Diseño Optimo de Circuitos Lógicos usando Algoritmos Genéticos

Carlos A. Coello Coello† **Alan D. Christiansen‡** **Arturo Hernández Aguirre‡**

† (coello@depauw.edu) 246 Julian Science Center, Department of Computer Science, DePauw University, Greencastle, IN 46135, USA

‡ ({adc,hernanda}@eecs.tulane.edu), 301 Stanley Thomas Hall, Department of Computer Science, Tulane University, New Orleans, LA 70118, USA

Resumen

En este trabajo se presenta una metodología, basada en el Algoritmo Genético (AG), para diseñar circuitos lógicos combinatorios en los cuales se trata de minimizar el número de compuertas utilizadas. Describiremos los problemas que encontramos al tratar de derivar una función de aptitud y una codificación cromosómica apropiadas, así como las limitaciones intrínsecas de la representación binaria en este dominio. Finalmente, también se incluirán algunos circuitos que nuestro programa ha producido, y que además de satisfacer las condiciones de optimalidad impuestas por el problema, parecen desafiar la habilidad de los diseñadores humanos más diestros por lo poco intuitivo de su diseño.

Palabras Clave: diseño de circuitos, optimización, algoritmos genéticos, diseño asistido por computadora, inteligencia artificial.

1 Introducción

El diseño es una actividad que se caracteriza por requerir una cantidad considerable de creatividad y conocimiento, que son atributos normalmente asociados con los humanos. La definición misma del término *diseño* parece un tanto elusiva, puesto que éste puede interpretarse de varias maneras distintas, dependiendo del contexto en el que se aplique. La definición que se ajusta mejor a los propósitos de este artículo es la siguiente:

> Diseño es el proceso de derivar, a partir de un comportamiento de entrada/salida específico, una estructura (en nuestro caso una cierta combinación de compuertas lógicas) que es funcional (es decir, que produce todas las salidas deseadas para todas las entradas especificadas), satisfaciendo a la vez una serie de restricciones pre-establecidas.

Además, queremos que este diseño sea óptimo en términos de ciertas características estructurales (por ejemplo, el número de compuertas utilizadas). Este proceso de diseño es muy tedioso y propenso a errores, y normalmente requiere de una cantidad considerable de experiencia humana. Aunque han habido varios intentos de desarrollar programas para diseño automatizado, tales programas son notablemente difíciles de construir.

El objetivo principal de la investigación reportada en este artículo es producir una herramienta computarizada que pueda hacer del proceso de diseño una tarea menos tediosa para los humanos, sin sacrificar la calidad del producto final. En esta primera etapa de nuestra investigación, nos hemos enfocado a la solución de circuitos lógicos combinatorios que no tienen elementos con memoria ni rutas de retroalimentación, aunque esas extensiones serán incorporadas en nuestro programa en el futuro. A fin de demostrar la eficiencia de los diseños producidos por nuestro programa, éste ha sido utilizado para generar circuitos relativamente sencillos, que se utilizan tradicionalmente en cursos de Organización de las Computadoras y Lógica Digital, y cuyas soluciones están bien documentadas. Como se verá más adelante, las soluciones que hemos producido en todos los casos, son mejores que las funciones Booleanas existentes en la literatura, pero a la vez son altamente no intuitivas para un diseñador humano, debido al hecho de que nuestro programa hace un uso frecuente de las compuertas XOR en sus soluciones. Este y algunos otros aspectos interesantes del diseño de los circuitos producidos por el algoritmo genético serán discutidos brevemente en una sección posterior de este artículo.

2 Trabajo Previo

La técnica general de búsqueda inspirada por la evolución natural, denominada *algoritmo genético* [Holland, 1992], ha sido ampliamente utilizada para tareas de optimización [Goldberg, 1989], y es bien sabida su eficiencia en un buen número de dominios diferentes. En este trabajo, tratamos de utilizar al algoritmo genético (AG) como una herramienta de diseño, con un particular énfasis en la producción de circuitos combinatorios.

El proceso de diseño de circuitos lógicos combinatorios ha evolucionado desde sus primeras nociones [Shannon, 1938] hasta convertirse en un elemento estándar de la educación básica de un estudiante de licenciatura en

las carreras de computación en todas partes del mundo [Roth, 1992]. En la actualidad, la metodología tradicional en las universidades consiste en enseñar el uso de las ayudas gráficas de diseño tales como los mapas de Karnaugh [Karnaugh, 1953] [Veitch, 1952]. Por otro lado, las herramientas más adecuadas para ser implementadas en una computadora han evolucionado del Método de Quine-McCluskey [Quine, 1955] [McCluskey, 1956] a las herramientas de dominio público tales como Espresso [Brayton *et al.*, 1984] y MisII [Brayton *et al.*, 1987], así como un buen número de productos comerciales.

Louis [Louis, 1993] es una de las pocas fuentes bibliográficas que pudimos encontrar en la que el uso del algoritmo genético estuviera enfocado al diseño de circuitos lógicos, y no a la simple optimización del ordenamiento de un grupo de variables que sirven como un paso adicional para el uso de una metodología determinística de diseño [Thomson and Miller, 1996]. Nuestro objetivo ha sido no utilizar ningún software adicional para el análisis o la simplificación de la función Booleana resultante, sino más bien utilizar al algoritmo genético para la doble tarea de generar soluciones válidas (es decir, circuitos funcionales) y además óptimos en términos del número de compuertas que los integran.

