabajar con arreglos de tamaño variable dependiendo de los valores de entrada, y no con valores fijos predeterminados en el código.

3.2 El Problema a Resolver.

Para codificar el circuito se utilizó la misma representación configurada en el programa del Dr. Coello [2, 8]. De acuerdo a ésta, un circuito es representado a través de una matriz bidimensional, y cada elemento de la matriz representa a una compuerta.
Para representar a cada individuo se utilizó la representación binaria, ya que ésta ofrece un mayor número de esquemas posibles por bit de información [6].
Para la selección sólo se ha programado la selección mediante torneo. Mientras que para la mutación se implementaron la mutación aleatoria y la mutación uniforme.
En lo referente a los operadores de cruza, se programaron los de la cruza de uno y dos puntos y la cruza uniforme, y se agregó un operador más, al que se llamó cruza de múltiples puntos, el cual consiste en generar dos rangos aleatoriamente, y realizar el intercambio de material genético según los rangos. Este nuevo método tiene la característica de fijar los puntos de intercambio, como la cruza de uno y dos puntos. Sin embargo, el número de puntos de cruza es aleatorio y se generan múltiples puntos de cruza como en la cruza uniforme.

3.3 Pruebas

El programa fue probado con tres CLC, uno de tres entradas y una salida, un sumador y un multiplicador de dos bits. Recordemos que se agregaron dos operadores de mutación y cuatro de cruza, con sólo un operador de selección (selección por torneo). Esto hace necesario que el programa sea probado 8 veces para cada uno de los problemas planteados (cada operador de cruza se combina con los de mutación).

A	B	C	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

Tabla 1: Tabla de verdad del primer circuito.

Cruza	Mutación	Soluciones Factibles	Soluciones Optimas Optimas	La peor aptitud	Media
Dos Puntos	Aleatoria	100%	40.91%	26	28.14
	Uniforme	100%	45.45%	26	28.32
Un Punto	Aleatoria	100%	45.45%	26	28.14
	Uniforme	100%	59.90%	27	28.5
Uniforme	Aleatoria	100%	9.09 %	24	27.18
	Uniforme	100%	9.09%	23	26.91
Múltiples Puntos	Aleatoria	100%	36.36%	23	27.77
	Uniforme	100%	49.91%	27	28.32

Tabla 2: Tabla de Resultados generales

4 Resultados

Todos los resultados que se muestran a continuación fueron obtenidos después de haber ejecutado el programa 22 veces, cada una con una semilla diferente. Como las 22 veces se ejecutaron con cada una de las 8 combinaciones de los operadores genéticos propuestos, en realidad el programa se ejecutó 176 ocasiones por cada circuito (puede experimentar con él en la página http://nuyoo.utm.mx/~jcruz/Tesis.html).
La tabla de verdad del primer circuito (tres entradas y una salida) se visualiza en la tabla 1.

Para resolverlo se utilizó una matriz de 5x5, así como una población de 800 individuos, los cuales fueron generados y evaluados durante 1000 generaciones. Se usó una probabilidad de cruza de 0.5 y una de mutación de 0.00222. Los resultados más importantes se muestran en la tabla 2.

Como se puede observar en la mayoría de las combinaciones, el 100% de las ejecuciones fueron circuitos factibles. Al probar la cruza de uno, dos y múltiples puntos en combinación con las mutaciones aleatoria y uniforme, se obtuvieron en más del 30% de las ocasiones soluciones óptimas.

Es de llamar la atención que la cruza uniforme aporte pocas soluciones óptimas. Se dijo con anterioridad que la cruza uniforme no ha sido muy utilizada, y aquí se puede

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

Tabla 3: Tabla de verdad del sumador de 2 bits.

observar por qué ocurre esto.

Algunos de los diseños gráficos se muestran en las figuras 1 y 2. Como se puede observar, aunque tienen el mismo número de compuertas y del mismo tipo, la salida la da una compuerta Xor en el diseño 1 (figura 1), mientras que en el segundo (figura 2) es una And.

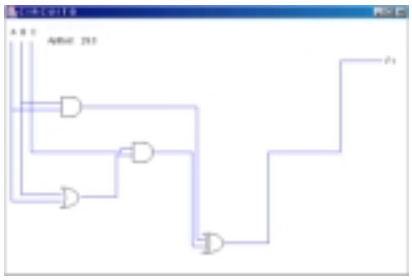

Figura 1: Un diseño del circuito.

