# Número Mínimo de Compuertas del Circuito de Paridad Impar Usando el Algoritmo Genético en Java

I. C. Javier Cruz Pérez, M. C. Hilda Caballero Barbosa
Carrera de Licenciatura en Informática.
Universidad Regional del Sureste
68150 Oaxaca, Oaxaca
Tel. (01) 951 512-0252 e-mail jcptesis@hotmail.com

**RESUMEN**

En el presente trabajo se muestran los resultados obtenidos al aplicar un paradigma de la computación evolutiva, el algoritmo genético (AG), usando una herramienta ya existente, el algoritmo genético en Java (AG en Java), con la finalidad de diseñar circuitos lógicos combinatorios (CLCs), en particular los circuitos de paridad impar de 4, 5 y 6 variables, con el objetivo de determinar el número mínimo de compuertas necesarias para diseñarlos.

**Palabras clave:**
Diseño, circuitos lógicos, algoritmos genéticos.

## I. INTRODUCCIÓN

El diseño de los CLC inicialmente se resolvía por medio del álgebra booleana [1,2], pero la minimización realizada por ésta depende de la capacidad de reconocer teoremas y postulados, siendo un proceso tedioso y propenso a errores [2,3,4]. 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 [3,4]), o el método de Quine-McCluskey, el cual además puede programarse [1,2,3]. 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.

## II. EL ALGORITMO GENÉTICO

Los AGs encuentran la solución óptima simulando el proceso evolutivo. Éstos ofrecen como ventaja la exploración eficiente en un amplio espacio de búsqueda.
El esquema básico del funcionamiento de un AG simple es el siguiente [5]:

- Generar una población inicial de cromosomas.
- Calcular la aptitud de cada individuo.
- Seleccionar a los mejores individuos con base en su aptitud.
- Aplicar los operadores genéticos de cruza mutación.
- Reemplazar a la población actual con obtenida en la selección.
- Repetir el ciclo hasta que cierta condición se satisfaga.

Algunos de los parámetros del AG son: el tamaño de la población, la probabilidad de cruza (Pc), probabilidad de mutación (Pm), el número máximo de generaciones, entre otros.
La representación tradicional es la binaria. Sin embargo, este método no mapea adecuadamente espacio de búsqueda cuando se trata de números adyacentes en dicho espacio, la representación entera es una alternativa a este tipo de problemas.
Para *seleccionar* a los mejores individuos existen varios métodos, pero pueden clasificarse en:

- Selección Proporcional, propuestos por Holland [6].
- Selección Mediante Torneo, propuesta por Wetzel [7].
- Selección de estado Uniforme, propuesta por Whitley [8].

Para realizar la cruza existen tres técnicas:

- Cruza de un punto [9].
- Cruza de dos puntos [10].
- Cruza uniforme [1,2].

La mutación es el operador genético que evita estancarse en óptimos locales, ya que permite realizar pequeños saltos en el espacio de búsqueda [1,2,11]. Por lo general es un operador NOT, el cual es llamado mutación uniforme [1,2]. 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 permite elegir un valor igual que se pretende mutar y por tal motivo no lo modifica. La mutación uniforme y la aleatoria son las técnicas más comunes para este fin.

J. Díaz de León, G. González, J. Figueroa (Eds.): Avances en Ciencias de la Computación, pp. 171-175, 2003.

## III. TRABAJOS PREVIOS EN EL DISEÑO DE CLCs

El diseño de los CLCs ya ha sido explorado por medio de los Ags. Se encuentra el trabajo de Coello, Christiansen y Hernández [12], cuyo programa está hecho en C sobre la plataforma UNIX; el de Islas [1], quien utiliza además el razonamiento basado en casos para darle al programa memoria, y el AG en Java de Cruz [13], Y existen otras implementaciones pero utilizando otras técnicas, por ejemplo la de Serna [2] la cual utiliza programación genética prefija y la de Mendoza que utiliza el sistema de hormigas [14].

## IV. ¿PORQUÉ EL AG EN JAVA?

Este programa se encuentra basado en el trabajo de Coello, Christiansen y Hernández y para ambos programas se ha comprobado su efectividad en el diseño de algunos circuitos lógicos combinatorios [12,13]. Sin embargo, el AG en Java permite insertar el circuito a resolver, modificar parámetros y operadores por medio de una interfaz gráfica. Así como también el resultado (el diseño del circuito) se realiza de la misma manera. Y el AG en Java tiene implementado un mayor número de operadores genéticos y las compuertas NAND y NOR que cualquiera de las otras implementaciones del AG no tienen. Además el AG en Java se ejecuta vía internet (http://www.utm.mx/~jcruz/programa/programa.html). La figura 1 muestra la pantalla principal del sistema.

