Diseño de Circuitos Lógicos Combinatorios Utilizando Programación Genética Postfija con Adaptación en Línea.

por

Edgar Galván López

Para obtener el grado de Maestro en Inteligencia Artificial.

Maestría en Inteligencia Artificial
Universidad Veracruzana - LANIA

Dr. Carlos A. Coello Coello
CINVESTAV-IPN
Asesor

Dra. Katya Rodríguez Vázquez
IIMAS-UNAM
Co-Asesora

Dr. Manuel Martínez Morales
MIA-UV
Revisor

Mayo del 2002

Dedicatorias

Dedico esta tesis a Ma. Teresa, mi madre, por siempre apoyarme y aconsejarme. A Isidro, mi padre, por alentarme a seguir con mis estudios y tranquilizarme cuando más lo necesitaba.

Los amo.

Agradecimientos

- Le doy gracias a Dios por poner en mi vida a tres seres extraordinarios: mi madre, mi padre y mi hermana. Gracias por estar conmigo.
- Agradezco al Dr. Carlos A. Coello Coello por brindarme su ayuda y amistad para la realización de esta tesis.
- Asimismo quiero darle las gracias a la Dra. Katya Rodríguez Vázquez por orientarme en este trabajo de tesis.
- Esta tesis se derivo del proyecto de CONACyT titulado "Estudio y Desarrollo de Técnicas Avanzadas de Manejo de Restricciones para Algoritmos Evolutivos en el Contexto de Optimización Numérica" (Ref. 32999-A) cuyo responsable es el Dr. Carlos A. Coello Coello. Se agradece el apoyo proporcionado por CONACyT a través de una beca de maestría y del proyecto antes citado para la realización de este trabajo.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Motivación.

El diseño de circuitos lógicos combinatorios generalmente es considerado como una actividad que requiere de una cierta creatividad humana y también de un cierto conocimiento. La electrónica ha utilizado diversas técnicas para el diseño de circuitos lógicos combinatorios, como por ejemplo el álgebra booleana, mapas de Karnaugh [13] o el método de Quine-McCluskey [17].

La Computación Evolutiva, cuya inspiración se basa en la selección natural, realiza una simulación de los procesos evolutivos en las computadoras. El problema de diseñar circuitos lógicos combinatorios, ya es atacado por las diversas técnicas evolutivas. A ésto se le denomina *Hardware Evolutivo.* El Hardware Evolutivo se puede clasificar en dos grupos:

- *Evolución Extrínseca.* Esta evolución se lleva a cabo a través de simulaciones, y es el que utilizamos para este trabajo de tesis.
- *Evolución Intrínseca.* Esta evolución se lleva a cabo a través de hardware reconfigurable.

Para poder resolver el diseño de circuitos lógicos combinatorios, es necesario utilizar una representación de individuos que puedan llegar a resolver un circuito determinado. En este caso los individuos son representados por expresiones postfijas.

1.2 Antecedentes.

El diseño de circuitos lógicos combinatorios ha evolucionado desde sus primeras apariciones [26], hasta el punto de llegar a utilizar estrategias evolutivas para lograr dicho diseño. El diseño gráfico estándar, como por ejemplo los Mapas de Karnaugh, es ampliamente utilizado. Sin embargo, también se utilizan métodos que son fáciles de implementar en una computadora, como por ejemplo el Método de Quine-McCluskey, o software de dominio público como Espresso.

No existe mucha información acerca del diseño de circuitos lógicos combinatorios utilizando algoritmos genéticos, Louis fue uno de los primeros investigadores en abordar este tema. Louis propuso combinar Sistemas Basados en el Conocimiento con Algoritmos Genéticos, haciendo uso de un operador genético denominado *"Masked Crossover"* que adapta la decodificación, siendo capaz de explotar la información no utilizada por el operador de cruza clásico. Sus resultados son muy alentadores para ciertos ejemplos, pero abordan de manera muy limitada la complejidad asociada al diseño de circuitos combinatorios. Por otra parte, su aportación constituye un gran paso para incrementar el poder de los algoritmos genéticos como una herramienta de diseño. Desafortunadamente la incorporación del conocimiento en los algoritmos genéticos decrementa su utilidad como una herramienta general de búsqueda.