La disertación de Louis [Louis, 1993] propone una combinación de sistemas expertos con el algoritmo genético, a través de un operador denominado *cruza con máscara*, el cual se adapta a la codificación existente, pudiendo explotar información no utilizada por los operadores de cruza clásicos. Sus resultados, aunque muy motivantes en algunas instancias, no parecen haber resuelto el problema del diseño de circuitos completamente, pues su metodología requiere de restricciones adicionales para producir resultados satisfactorios. No obstante, su idea de incorporar conocimiento acerca del dominio en los operadores genéticos constituye un gran paso en cuanto al mejoramiento del desempeño de esta heurística cuando se utiliza como una herramienta de diseño. Desafortunadamente, esta incorporación de conocimiento es un tanto dilemática, porque reduce la utilidad del algoritmo genético como una técnica de búsqueda más *general*. Louis resuelve este conflicto mediante la definición de un operador que él afirma es independiente del dominio, pero cuya eficiencia resulta estar en función de la representación utilizada [Louis, 1993].

La generación de funciones Booleanas parece encajar más dentro del dominio de la programación genética que en el del algoritmo genético, y por ello no debe resultar sorprendente que el mismo Koza [Koza, 1992] haya abordado este problema. En su propuesta original, sin embargo, Koza diseña circuitos para funciones muy simples (por ejemplo un sumador de 2 bits) utilizando pocas compuertas (AND, OR y NOT), enfatizando únicamente el aspecto funcional de la solución, y no el de optimización. Los circuitos producidos en consecuencia son altamente redundantes, pero a la vez difíciles de analizar por un humano. Trabajo más reciente de Koza [Koza *et al.*, 1996b] [Koza *et al.*, 1996a] se enfoca más hacia el diseño de circuitos analógicos en los cuales la meta es producir la topología y el tamaño apropiados del circuito deseado para todos los componentes dados. Indudablemente, la representación de árboles utilizada por la programación genética es más apropiada y poderosa para el diseño en general, pero a la vez dificulta la optimización de los diseños resultantes, además de requerir recursos de cómputo normalmente considerables para producir soluciones satisfactorias. De ahí nuestra decisión de intentar reducir el sesgo que impone la representación lineal del algoritmo genético tradicional para poder generar expresiones Booleanas en un tiempo relativamente corto, pero a la vez manteniendo las ventajas que dicha representación presenta para llevar a cabo optimizaciones.

3 Planteamiento del Problema

El problema que nos interesa resolver consiste en diseñar un circuito que realice una cierta función (especificada mediante una tabla de verdad), utilizando un cierto número de compuertas disponibles. En este dominio, es posible definir varios criterios de optimalidad. Por ejemplo, desde una perspectiva puramente matemática, podríamos minimizar el número total de literales, o el número total de operaciones binarias, o el número total de símbolos en una expresión. En cualquiera de estos casos, el problema de minimización resultante es, no obstante, bastante complejo. Por otra parte, si consideramos el circuito como una red de compuertas, entonces tiene sentido utilizar como criterio de optimización el número de componentes que la integran, imponiendo como restricciones adicionales el número máximo de niveles y/o de paquetes que pueden utilizarse para el diseño. En general, resulta muy difícil encontrar estas redes mínimas de manera directa, o incluso demostrar la optimalidad de una red de compuertas cualquiera [Brzozowski and Yoeli, 1976]. De tal manera, podemos decir que aunque es posible resolver de manera sistemática una serie de problemas de minimización relacionados con circuitos eléctricos, en general nos tenemos que conformar con resolver instancias específicas de diseño, o bien recurrir a la intervención humana.

La complejidad de un circuito lógico es una función de su número de compuertas. La complejidad de una compuerta es generalmente una función del número de entradas que llegan a ella. Debido a que un circuito lógico es la implementación de una función Booleana en hardware, la reducción del número de literales en la función debiera reducir el número de entradas a cada compuerta y el número total de compuertas del circuito, reduciendo en consecuencia su complejidad. El método algebraico usado tradicionalmente para minimizar funciones es tedioso y propenso a errores. Su éxito depende de nuestra habilidad para reconocer la aplicación de un teorema o un postulado durante el proceso de min-

imización, y tal inferencia puede resultar poco obvia en algunas ocasiones. Además, no existe ningún conjunto de reglas específicas (o un algoritmo determinístico) que sirva como ayuda en este proceso de minimización. Dos técnicas populares de minimización son el *Mapa de Karnaugh* [Karnaugh, 1953], que se basa en una cierta representación gráfica de las funciones Booleanas, y el *Procedimiento de Quine-McCluskey* [Quine, 1955; McCluskey, 1956], que es un método tabular. Los Mapas de Karnaugh son útiles para minimizar funciones con un máximo de cinco o seis variables. El Procedimiento de Quine-McCluskey es útil con funciones de cualquier número de variables y puede programarse fácilmente en una computadora. Ambos métodos son eminentemente mecánicos, y por lo general ambos producen varias funciones mínimas diferentes para la misma tabla de verdad, dependiendo del orden en que se apliquen ciertos pasos durante el proceso de diseño. Dado que todas las funciones mínimas con el mismo número de literales producen circuitos de la misma complejidad, cualquiera de ellas puede seleccionarse como un diseño válido. Tanto el Mapa de Karnaugh como el Procedimiento de Quine-McCluskey producen circuitos de *dos niveles* (por ejemplo, la suma mínima de productos). Este tipo de diseño es el mejor si nuestra preocupación principal es minimizar el retraso de la propagación de las señales a través del circuito. Sin embargo, en muchas ocasiones resulta más importante minimizar el número de compuertas presentes en el circuito, y se considera entonces aceptable una pequeña penalización en su velocidad. Para minimizar el número total de compuertas, normalmente se requiere producir un diseño de varios niveles. Para poder producir diseños de varios niveles con el Mapa de Karnaugh y el Procedimiento de Quine-McCluskey, es necesario combinar estos métodos con técnicas como la manipulación algebraica de expresiones.[1]

Es importante hacer notar que el proceso de simplificación algebraica depende totalmente de la familiaridad del diseñador con los postulados y teoremas del álgebra Booleana y de su capacidad para reconocer la mejor forma de poder aplicarlos. Obviamente, dichas habilidades varían de individuo a individuo, y por tanto producir diseños optimizados no es una tarea fácil. Además, la secuencia en la que se aplican los teoremas y postulados también afecta el resultado final, y es posible producir más de un diseño final a partir de la misma expresión original. Dado que normalmente todos estos diseños son válidos y aceptables, no hay (en general) un diseño mínimo único de ninguna expresión Booleana.