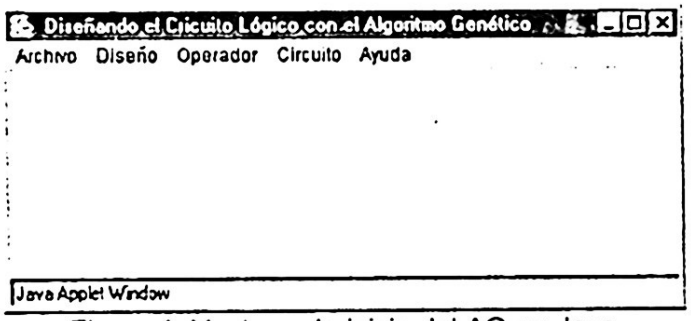


Figura 1. Ventana de inicio del AG en Java.

## V. EL PROBLEMA A RESOLVER

Como se había mencionado en el resumen, el interés se centrará en diseñar circuitos de paridad impar (de 4, 5 y 6 variables de entrada). Lo que motivó a este desarrollo, se basa en el hecho de que las técnicas tradicionales no logran simplificar este tipo de problemas, debido a que los valores de la función son islas de unos, y manejan sólo dos niveles para el diseño. La única forma de poder diseñarlos con un mínimo de compuertas es utilizando la compuerta XOR.

Los problemas de paridad impar son aquellos que permiten saber si la información enviada es correcta, la manera de saberlo es sumando el número de unos que contenga la información, y el último valor será uno si la suma es impar, y cero si es par.

Para el problema de paridad impar es posible determinar de cuantas compuertas estará compuesto su diseño, siempre que se utilice la compuerta XOR, esto por que el número mínimo de compuertas que puede tener el diseño es de n compuertas, donde n es el número de entradas. El problema de paridad impar se diseña con n-1 compuertas XOR y una NOT. Haciendo uso del programa AG en Java, se puede demostrar lo antes dicho (Fig. 2, 3 y 4) .

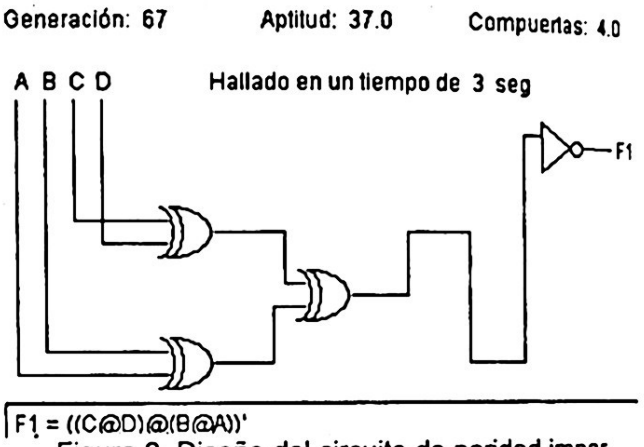


Figura 2. Diseño del circuito de paridad impar de cuatro entradas con XOR

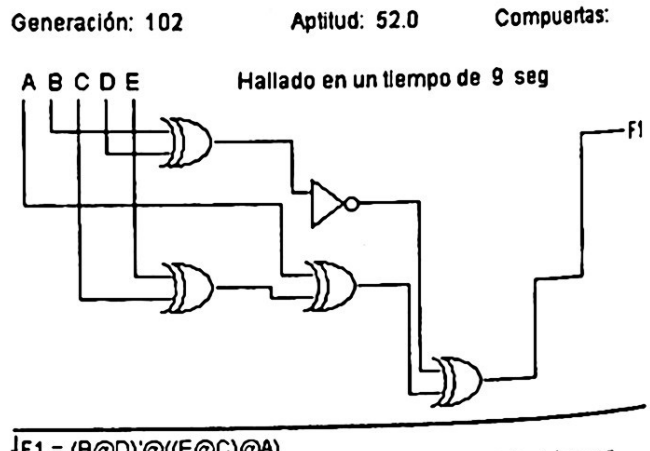


Figura 3. Diseño del circuito de paridad impar de cinco entradas con XOR.