Koza [14] ha utilizado la programación genética para diseñar circuitos lógicos. El ha diseñado, por ejemplo, el sumador de dos bits, usando un pequeño conjunto de compuertas AND, OR, NOT, pero su trabajo enfatiza la generación de circuitos funcionales más que en una optimización de los mismos. La programación genética ha sido considerada una herramienta poderosa en muchas tareas, debido a que la representación que utiliza es más poderosa para el diseño estructural en general.

1.3 Objetivo.

Los objetivos principales que se plantean en esta tesis son los siguientes:

- Usar la programación genética con cadenas postfijas que representen los individuos de una población, de manera que éstos a su vez representen los circuitos a evaluar con una función de aptitud definida.

- Resolver circuitos de una y múltiples salidas para ver el comportamiento de la implementación propuesta en esta tesis, así como evaluar la calidad de los circuitos producidos.

1.4 Estructura de la tesis.

La tesis se compone de 7 capítulos, los cuales se describen a continuación:

- **Capítulo 1.** En este capítulo se presenta a grosso modo una introducción al diseño de circuitos lógicos combinatorios así como parte del trabajo previo en este campo. También se presenta una breve descripción de cada uno de los capítulos restantes que conforman la tesis.

- **Capítulo 2.** En este capítulo se presentan los teoremas y propiedades principales que involucran el diseño de circuitos lógicos combinatorios así como una explicación de las diversas técnicas que se han desarrollado para llevar a cabo este tipo de diseños.

- **Capítulo 3.** En este capítulo se da una explicación de lo que es la programación genética así como sus fundamentos teóricos. También se explican a detalle los operadores principales que se pueden aplicar dentro de este paradigma de la computación evolutiva.

- **Capítulo 4.** En este capítulo se describe la implementación llevada a cabo con la técnica de programación genética postfija. También se describe la función de aptitud que se utiliza para evaluar cada individuo de la población así como los operadores genéticos utilizados.

- **Capítulo 5.** En este capítulo se muestran los resultados de circuitos de una sola salida. En este caso se seleccionaron los circuitos Even-n-Parity. Se resuelven circuitos de distintos grados de complejidad para ver el comportamiento de la implementación propuesta en esta tesis.

- **Capítulo 6.** En este capítulo se muestran los resultados de circuitos de más de una salida. Se resuelven circuitos de distintos grados de complejidad para ver el comportamiento de la implementación propuesta en esta tesis.

- **Capítulo 7.** En este último capítulo se presentan las conclusiones finales del trabajo realizado en esta tesis, así como algunas líneas posibles de trabajo futuro.

Capítulo 2

Circuitos Lógicos.

El diseño digital [3, 23] se conoce también con otros nombres, como *diseño lógico, circuitos conmutadores, lógica digital y sistemas digitales.* Los circuitos se emplean en diseño de sistemas, como por ejemplo computadoras digitales, calculadoras electrónicas, dispositivos digitales de control, equipo de comunicación digital y muchas otras aplicaciones que requieren hardware digital electrónico.

2.1 Álgebra Booleana.

En 1854, George Boole [22] introdujo un tratamiento sistemático de la lógica y desarrolló para este propósito un sistema algebraico que ahora se conoce como álgebra booleana. En 1983, Claude Shannon introdujo el álgebra booleana de dos valores denominada *álgebra de interruptores*, y demostró que las propiedades de los circuitos eléctricos estables con interruptores puede representarse matemáticamente usando esta álgebra.

El álgebra booleana, como cualquier otro sistema matemático deductivo, puede definirse con un conjunto de elementos, un conjunto de operadores y un número de axiomas no probados o postulados.

El álgebra booleana trata con variables que toman dos valores discretos y con operaciones que tienen significado lógico. Los dos valores que toman las variables pueden designarse con nombres diferentes (ésto es, *verdadero y falso, si y no, 0 y 1, etc.*). El álgebra booleana se usa para describir, en forma matemática, la manipulación y el proceso de la información binaria. El álgebra booleana es en particular adecuada para el análisis y diseño de

sistemas digitales.