En este trabajo intentamos comparar los diseños producidos por un algoritmo genético con los que produce un humano usando mapas de Karnaugh. Esta comparación es un tanto injusta porque resulta difícil para un humano producir el tipo de diseños que en muchas ocasiones generará el algoritmo genético. Esto se debe sobre todo al hecho de que (como veremos más adelante) el algoritmo genético que utilizamos tiende a hacer uso frecuente de la compuerta XOR, mientras que los humanos tienden a usar casi exclusivamente las compuertas más tradicionales (i.e., AND, OR, NOT) en sus diseños a menos que el uso de la compuerta XOR parezca bastante "obvio". Con la representación cromosómica utilizada, resulta relativamente fácil producir diseños con compuertas XOR anidadas en dos o tres niveles consecutivos, los cuales son virtualmente imposibles de "visualizar" por un humano.

[1] Una herramienta como MissII [Brayton *et al.*, 1987] *puede* producir diseños de varios niveles, pero requiere de intervención humana para lograrlo de manera efectiva.

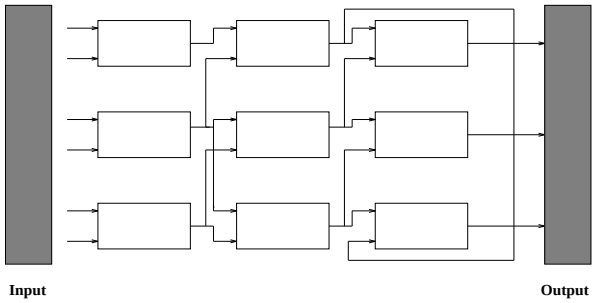

Figura 1: Un circuito es representado como una matriz bidimensional de compuertas, en la cual cada elemento de la matriz obtiene sus entradas de cualquier elemento de la columna anterior.

3.1 Representación de un circuito

Para codificar un circuito utilizamos una representación de acuerdo a la cual un circuito es una matriz bidimensional en la que cada elemento es una compuerta (hay cinco tipos de compuertas: AND, NOT, OR, XOR y WIRE) que recibe sus dos entradas de cualquier compuerta de la columna anterior (ver Figura 1). Nuestro objetivo es producir un diseño funcional (i.e., que produzca todas las salidas esperadas para cualquier combinación de entradas de acuerdo a la tabla de verdad del problema) que maximice el uso de la compuerta WIRE.[2] Más formalmente, podemos decir que cualquier circuito puede representarse como un arreglo bidimensional de compuertas $S_{i,j}$. El índice j indica el *nivel* de la compuerta, de manera que las que están más cerca a las entradas tienen los valores más bajos de j. (Los valores del nivel se incrementan de izquierda a derecha en la Figura 1.) Para una j fija, el índice i varía con respecto a las compuertas que están "próximas" una a otra en el circuito, pero sin estar necesariamente conectadas entre sí.

[2] WIRE indica básicamente una operación nula, o sea la ausencia de compuerta, y se utiliza sólo para mantener regularidad en la representación utilizada por el algoritmo genético.

4 Uso del Algoritmo Genético

Nuestro tratamiento inicial del problema fue intentar reproducir los resultados presentados por Louis en su disertación [Louis, 1993] para el algoritmo genético tradicional. Sin embargo, la función de aptitud que Louis sugiere en su tesis doctoral no funcionó en ninguno de los problemas que intentamos, incluyendo el utilizado por él mismo para ilustrar su técnica.[3] Tras experimentar con un sinnúmero de diferentes parámetros, fuimos incapaces de lograr convergencia, y desistimos de usar la función de aptitud sugerida por Louis [Louis, 1993] optando mejor por una función que parece proporcionar más información al algoritmo genético para guiar la búsqueda. En vez de utilizar el número de operaciones efectuadas correctamente por el circuito codificado en un cromosoma [Louis, 1993], utilizamos una comparación bit por bit del resultado producido para una cierta combinación de entradas con respecto a las salidas esperadas. Por cada error, la función de aptitud penaliza el valor total. Posteriormente, si la solución producida corresponde a un circuito funcional (i.e., no hay errores en ninguna de las salidas), entonces la función de aptitud "premia" la solución por cada WIRE que contenga, a fin de alentar los diseños que usen menos compuertas. Esta función de aptitud en dos etapas parece proporcionar resultados altamente satisfactorios, como veremos más adelante.

Cabe aclarar que la función de aptitud no es la única diferencia entre nuestra técnica y la de Louis, porque aunque ambos planteamos este problema como uno de objetivos múltiples (i.e., encontrar un circuito funcional, y que además tenga el número mínimo de compuertas), en el caso de Louis se propone el uso de un operador de cruza especial combinado con un sistema de razonamiento basado en casos (*case-based reasoning*) que permita obtener información previa sobre las mejores soluciones producidas a lo largo de la ejecución del algoritmo genético, sin embargo, como los resultados de su disertación lo indican (ver página 67 de [Louis, 1993]), este enfoque no produce soluciones que sean mejores que la nuestra para el problema del sumador de 2 bits que presentaremos más adelante, el cual es el único problema de circuitos resuelto en la disertación de Louis.