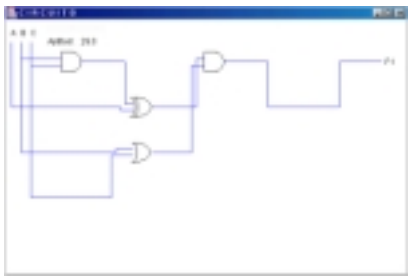

Figura 2: Otro diseño del circuito.

El segundo problema fue un sumador de 2 bits, cuya tabla de verdad se muestra en la tabla 3.

En este caso se utilizaron como parámetros del AG, una probabilidad de cruza de 0.5, una probabilidad de mutación de 0.00222, una población de 2000 individuos evaluados en 3000 generaciones. La matriz usada fue de 5x5. Como se observa en la tabla 4, al incrementar la dificultad del problema, la aportación de circuitos óptimos es menor.

Cruza	Mutación	Soluciones Factibles	Soluciones Optimas Optimas	La peor aptitud	Media
Dos Puntos	Aleatoria	90.90%	9.90%	45	61.55
	Uniforme	90.90%	13.63%	47	62.18
Un Punto	Aleatoria	86.36%	4.54%	45	61.55
	Uniforme	95.45%	4.54%	44	63
Uniforme	Aleatoria	0%	0 %	40	41.86
	Uniforme	0%	0%	40	41.86
Múltiples Puntos	Aleatoria	72.73%	13.63%	44	58359
	Uniforme	86.36%	13.63%	46	60.12

Tabla 4: Tabla de Resultados del sumador

En este caso es mucho más notoria la ineficiencia de la cruza uniforme. También se observa que la cruza de múltiples puntos da un mejor porcentaje de soluciones óptimas que la cruza de un punto y que la cruza de dos puntos.

Las figuras 3 y 4 nos presentan un mismo número de compuertas pero el diseño de la figura 3 maneja cuatro compuertas Xor, dos And y una Or, mientras que el diseño de la figura 4 utiliza sólo una And y una Or, pero cinco Xor.

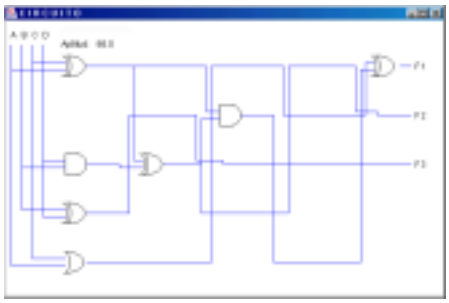

Figura 3: Un diseño del Sumador de dos bits.

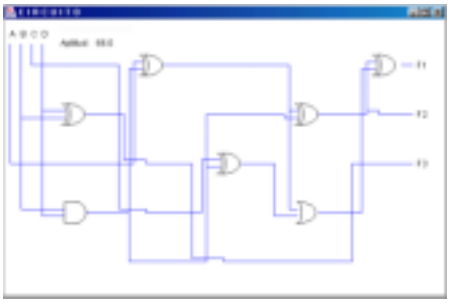

Figura 4: Otro diseño del Sumador de dos bits.

El último circuito probado fue el multiplicador de dos bits. En la tabla 5 se muestra su tabla de verdad.

Para ejecutar el programa con este circuito, se utilizó una población de 2000 individuos, en 3000 generaciones. La probabilidad de cruza y mutación son las mismas que en los dos ejemplos anteriores y también se utilizaron las mismas dimensiones de la matriz.

Los resultados se presentan en la tabla 6, donde podemos confirmar la poca aportación de la cruza uniforme, ya que el porcentaje de circuitos factibles es mínimo y en lo referente al porcentaje de circuitos óptimos es nula.

Analizando una vez más la tabla 6 se puede observar que la cruza de dos puntos aporta de una manera casi constante soluciones óptimas, y al combinarla con la

$\mathbf{A}_1$	$\mathbf{A}_0$	$\mathbf{B}_1$	$\mathbf{B}_0$	$\mathbf{C}_3$	$\mathbf{C}_2$	$\mathbf{C}_1$	$\mathbf{C}_0$
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

Tabla 5: Tabla de verdad del multiplicador de 2 bits.