El álgebra booleana consta de variables binarias y operaciones lógicas. Las variables se denotan con letras del alfabeto como A, B, C, x, y, z, etc., y cada variable tiene dos y sólo dos valores posibles distintos: 1 y 0. Hay dos operaciones lógicas binarias: `AND y OR` y una operación unaria: `NOT`. Veamos cada una de ellas en mayor detalle:

1. AND (Y). Esta operación se representa mediante un punto o por la ausencia de operador. Por ejemplo, $x \cdot y = z$ o $xy = z$ se lee "x Y y es igual a z". La operación lógica AND se interpreta con el significado $z = 1$ si y sólo si $x = 1$ y $y = 1$; en cualquier otro caso $z = 0$. (Recuérdese que x,y y z son variables binarias y pueden ser iguales a 1 ó 0 únicamente).

2. OR (O). Esta operación se representa mediante un signo de suma. Por ejemplo, $x + y = z$, se lee "x O y es igual a z", lo cual significa que $z = 1$ si $x = 1$ o si $y = 1$ o si tanto $x = 1$ como $y = 1$. Si tanto $x = 0$ como $y = 0$, entonces $z = 0$.

3. NOT (NO). Esta operación está representada por un apóstrofe. Por ejemplo, $x' = z$ se lee "x no es igual a z",y significa que z es la negación de x. En otras palabras, si $x = 1$, entonces $z = 0$; pero si $x = 0$, entonces $z = 1$.

Para cada combinación de los valores x y y, hay un valor de z especificado por la definición de la operación lógica. Estas definiciones pueden listarse en forma compacta usando las *tablas de verdad*. Una tabla de verdad es una tabla de todas las combinaciones posibles de las variables, que muestra la relación entre los valores que pueden tomar las variables y el resultado de la operación. Por ejemplo, las tablas de verdad para las operaciones `AND y OR` con las variables x y y se obtienen haciendo la lista de todos los valores posibles que pueden tener las variables cuando se combinan en pares. El resultado de la operación para cada combinación se lista entonces en una columna separada. Las tablas de verdad para `AND, OR y NOT` se muestran en la Tabla 2.1.

2.1.1 Teoremas y propiedades del álgebra booleana.

Los postulados de un sistema matemático forman los supuesto básicos mediante los cuales es posible deducir las reglas, teoremas y propiedades

AND			*OR*			*NOT*	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Tabla 2.1: Tabla de verdad de operaciones lógicas.

del sistema. Los postulados más comunes que se utilizan para formular diversas estructuras algebraicas son:

1. **Propiedad de Cierre.**
   Un conjunto S se dice que es cerrado para un operador binario si, para cada elemento de S el operador binario especifica una regla para obtener un elemento único de S.

2. **Ley Asociativa.**
   El operador binario $*$ en un conjunto S se dice que es asociativo siempre que $(x * y) * z = x * (y * z)$ para todas x,y,$z \epsilon S$.

3. **Ley Conmutativa.**
   Un operador binario $*$ en un conjunto S se dice que es conmutativo siempre que: $x * y = y * x$ para todas x,$y \epsilon S$.

4. **Elemento Identidad.**
   Un conjunto S se dice que tiene un elemento identidad respecto a una operación binaria $*$ en S si existe un elemento $e \epsilon S$ con la propiedad: $e * x = x * e = x$ para cada $x \epsilon S$.

5. **Inversa.**
   Un conjunto S que tiene un elemento identidad e con respecto a un operador binario $*$ se deduce que tiene una inversa siempre que, para cada $x \epsilon S$, exista un elemento $y \epsilon S$ tal que: $x * y = e$.

6. **Ley Distributiva.**
   Si el operador $*$ y el operador $\cdot$, son dos operadores binarios en un conjunto S, $*$ se dice que es distributivo sobre x, siempre que: $x * (y \cdot z) = (x * y) \cdot (x * z)$.

Los operadores y postulados tienen los siguientes significados:

- El operador binario + define la adición.