La codificación representa otro aspecto interesante de este problema, y en este caso sí optamos por recurrir a la técnica sugerida por Louis [Louis, 1993], de acuerdo a la cual una cadena cromosómica contiene el tipo de compuerta y sus dos entradas de manera consecutiva, tal y como se muestra a continuación:

$$cromosoma = tipo\_de\_compuerta|entrada\_1|entrada\_2 \quad (1)$$

donde *tipo_de_compuerta* es un entero entre 0 y 4 (0 =AND, 1 =NOT, 2 =OR, 3 =WIRE y 4 =XOR), y *entrada_1* y *entrada_2* son un entero entre 1 y el número de filas permitidas en la matriz (valor constante durante la ejecución del algoritmo genético). El orden de codificación de la matriz fue columna principal, empezando con la posición $S_{1,1}$ (el elemento superior izquierdo), seguida del elemento que se encuentra abajo de éste (i.e., el elemento $S_{2,1}$), y así sucesivamente. Es interesante advertir que si las compuertas se codifican usando un orden de fila principal, el problema se vuelve disruptivo [Louis, 1993], tornándose sumamente difícil para el algoritmo genético. La razón es que usando esta otra forma de codificación, cualquier grupo de circuitos que esté cercano en el espacio fenotípico bidimensional podrá estar muy lejos en el espacio genotípico unidimensional, lo cual dificulta la preservación de esquemas muy aptos.

Una parte fundamental de lograr que el algoritmo genético produzca diseños óptimos (o semi-óptimos) es la representación cromosómica utilizada. Aunque se ha argumentado que una representación binaria proporciona el número máximo de esquemas [Michalewicz, 1992], existe evidencia de que en algunos dominios tales como el de la optimización numérica, los alfabetos de cardinalidad más alta proporcionan mejores resultados en un período de tiempo más corto que sus contrapartes binarias [Coello, 1996]. Con este precedente en mente, decidimos experimentar con un alfabeto de cardinalidad n, donde n es un valor definido por el usuario, representando en este caso el número de filas que tendrá el circuito, de acuerdo a la representación matricial presentada anteriormente. Esta representación permite la manipulación de cadenas más cortas, decrementando la complejidad de la decodificación, y al mismo tiempo permite explorar regiones del espacio de búsqueda que la representación binaria no parece cubrir apropiadamente cuando la distancia fenotípica es muy pequeña, como en este caso [Coello, 1996].

Estimar el tamaño del espacio de búsqueda resulta un tanto engañoso a primera vista, y algunos pueden asumir erróneamente que este problema es trivial. Sin embargo, si analizamos la representación utilizada, podemos fácilmente deducir que para una cardinalidad n, y asumiendo una longitud cromosómica l, el tamaño intrínseco del espacio de búsqueda es de n^l. La cardinalidad y la longitud de la cadena son una función directa del tamaño total de la matriz: l=3 $\times$ t, donde $t = r \times q$, siendo r y q el número de filas y columnas de la matriz, respectivamente. Para darnos una idea más clara del significado de estas fórmulas, podemos analizar el multiplicador de 2 bits que resolvemos en el cuarto ejemplo presentado en la siguiente sección. En este caso, la longitud cromosómica es de 96 ($r = 8, q = 4, t = 32, l = 96$), y la

[3] Debe mencionarse que no existe evidencia convincente de que la técnica de Louis realmente funcione, porque los resultados presentados en su disertación (ver, por ejemplo, la gráfica de convergencia de la página 67 [Louis, 1993]) parecen indicar que se logran soluciones con una aptitud máxima de 41 después de 50 generaciones para el problema de un sumador de 2 bits en el que una aptitud de 44 (de acuerdo a su propia métrica) debiera indicar un diseño funcional.

cardinalidad es 8, por lo que el tamaño intrínseco del espacio de búsqueda es de aproximadamente 5×10^{86}. Si pudiésemos evaluar 1 millón de diseños por segundo, nos tomaría aproximadamente 1.58×10^{73} años evaluar todos los circuitos posibles que se podrían codificar para este problema con esta representación. Nuestro algoritmo genético, en contraste, sólo requirió evaluar unos 800000 diseños en cada caso para producir los resultados presentados en este artículo.

5 Comparación de Resultados

Aunque hemos utilizado un buen número de circuitos de diferentes grados de complejidad para probar nuestra técnica, sólamente incluiremos 4 ejemplos para ilustrar la eficacia del algoritmo genético. Estos ejemplos han sido tratados extensamente en la literatura, y además han sido utilizados como ejercicios en clase y/o tareas en al menos 2 cursos de organización de computadoras y 2 cursos de lógica digital impartidos por dos de los autores. De tal forma, los resultados contra los que se compararán los diseños producidos por el algoritmo genético son producto de un análisis muy cuidadoso efectuado por varios diseñadores experimentados, que han intentado obtener las expresiones mínimas posibles aplicando las técnicas convencionales de álgebra Booleana a las expresiones obtenidas como resultado de aplicar mapas de Karnaugh a cada una de las tablas de verdad de las funciones seleccionadas como ejemplos. Es importante mencionar que estos diseños han sido revisados en múltiples ocasiones, con técnicas más sofisticadas, y que en algunos casos (como en el ejemplo 2), el circuito fue comparado contra el mejor resultado producido por un algoritmo especial de simplificación que presenta su mejor comportamiento con dicho ejemplo.