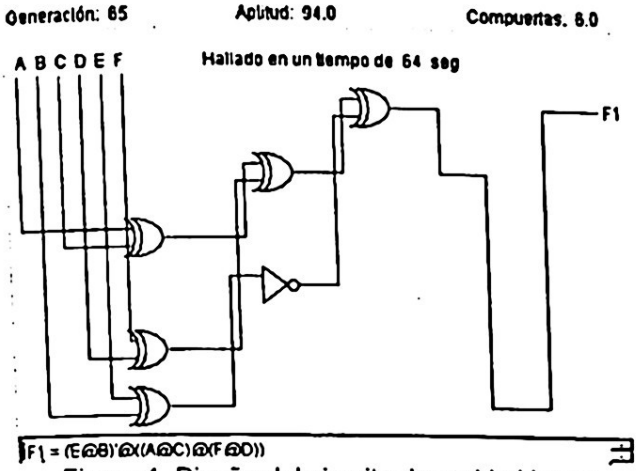


Figura 4. Diseño del circuito de paridad impar de seis entradas con XOR.

Sin embargo, al usarse otras compuertas para el diseño de la XOR es necesario determinar cual será el número mínimo de compuertas para poder generar el circuito.

Para lograr esto se utilizan las compuertas básicas AND y OR en conjunto con las compuertas universales (NAND y NOR).

Lo que se pretende comprobar es que el diseño de los circuitos que se está buscando estarán conformados de 3(n-1) compuertas, esto debido a que por medio del AG en Java se obtuvo el equivalente de la compuerta XOR con las compuertas que se piensa utilizar, el cual se muestra en la figura 5.

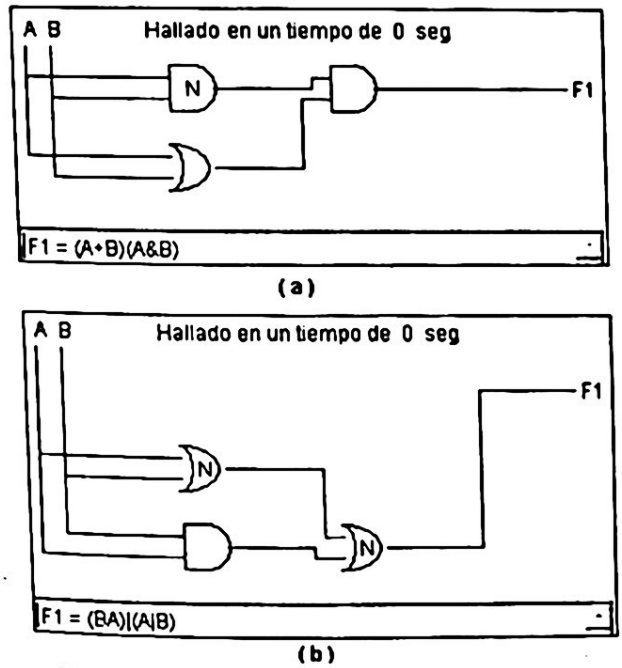


Figura 5. a) Una forma equivalente de diseñar la compuerta XOR. b) Otra forma de diseñar la XOR.

Y al aplicarse que cada compuerta XOR se conforma de 3 compuertas es posible intuir que el diseño será de 3(n-1) compuertas.

## VI. RESULTADOS.

Se realizaron las pruebas utilizando la representación entera, esto por que se ha visto en la práctica que es más rápida que la binaria. La probabilidad de cruza utilizada es de 0.5 debido a que es la usada en [12,13]. Para el tamaño de población se fijó en 2000 individuos por que se obtiene un mayor número de circuitos óptimos (se considerará un diseño óptimo aquel que tiene el menor número de compuertas encontradas) en un número pequeño de generaciones (menos de 500 iteraciones) [13]. Mientras que la probabilidad de mutación usada es de 0.5/L (donde L es la longitud del cromosoma) por que es la probabilidad que permite obtener un mayor número de circuitos óptimos [13].