- La identidad aditiva es cero.
- La inversa aditiva define la sustracción.
- El operador binario $\cdot$ define la multiplicación.
- El inverso multiplicativo de $a = 1/a$ define la división, ésto es, $a * 1/a = 1$
- La única ley distributiva aplicable es la de $\cdot$ sobre el operador $+$: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$

Al comparar el álgebra booleana con la aritmética y el álgebra ordinaria (el campo de los números reales), se observan las siguientes diferencias:

1. Los postulados de Huntington[1] no incluyen la ley asociativa. No obstante, esta ley es válida para el álgebra booleana y puede derivarse (para ambos operadores) mediante otros postulados.
2. La ley distributiva del operador $+$ sobre el operador $\cdot$, esto es: $x + (y \cdot z) = (x + y) \cdot (x + z)$, la cual es válida para el álgebra booleana pero no para el álgebra ordinaria.
3. El álgebra booleana no tiene inversa aditiva o multiplicativa; por lo tanto, no hay operaciones de sustracción o división.
4. El postulado 5 define un operador llamado complemento que no se encuentra en el álgebra ordinaria.
5. El álgebra ordinaria trata con números reales, los cuales constituyen un conjunto infinito de elementos. El álgebra booleana que trata con el conjuto B, se define en términos de dos elementos, 0 y 1.

Los postulados y teoremas del álgebra booleana se listan en la Tabla 2.2.

2.1.2 Compuertas Lógicas.

Ya que las funciones booleanas se expresan en términos de operaciones AND, OR y NOT, es fácil implementar una función booleana con estos tipos de compuertas. La posibilidad de construir compuertas para otras operaciones lógicas es de interés práctico. Los factores que hay que tomar en cuenta cuando se considera la construcción de otros tipos de compuertas lógicas son:

[1] En 1904 E.V. Hutington formuló los cuatro postulados para el álgebra de Boole (conmutativa, distributiva, idempotente, absorción).

Postulados y Teoremas	*Representaciones*	
Postulado 2	(a) $x + 0 = x$	(b) $x \cdot 1 = x$
Postulado 5	(a) $x + x' = 1$	(b) $x \cdot x' = 0$
Teorema 1	(a) $x + x = x$	(b) $x \cdot x = x$
Teorema 2	(a) $x + 1 = 1$	(b) $x \cdot 0 = 0$
Teorema 3, involución	$(x')' = x$	
Postulado 3, conmutativo	(a) $x + y = y + x$	(b) $xy = yx$
Teorema 4, asociativo	(a) $x + (y + z) = (x + y) + z$	(b) $x(yz) = (xy)z$
Postulado 4, distributivo	(a) $x(y + z) = xy + xz$	(b) $x + yz = (x + y)(x + z)$
Teorema 5, de De Morgan	(a) $(x + y)' = x'y'$	(b) $(xy)' = x' + y'$
Teorema 6, absorción	(a) $x + xy = x$	(b) $x(x + y) = x$

Tabla 2.2: Tabla de Postulados y Teoremas del álgebra Booleana.

1. La factibilidad y economía de producir la compuerta con componentes físicos.

2. La posibilidad de extender la compuerta a más de dos entradas.

3. Las propiedades básicas del operador binario como conmutabilidad y asociatividad, y

4. La habilidad de la compuerta para implantar compuertas booleanas solas o junto con otras compuertas.

De las 16 funciones que se definen en la Tabla 2.3, dos son iguales a una constante y otras cuatro se repiten dos veces. Sólo quedan diez funciones qué considerar como candidatos para compuertas lógicas. Dos de ellas, inhibición e implicación no son conmutativas o asociativas y, por tanto, no es práctico usarlas como compuertas lógicas estándar. Las otras ocho: complemento, transferencia, AND, OR, NAND, NOR, OR-exclusivo, y equivalencia, se utilizan como compuertas estándar en el diseño digital.

Los símbolos gráficos y las tablas de verdad de las cuatro compuertas básicas se muestran en la Figura 2.1. Cada compuerta tiene una o dos variables binarias de entrada designadas por x y y y una variable binaria de salida designada por F. El circuito inversor invierte el sentido lógico de una variable binaria. Produce la función NOR o complemento. El pequeño círculo en la salida del símbolo gráfico de un inversor designa el complemento lógico. El símbolo del triángulo por sí mismo denota un circuito buffer. Un buffer produce la función de transferencia pero no produce alguna operación lógica particular, ya que el valor binario de la salida es igual al valor binario de la entrada. El circuito se usa simplemente para amplificación de potencia de la señal y es equivalente a dos inversores conectados en cascada.