Los ejemplos mostrados a continuación fueron resueltos utilizando el programa MOSES ([Coello, 1996]), que trata al problema como si tuviera objetivos múltiples. Cada uno de los ejemplos presentados a continuación fue ejecutado 81 veces en un ciclo anidado (ciclando la mutación y la cruza de 0.1 a 0.9, a intervalos de 0.1, y con una población fija de 3000 cromosomas, y un número máximo de generaciones = 200, en conformidad con los parámetros sugeridos en [Coello, 1996]. El tamaño de población está realmente en función del tamaño de la matriz utilizada para representar al circuito. Como esta matriz se consideró de tamaño ≤ 32 en todos los casos, entonces la población se manejó como una constante (3000 cromosomas). Experimentos más exhaustivos con los parámetros utilizados en este problema demuestran que es sumamente sensible a los valores de la cruza y la mutación, y que presenta sus mejores resultados en una sola corrida utilizando un porcentaje de cruza de 0.5 y uno de mutación de 0.3, siempre y cuando el tamaño de la población esté por encima de los 2500 cromosomas. Los detalles de los experimentos y sus resultados se encuentran en [Coello, 1996]. MOSES sólo requiere como entrada la tabla de verdad del circuito y las dimensiones de la matriz que se desea utilizar, y el programa se encarga de traducir automáticamente dicha información en una representación apropiada para el algoritmo genético.

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

Tabla 1: Tabla de verdad del circuito del primer ejemplo.

5.1 Ejemplo 1

Nuestro primer ejemplo es una función que tiene sólo 3 entradas y una salida, tal y como se expresa en la Tabla 1. En este caso, la matriz utilizada fue de 4×4, la longitud cromosómica fue de 48 ($r = 4, q = 4, t = 4 \times 4 = 16, l = 3 \times t = 48$).

Las mejores soluciones producidas por un diseñador humano, y su comparación con las producidas por el algoritmo genético (con una población de 1000 cromosomas y después de 200 generaciones) se muestran en la Tabla 2. Note cómo la solución obtenida por el algoritmo genético no hace uso de la compuerta NOT, y usa sólo un AND, en vez de los 2 utilizados por el diseñador humano. El ahorro logrado en este caso es del 33%, aunque el ejemplo es demasiado sencillo como para considerarlo una aplicación seria; sin embargo, resultará interesante en cada ejemplo subsecuente analizar la solución producida por el algoritmo genético y compararla con la producida por un humano, para intentar de encontrar claves en torno a la forma en que el primero reduce la función Booleana.

5.2 Ejemplo 2

Nuestro segundo ejemplo consta de 4 entradas y una salida, tal y como se indica en la Tabla 3. En este caso, la matriz utilizada fue de 5×5, la longitud cromosómica fue de 75 ($r = 5, q = 5, t = 5 \times 5 = 25, l = 3 \times t = 75$).

La comparación de los resultados producidos por el algoritmo genético (con una población de 1000 cromosomas y después de 200 generaciones) y un diseñador humano se muestran en la Tabla 4. Note cómo la solución humana no hace uso de la compuerta XOR, mientras que el algoritmo genético utiliza 3 de ellas en su solución, produciendo un ahorro del 29%. Un aspecto interesante de este problema es que está documentado en la literatura de optimización de circuitos. Sasao [Sasao, 1993] lo utiliza para ejemplificar su técnica de simplificación de circuitos mediante compuertas ANDs

Algoritmo Genético	Diseñador Humano
$F = (X + Z)(XZ \oplus Y)$	$F = \overline{X}YZ + X(Y \oplus Z)$
4 compuertas	6 compuertas
2 ANDs, 1 OR, 1 XOR	3 ANDs, 1 OR, 1 XOR, 1 NOT

Tabla 2: Comparación de resultados entre el Algoritmo Genético y un Diseñador Humano para el circuito del primer 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

Tabla 3: Tabla de verdad del circuito del segundo ejemplo.

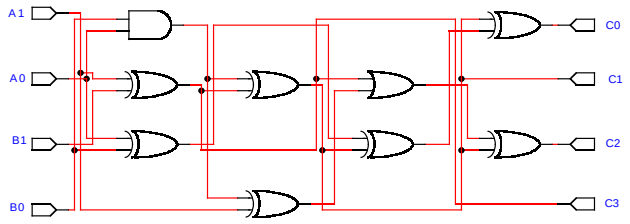

Figura 2: Un sumador de 2 bits generado por el algoritmo genético usando representación binaria.

y XORs, y presenta como solución mínima la función $F = \overline{X} \oplus \overline{Y}W \oplus XY\overline{Z} \oplus \overline{XY}W$, que usa 12 compuertas (5 ANDs, 3 XORs, 4 NOTs), y que es todavía 15% menos eficiente que la solución generada por el algoritmo genético.

5.3 Ejemplo 3

Este ejemplo es un sumador de 2 bits, cuya función Booleana consta de 4 entradas y 3 salidas, tal y como se muestra en la Tabla 5. En este caso, la matriz utilizada fue de 5×5, la longitud cromosómica fue de 75 ($r = 5, q = 5, t = 5 \times 5 = 25, l = 3 \times t = 75$).

Los resultados producidos por el algoritmo genético (usando una población de 2000 cromosomas y después de

A_1	A_0	B_1	B_0	X_2	X_1	X_0
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 5: Tabla de verdad del sumador de 2 bits del tercer ejemplo.