Para el circuito de paridad impar de 4 entradas se utilizó como tamaño de población 1000 individuos, y tres tipos de cruza (cruza de dos puntos, cruza de un punto y cruza de múltiples puntos, éste último es un nuevo operador de cruza creado en [13]) dando un total de 30 ejecuciones y se usó una matriz de 5*5. El diseño que se muestra en la figura 6 se obtuvo en la generación 259 con los siguientes datos:

- Semilla inicial de 0.91
- Cruza de un punto

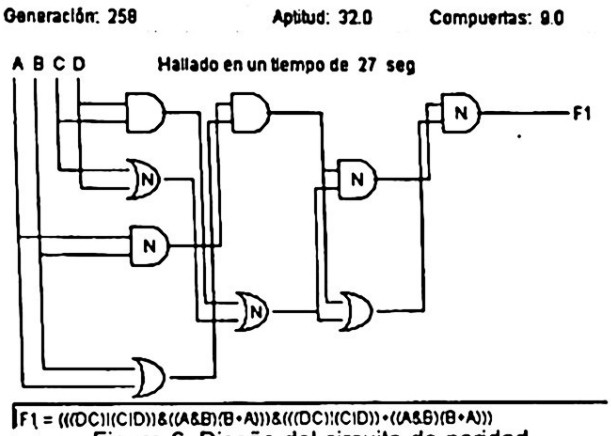


Figura 6. Diseño del circuito de paridad impar de cuatro entradas.

Podemos observar que este diseño tiene el número mínimo de compuertas calculadas por 3(n-1).

Es posible hacer una comparación contra un diseño obtenido por medio de el programa Electronics Workbench EDA versión 5.0a, aunque es una comparación un poco desleal por que el programa utiliza el método de Quine-McCluskey (solo genera

diseños con compuertas AND, OR y NOT), para este circuito obtiene 35 compuertas (4 NOTs, 7 Ors y 24 ANDs). Sin embargo, podemos utilizar el circuito de la Fig. 5 para generar un circuito solo con compuertas básicas (AND, OR y NOT) ya que una compuerta NAND o NOR se compone de una AND o un OR seguidas de un NOT, conociendo esto es posible afirmar que este circuito se compone de 4 ORs, 5 ANDs y 5 NOTs, dando un total de 14 compuertas, 11 compuertas menos que el obtenido por el programa Workbench. Claro que el diseño obtenido fue solo uno de las 30 ejecuciones.

Para el circuito de paridad impar de 5 entradas se utilizaron los datos descritos en un inicio, complementándolos con una matriz de 7*7 y los mismos tipos de cruza. El diseño que muestra la Fig. 7 se obtuvo con los siguientes datos:

- Cruza de dos puntos
- Semilla inicial de 0.364

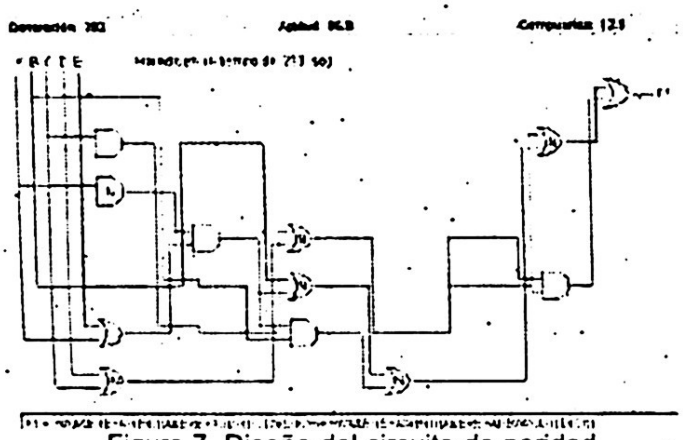

Figura 7. Diseño del circuito de paridad impar de cinco entradas

Se vuelve a obtener un diseño que cae dentro del cálculo de 3(n-1).

De igual manera como en el circuito anterior es posible determinar el número de compuertas básicas con las que se conforma este circuito, esto con la finalidad de compararlo con el obtenido por el programa Workbench. El circuito obtenido por el AG en Java se compone de 5 NANDs, 7 ORs y 6 NOTs, dando un total de 18 compuertas, mientras que el otro programa genera un diseño de 84 compuertas (5 NOTs, 15 ORs y 64 ANDs) 66 compuertas más que el obtenido por el AG en Java. Sin embargo, solo fue un diseño el logrado de 30 posibles. Aunque se obtuvieron 17 circuitos factibles (se toma como circuito factible aquel que cumple con el comportamiento dado por la tabla pero con un mayor número de compuertas que el óptimo [13]) de 30 posibles.

Para el problema de paridad impar de 6 entradas solo se utilizó la cruza de un punto y de dos puntos, pero se agrego la probabilidad de 1/L ya que con probabilidad inicial no fue posible encontrar el diseño óptimo, dando 40 ejecuciones. También que en los casos anteriores, solo fue uno de 40 posibles el diseño alcanzado, la figura 8 muestra este diseño.