Cruza	Mutación	Soluciones Factibles	Soluciones Optimas Optimas	La peor aptitud	Media
Dos Puntos	Aleatoria	72.73%	18.18%	60	75.14
	Uniforme	77.27%	18.18%	62	76.18
Un Punto	Aleatoria	72.73%	0%	61	75.18
	Uniforme	72.73%	9.09%	61	75.32
Uniforme	Aleatoria	9.09%	0 %	58	62
	Uniforme	0%	0%	60	60.96
Múltiples Puntos	Aleatoria	45.45%	4.54%	60	69.68
	Uniforme	50%	0%	61	70.86

Tabla 6: Resultados del multiplicador

mutación uniforme arroja un mayor número de soluciones factibles que la mutación aleatoria.

5 Conclusiones y Trabajo Futuro

Como se observó en la sección anterior la cruza de dos puntos en combinación con la mutación uniforme arroja un mayor número de soluciones factibles y óptimas, en comparación con las otras combinaciones. Por lo tanto, se podría concluir que ésta es la mejor combinación para la optimización de los circuitos lógicos cuando se aplica el paradigma de los AGs. Sin embargo puede tratarse de una conclusión un poco adelantada, ya que no se han probado los problemas cambiando los valores de otras variables, como son el tamaño de la población, el tamaño de la matriz y la probabilidad de cruza, así como también no se puede generalizar, por que sólo se experimentó con tres circuitos.

Como trabajo inmediato, se planean realizar pruebas modificando los parámetros de entrada, así como implementar otro operador de selección para poder estudiar el comportamiento del AG.

Agradecimientos

El primer agradece a CONACyT por la beca otorgada para realizar su tesis de licenciatura. El tercer autor agradece el apoyo proporcionado por CONACyT a través del proyecto 34201-A (del cual se derivó la beca del primer autor).

Bibliografía

[1] Bill P. Buckles and Fred E. Petry editors. Genetic Algorithms. IEEE Compurter Society Press, 1992.

[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] Nareli Cruz Cortés. Uso de Emulaciones del Sistema Inmune para Manejo de Restricciones en Algoritmos Evolutivos. Tesis de Maestría, Maestría en Inteligencia Artificial-UV/LANIA, Xalapa, Veracruz México, 2000.

[4] Agustín Froufe Quintas. Java 2 Manual de usuario y tutorial, segunda edición, Alfaomega Ra-Ma, pp. 8–11

[5] David E. Goldberg and Kalyanmoy Deb. A Comparison of Selection Schemes Used in Genetic Algorithms. In Gregory J. E. Rawlins, editor, Foundations of Genetic Algorithms, pp. 69–93, San Mateo, California, 1989. Morgan Kaufmann.

[6] David E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison–Wesley Publishing Co. Reading, Massachusetts, 1989.

[7] John H Holland. Adaptation in Natural an Artificial Systems. MIT Press, Cambridge, Massachusetts, second edition, 1992.

[8] Eduardo Islas Pérez. Development of a learning platform using case–based reasoning and genetic algorithms. Case study: Optimization of combinational logic circuits. Master thesis, Maestría en Inteligencia Artificial–UV/LANIA, Xalapa, Veracruz México, Noviembre 2000.

[9] A. K. de Jong. An Analysis of the Behavior of a Class of Genetic Adaptative Systems. PhD thesis, University of Michigan, Michigan, 1975.

[10] John R. Koza. Genetic Programing. On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, Massachusetts, 1992.

[11] M. Morris Mano. Diseño Digital. Prentice Hall. Edición en Español. México 1987.

[12] Victor P. Nelson, H. Troy Nagle, Bill D. Carroll, J. David Irwin. Análisis y Diseño de Circuitos Lógicos Digitales. Prentice Hall. México 1996.

[13] Constantino Sánchez Ballesteros. Java: Aprende a Programar Sólo Programadores, año VIII, Segunda época, número 90, Manual pp. 3.

[14] Herbert Schildt. Java 2: Manual de referencia, cuarta edición, McGraw-Hill/INTERAMERICANA DE ESPAÑA, S. A. U. España. pp. 13–145

[15] Eduardo Serna Pérez. Diseño de Circuitos Lógicos Combinatorios utilizando Programación Genética. Tesis de Maestría, Maestría en Inteligencia Artificial–UV/LANIA, Xalapa, Veracruz México, 2001.

[16] A. Wetzel. Evaluation of Efectiveness of genetic algorithms in combinational optimization. University of Pittsburgh, Pittsburgh (unpublished), 1983.

[17] Darrel Whitley. The GENITOR Algorithm and Selection Pressure: Why Rank–Based Allocation of Reproductive Trials is Best. In Proceedings of the Third International Conference on Genetic Algorithms, pp. 116–121, San Mateo, California, July 1989. Morgan Kaufmann Publishers.