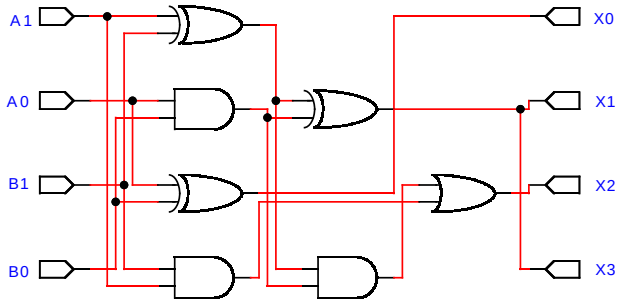

Figura 3: Un sumador de 2 bits generado por el algoritmo genético usando un alfabeto de cardinalidad 4.

Algoritmo Genético	Diseñador Humano
$F = \overline{WY\overline{X}} \oplus ((W+Y) \oplus Z \oplus (X+Y+Z))$	$F = \overline{W}(\overline{X+Y} + Y(X+Z)) + (\overline{X+Z} + X\overline{Y}Z)W$
10 compuertas	14 compuertas
2 ANDs, 3 ORs, 3 XORs, 2 NOTs	5 ANDs, 5 ORs, 4 NOTs

Tabla 4: Comparación de resultados entre el Algoritmo Genético y un Diseñador Humano para el circuito del segundo ejemplo.

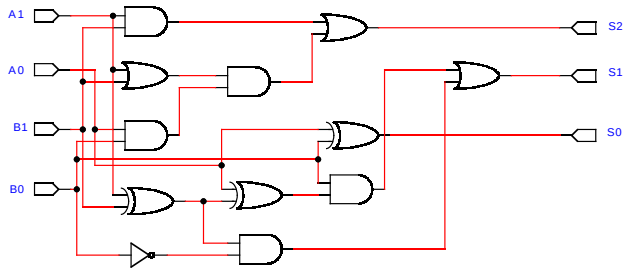

Figura 4: Un sumador de 2 bits generado por un diseñador humano.

200 generaciones) y su comparación con los producidos por un diseñador humano (ver Figura 4) se resumen en la Tabla 6.

Note cómo la solución generada por el algoritmo genético (ver Figura 3) produce ahorros del 42% con respecto al circuito del diseñador humano, haciendo uso de un hábil reacomodo de las compuertas XOR, que es altamente no intuitivo para un humano. Con respecto a la solución de Louis, debe hacerse notar que aunque es equivalente a la que produce nuestro algoritmo genético (usa 4 XORs y 3 ANDs), Louis aplica varias restricciones al problema a fin de hacer que su técnica pueda converger en un tiempo razonable. Por ejemplo, Louis considera fija una de las 2 entradas a cada compuerta, y sólo codifica la segunda entrada, a fin de reducir la longitud total de los cromosomas que, debido al uso de una representación binaria, puede hacerse indeseablemente larga en problemas más grandes. Cabe aclarar que, aunque la mayoría de los lenguajes de programación permiten el acceso directo o indirecto a bits (por ejemplo, Pascal y C), un buen número de implementaciones de algoritmos genéticos recurren, por facilidad, al uso de arreglos o listas enlazadas de enteros para representar con el mismo esquema alfabetos binarios y alfabetos de cualquier otra cardinalidad; este es el caso de MOSES [Coello, 1996], y de ahí el argumento de que una representación binaria consume más memoria. Obviamente, al ser reducido cada entero (o caracter) de una representación no binaria a bits, cualquier representación fuera de la binaria será más ineficiente. Nuestro argumento se fundamenta más que nada en el hecho de que las cadenas cromosómicas muy largas (asumiendo que una representación binaria se manejara como una lista enlazada de enteros) resultan mucho más difíciles de manejar que las cadenas cor-

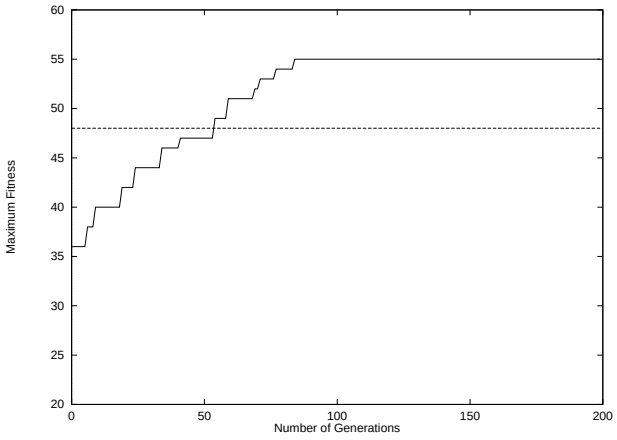

Figura 5: Gráfica de convergencia del AG que generó un sumador de 2 bits usando una representación binaria. La evolución de la aptitud máxima (*maximum fitness*) se muestra a través de 200 generaciones. Un valor de aptitud por encima de la línea horizontal punteada indica que se trata de un diseño funcional.

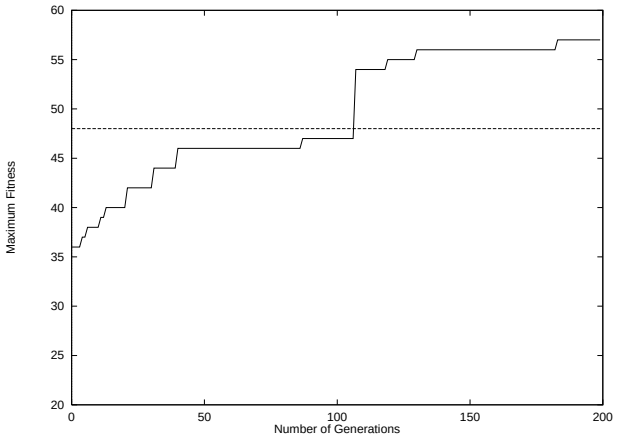

Figura 6: Gráfica de convergencia del AG que generó un sumador de 2 bits usando un alfabeto de cardinalidad 4. La evolución de la aptitud máxima (*maximum fitness*) se muestra a través de 200 generaciones. Un valor de aptitud por encima de la línea horizontal punteada indica que se trata de un diseño funcional.