La función NAND es el complemento de la función AND. La función NOR es el complemento de la función OR y usa un OR seguido de un círculo pequeño. Las compuertas NAND y NOR se utilizan en forma extensa como

Nombre	Símbolo Gráfico	Función Algebraica	Tabla de Verdad
AND		F = xy	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		F = x+y	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
INVERSOR		F = x'	x | F 0 | 1 1 | 0
OR EXCLUSIVO		F = xy'+x'y	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

Figura 2.1: Compuertas Lógicas Digitales.

F. Booleanas	*Símbolo*	*Nombre*	*Comentarios*
$F_0 = 0$	Nulo	Nulo	Constante binario 0
$F_1 = xy$	$x \cdot y$	AND	x y y
$F_2 = xy'$	x/y	Inhibición	x pero no y
$F_3 = x$		Transferencia	x
$F_4 = x'y$	y/x	Inhibición	y pero no x
$F_5 = y$		Transferencia	y
$F_6 = xy' + x'y$	$x \oplus y$	OR-exclusivo	x o y pero no ambas
$F_7 = x + y$	$x + y$	OR	x o y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	NOT-OR
$F_9 = xy + x'y'$	$x \odot y$	Equivalencia	x igual a y
$F_10 = y'$	y'	Complemento	No y
$F_11 = x + y'$	$x \subset y$	Implicación	Si y,entonces x
$F_12 = x'$	x'	Complemento	No x
$F_13 = x' + y$	$x \supset y$	Implicación	Si x, entonces y
$F_14 = (xy)'$	$x \uparrow y$	NAND	NOT-AND
$F_15 = 1$		Identidad	Constante binaria 1

Tabla 2.3: Expresiones Booleanas para las 16 funciones de dos variables.

compuertas lógicas estándar y de hecho se emplean más que las compuertas AND y OR. Las funciones booleanas pueden implementarse con sencillez con dichas compuertas.

La compuerta OR-exclusivo tiene un símbolo gráfico similar al de la compuerta OR excepto por la línea adicional curva en el lado de la entrada. La equivalencia, o compuerta NOR-exclusivo es el complemento de la OR-exclusivo.

2.1.3 Funciones Booleanas.

Una función booleana es una expresión formada de variables binarias (las cuales toman valores de 0 o 1), un conjunto de operadores (los cuales pueden ser los operadores binarios OR y AND y el operador unario NOT), paréntesis y el signo de igual. De tal forma, para un valor dado de variables la función puede tomar el valor de 0 o 1. Una función booleana se puede representar:

- En forma algebraica: por ejemplo $F = xy$, donde F toma el valor de 1 si $x = 1$ y $y = 1$; en cualquier otro caso $F = 0$.

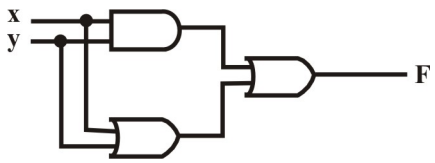

Figura 2.2: Circuito F = xy + (x+y).

x	y	$F = xy$
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 2.4: Tabla de verdad de la función F=xy

- En forma de tabla de verdad: en este caso se necesita una lista de 2^n combinaciones de 1 y 0 para n variables binarias y una columna que muestre las combinaciones para las cuales la función F toma un valor de 0 o 1. Para el ejemplo $F = xy$, se tienen 2 variables, por lo que la tabla de verdad está formada por 2^2 posibles combinaciones. Esto se muestra en la Tabla 2.4.

- En forma de diagrama lógico: éste está compuesto de compuertas AND, OR y NOT (símbolos gráficos que se muestran en la Figura 2.2). Ejemplo: $F = xy + (x + y)$, cuya representación se visualiza en la Figura 2.1.

2.2 Diseño de Circuitos Lógicos Combinatorios.

Un circuito combinatorio [3] consta de variables de entrada, compuertas lógicas y variables de salida. Las compuertas lógicas aceptan las señales de las entradas y generan señales a las salidas. Este proceso transforma la información binaria de los datos de entrada en los datos requeridos de salida. En forma obvia, tanto los datos de entrada y salida se representan por señales binarias. Esto es, existen dos valores posibles, uno representa el 1 y el otro el 0. En la Figura 2.3, se muestra un diagrama de bloques de un circuito. Las n variables binarias de entrada provienen de una fuente externa; las m variables de salida van a un destino externo.