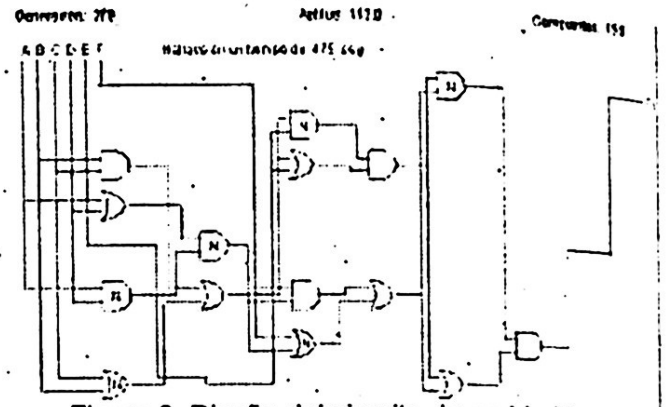

Figura 8. Diseño del circuito de paridad impar de seis entradas

El cual fue alcanzado en la generación 280 con los siguientes datos:

- Cruza de un punto
- Semilla inicial de 0.091
- Probabilidad de mutación de 1/L.

El programa Workbench obtiene un diseño de 197 compuertas (6 NOTs, 31 ORs y 160 ANDs), mientras que el AG en Java muestra un diseño de compuertas usando las compuertas universales y AND y la OR. Descomponiendo las compuertas universales se puede afirmar que el diseño conforma de 8 ANDs, 7 ORs y 6 NOTs, teniendo compuertas como total, mostrando 176 compuertas menos que el logrado por el Workbench.

El circuito óptimo del problema de paridad impar de 6 entradas (Fig. 8) muestra un diseño de 3(n-1) compuertas.

## VII. CONCLUSIÓN

Se demostró que el número mínimo de compuertas que puede tener este tipo de problemas haciendo del AG es de 3(n-1) cuando se utiliza las compuertas básicas AND y OR junto con las universales (NAND y NOR) y usando un diseño con más de dos niveles, superando a los mostrados por un programa que tiene el método de Quine-McCluskey como método simplificación. Y esto se puede generalizar para cuando se desee encontrar el número mínimo compuertas para un mayor número de entradas.

Como trabajo futuro se puede intentar demostrar cuando solo se utilizan las compuertas básicas, diseño se conformará de 4(n-1)+1 compuertas.

## REFERENCIAS

[1] E. Islas Pérez. "Development of a learning plataform using case-base reasoning and genetic algorithms. Case study: Optimization of combanitional logic circuits". Master thesis, Maestría en Inteligencia Artificial-UV/LANIA, Xalapa, Veracruz México, Noviembre 2000.

[2] E. Serna Pérez. "Diseño de Circuitos Lógicos Combinatorios Utilizando Programación Genética". Tesis de Maestría, Maestría en Inteligencia Artificial-UV/LANIA, Xalapa, Veracruz México, 2001.

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

[4] V. P. Nelson, H. Troy Nagle, B. D, Carrol y J. D. Irwin. "Análisis y Diseño de Circuitos Lógicos Digitales". Prentice Hall. México 1996.

[5] B. P. Buckles and F. E. Petry editors. "Genetic Algorithms". IEEE Computer Society Press, 1992.

[6] J. H. Holland. "Adaptation in Natural an Artificial Systems. MIT Press, Cambridge Massachussets, second edition, 1992.

[7] A. Wetzel. "Evaluation of Efectiveness of genetic algorithms in combinational optimization". University of Pittsburg, Pittsburg (unpublished), 1983.

[8] D. 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.

[9] J. R. Koza. "Genetic Programing. On the Programing of Computers by Means of Natural Selection". MIT Press, Cambridge, Massachusetts, 1992.

[10] A. K. De Jong. "An Analisys of the behavior of a Class of Genetic Adaptative Systems", PhD thesis, University of Michigan, Michigan, 1975.

[11] N. 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.

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

[13] J. Cruz Pérez. "Plataforma en Java para el Diseño de Circuitos Lógicos Combinatorios, Experimentando con Diferentes Operadores Genéticos". Tesis de Ingeniería, Ingeniería en Computación, Universidad Tecnológica de la Mixteca, Huajuapan de León, Oaxaca, México, 2003.

[14] B. Mendoza García. "Uso del Sistema de la Colonia de Hormigas para Optimizar Circuitos Lógicos Combinatorios". Tesis de maestría, maestría en Inteligencia Artificial, UV/LANIA, Xalapa, Veracruz, México, 2001.