Algoritmo Genético	Diseñador Humano
7 compuertas	12 compuertas
$X_0 = A_0 \oplus B$ $X_1 = (A_1 \oplus B_1) \oplus A_0 B_0$ $X_2 = A_1 B_1 + A_0 B_0 (A_1 \oplus B_1)$	$X_0 = A_0 \oplus B_0$ $X_1 = (A_1 \oplus B_1)\overline{B_0} + ((A_1 \oplus B_1) \oplus A_0) B_0$ $X_2 = A_1 B_1 + A_0 B_0 (A_1 + B_1)$
2 ANDs, 2 ORs, 3 XORs	5 ANDs, 3 ORs, 3 XORs, 1 NOT

Tabla 6: Comparación de resultados entre el Algoritmo Genético y un Diseñador Humano para el sumador de 2 bits del tercer ejemplo.

tas, a nivel de la estructura de datos, y no a nivel de la representación interna. Asumiendo una verdadera representación de bits en la implementación, esta afirmación es una falacia, porque la representación binaria resulta óptima en todos los casos.

Nuestro algoritmo genético no hace uso de la restricción impuesta por Louis, lo que incrementa el tamaño del espacio de búsqueda, haciendo el problema más difícil de resolver para el algoritmo genético. Otro aspecto interesante al comparar la solución de Louis con la nuestra es el hecho de que las funciones Booleanas para las salidas X_0 y X_1 son exactamente las mismas que produce nuestro AG, pero X_2 tiene una pequeña diferencia: Louis usa una compuerta XOR para separar los 2 términos indicados en la Tabla 6, mientras que nuestra solución usa un OR (i.e., para Louis, $X_2 = A_1 B_1 \oplus A_0 B_0 (A_1 \oplus B_1)$. Esto se debe a que con esta función en particular es indistinto utilizar cualquiera de estas 2 compuertas, pues ambas producen exactamente el mismo resultado. Cabe mencionar que en este ejemplo, también experimentamos con un algoritmo genético que usara representación binaria, y el mejor resultado producido (con una población de 3000 cromosomas y después de 200 generaciones) tuvo 9 compuertas (ver Figura 2), y aunque es todavía 25% mejor que la solución del diseñador humano, no es más optimizado que el diseño producido por nuestro AG con cardinalidad 4. Las gráficas de convergencia para este ejemplo, usando representación binaria y un alfabeto de cardinalidad 4 se muestran en las Figuras 5 y 6 respectivamente. Una aptitud de 48 es el mínimo requerido para considerar a un circuito como válido (hay 48 valores correspondientes a todas las salidas de la tabla de verdad). Observe cómo la representación binaria contribuye a que se encuentren circuitos funcionales más rápido que nuestra representación alterna, pero a la vez se queda atrapada con mayor facilidad en un mínimo local (los resultados aquí mostrados son producto de varias corridas en las que se probaron diferentes parámetros para el AG), mientras que un alfabeto de mayor cardinalidad permite encontrar soluciones más optimizadas aún después de correr el AG por más de 100 generaciones.

5.4 Ejemplo 4

El último ejemplo que incluiremos será un multiplicador de 2 bits, tal y como se representa en la Tabla 7. En este caso, la matriz utilizada fue de 5×5, la longitud cromosómica fue de 96 ($r = 8, q = 4, t = 8 \times 4 = 32, l = 3 \times t = 96$).

$\mathbf{A_1}$	$\mathbf{A_0}$	$\mathbf{B_1}$	$\mathbf{B_0}$	$\mathbf{C_3}$	$\mathbf{C_2}$	$\mathbf{C_1}$	$\mathbf{C_0}$
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 7: Tabla de verdad del multiplicador de 2 bits del cuarto ejemplo.

En este caso, las salidas fueron consideradas de manera combinada y los resultados producidos por el algoritmo genético (con una población de 2000 cromosomas y después de 200 generaciones) y el diseñador humano se resumen en la Tabla 8. Note cómo en este caso la solución generada por el algoritmo genético produce ahorros de tan sólo el 13% con respecto a la solución del diseñador humano. Sin embargo, pese al ahorro de sólo una compuerta, el diseño producido por el algoritmo genético dista mucho de ser trivial, pues se vale de un XOR para producir este ahorro. Curiosamente, la solución para la salida C_2 (ver Tabla 8) es más compleja que la generada por un diseñador humano, pero debido a que utiliza nuevamente el término $A_0 B_0 A_1 B_1$ empleado también en la salida C_3, el circuito total resultante es ligeramente más simple. Este tipo de reducción es no sólo difícil de lograr usando las identidades algebraicas convencionales, sino que además resulta poco intuitiva para un humano. También en este ejemplo utilizamos

un algoritmo genético con representación binaria, y el mejor resultado producido después de 200 generaciones utiliza 17 compuertas, siendo mucho mas ineficiente que el diseño producido por un humano y sirve como indicativo del porqué realizamos nuestros experimentos con una representación alterna.

6 Conclusiones

Hemos introducido una técnica para diseño automatizado de circuitos lógicos combinatorios basada en el algoritmo genético, y hemos comparado los resultados que produce con los circuitos generados por un diseñador humano. Los resultados de los ejemplos presentados en este artículo así como los demás que hemos resuelto en nuestros experimentos son muy alentadores, porque no sólo utilizan menos compuertas que las soluciones de un humano, sino que también son muy fáciles de producir (sólo se requiere de una tabla de verdad que represente el mapeo de entradas y salidas de la función Booleana y unos cuantos parámetros adicionales para el algoritmo genético) con nuestro sistema. Hemos mostrado también cómo el uso de un alfabeto de cardinalidad mayor que 2 permite una mayor flexibilidad a la representación cromosómica, e impide que la longitud de una cadena se vuelva innecesariamente larga, ayudando de paso a encontrar mejores soluciones que las halladas por una representación binaria.

Trabajo Futuro

La primera extensión lógica de este trabajo es el abordar el diseño de circuitos de más alta complejidad que los logrados hasta ahora, y asimismo poder resolverlos considerándolos como una sola unidad, en vez de tener que separar cada una de sus salidas. En ese sentido, estamos trabajando en una representación alterna similar a la propuesta por Sadasivan [Sadasivan, 1993], y que combina el poder de las expresiones S usadas por la programación genética [Koza, 1992] con la flexibilidad de la representación lineal usada por el algoritmo genético y que parece favorecer la optimización. Esta representación parece ser una línea de investigación muy prometedora, y queremos utilizarla para implementar un sistema completo de diseño automático de circuitos lógicos que podría servir no sólo para fines académicos (como hasta ahora), sino también para fines profesionales y/o comerciales. Finalmente, también nos interesa extender nuestra implementación a fin de tomar en cuenta otros factores tales como el número de niveles del circuito, el número total de compuertas, el número de paquetes de circuitos integrados, los costos de cada parte, etc. En base a los alentadores resultados que hemos obtenido hasta la fecha, estamos convencidos de que el algoritmo genético puede volverse una herramienta práctica muy útil para asistir a los diseñadores de circuitos lógicos en su compleja y tediosa labor.

Referencias

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

[Brayton *et al.*, 1987] Brayton, R. K.; Rudell, R.; Sangiovanni-Vincentelli, A.; and Wang, A. R. 1987. MIS: A multiple-level logic optimization system. *IEEE Transactions on Computer-Aided Design* CAD-6 (6):1062–1081.

[Brzozowski and Yoeli, 1976] Brzozowski, Janusz A. and Yoeli, Michael 1976. *Digital Networks*. Prentice Hall.

[Coello, 1996] Coello, Carlos Artemio Coello 1996. *An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design*. Ph.D. Dissertation, Department of Computer Science, Tulane University, New Orleans, LA.

[Goldberg, 1989] Goldberg, David E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, Mass. : Addison-Wesley Publishing Co.

[Holland, 1992] Holland, John H. 1992. *Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, Massachusetts.

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

[Koza *et al.*, 1996a] Koza, John R.; Andre, David; Forrest H. Bennett, III; and Keane, Martin A. 1996a. Use of automatically defined functions and architecture-altering operations in automated circuit synthesis with genetic programming. In Koza, John R.; Goldberg, David E.; Fogel, David B.; and Riolo, Rick L., editors 1996a, *Proceedings of the First Annual Conference on Genetic Programming*, Cambridge, Masachussetts. Stanford University, The MIT Press. 132–140.

[Koza *et al.*, 1996b] Koza, John R.; Forrest H. Bennett, III; Andre, David; and Keane, Martin A. 1996b. Automated WYWIWYG design of both the topology and component values of electrical circuits using genetic programming. In Koza, John R.; Goldberg, David E.; Fogel, David B.; and Riolo, Rick L., editors 1996b, *Proceedings of the First Annual Conference on Genetic Programming*, Cambridge, Masachussetts. Stanford University, The MIT Press. 123–131.

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

Algoritmo Genético	**Diseñador Humano**
7 compuertas	8 compuertas
$C_0 = A_0B_0$ $C_1 = A_0B_1 \oplus A_1B_0$ $C_2 = A_1B_1 \oplus (A_0B_0A_1B_1)$ $C_3 = A_0B_0A_1B_1$	$C_0 = A_0B_0$ $C_1 = A_0B_1 \oplus A_1B_0$ $C_2 = A_1B_1(\overline{A_0B_0})$ $C_3 = A_1A_0B_1B_0$
5 ANDs, 2 XORs	6 ANDs, 1 XORs, 1 NOT

Tabla 8: Comparación de resultados entre el Algoritmo Genético y un Diseñador Humano para el multiplicador de 2 bits del cuarto ejemplo.

[Louis, 1993] Louis, Sushil J. 1993. *Genetic Algorithms as a Computational Tool for Design*. Ph.D. Dissertation, Department of Computer Science, Indiana University.

[McCluskey, 1956] McCluskey, E. J. 1956. Minimization of boolean functions. *Bell Systems Technical Journal* 35 (5):1417–1444.

[Michalewicz, 1992] Michalewicz, Zbigniew 1992. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag.

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

[Roth, 1992] Roth, Charles H. Jr. 1992. *Fundamentals of Logic Design (4th Edition)*. West Publishing Company.

[Sadasivan, 1993] Sadasivan, Thyagarajan 1993. Genetic algorithm approach to information retrieval. Center for Intelligent and Knowledge-Based Systems CS/CIAKS-93-012/TU, Tulane University, Department of Computer Science, New Orleans, Louisiana (USA).

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

[Shannon, 1938] Shannon, C. E. 1938. A symbolic analysis of relay and switching circuits. *Transactions of the AIEE* 57:713–723.

[Thomson and Miller, 1996] Thomson, P. and Miller, J. F. 1996. Symbolic method for simplifying AND-EXOR representations of boolean functions using a binary-decision technique and a genetic algorithm. *IEE Proceedings. Computers and Digital Techniques* 143(2):97–101.

[Veitch, 1952] Veitch, E. W. 1952. A chart method for simplifying boolean functions. *Proceedings of the ACM* 127–133.