Figura 2.3: Diagrama de bloques de un circuito combinatorio.

Para las n variables de entrada, hay 2^n combinaciones posibles de los valores binarios de entrada. Para cada combinación posible de entrada, hay una y sólo una combinación posible de salida. Un circuito combinatorio puede describirse por m funciones booleanas, una para cada variable de salida. Cada función de salida se expresa en términos de las n variables de entrada. El diseño de circuitos combinatorios surge del planteamiento verbal del problema y termina en un diagrama de circuito lógico, o un conjunto de funciones booleanas del cual puede obtenerse con facilidad el diagrama lógico. El procedimiento sigue estos pasos:

1. Se enuncia el problema.
2. Se determina el número de variables de entrada disponibles y de las variables de salida requeridas.
3. Se asignan símbolos literales a las variables de entrada y salida.
4. Se deriva la tabla de verdad que define las relaciones requeridas entre las entradas y las salidas.
5. Se obtiene la función booleana simplificada para cada salida.
6. Se dibuja el diagrama lógico.

Una tabla de verdad para un circuito combinatorio consta de columnas de entrada y columnas de salida. Los 1's y 0's en las columnas de entrada se obtienen de las 2^n combinaciones binarias disponibles para las n variables de entrada. Los valores binarios para las salidas se determinan del examen del problema enunciado. Una salida puede ser igual ya sea a 0 ó 1 para cada combinación válida de entrada. Sin embargo, las especificaciones pueden indicar que algunas combinaciones de entrada no ocurrirán. Estas combinaciones se vuelven condiciones "no importa". A continuación se muestra un ejemplo de un circuito lógico simple.

Entradas			*Salidas*	
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabla 2.5: Tabla de verdad para el sumador completo.

Sumador Completo.

Un sumador completo [3] es un circuito combinatorio que representa la suma aritmética de tres bits de entrada. Consta de tres entradas y dos salidas. Dos de las variables de entrada, que se indican por x y y, representan los dos bits significativos, mientras que la tercera entrada, z, representa la cuenta que se lleva de la posición previa significativa más baja. Son necesarias dos salidas debido a que la suma aritmética de tres dígitos binarios varían en valor desde 0 a 3 y el 2 o 3 binarios requieren de dos dígitos. Las dos salidas se denotan por los símbolos S para suma y C para la cuenta que se lleva (acarreo). La variable binaria S da el valor del bit menos significativo de la suma. La variable binaria C da la cuenta que se lleva de salida. La Tabla 2.5 corresponde al sumador completo.

Los ocho renglones bajo las variables de entrada denotan todas las combinaciones posibles de 1 y 0 que pueden tener esas variables. Los 1's y 0's de las variables de salida se determinan de la suma aritmética de los bits de entrada. Cuando todos los bits de entrada son 0, la salida es 0. La salida S es igual a 1 sólo cuando una entrada es igual a 1, o cuando las tres entradas son iguales a 1. La salida C tiene una cuenta que se lleva de 1, si dos o tres entradas son iguales a 1.

Basándose en la Tabla 2.5, se derivan las funciones lógicas utilizando la suma de minitérminos y se simplifican las expresiones resultantes utilizando álgebra booleana.

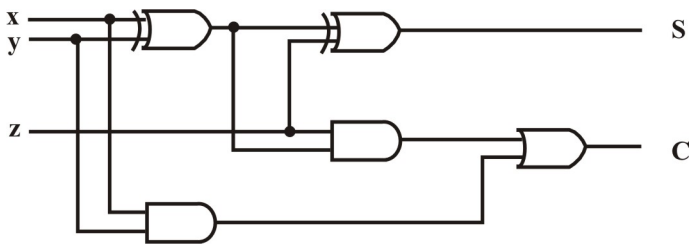

Figura 2.4: Diagrama Lógico del Sumador Completo.

$S = z \oplus (x \oplus y)$
$= z'(xy' + x'y) + z(xy' + x'y)'$
$= z'(xy' + x'y) + z(xy + x'y')$
$= xy'z' + x'yz' + xyz + x'y'z$

$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$

El diagrama lógico del sumador completo se muestra en la Figura 2.4. Nótese que la compuerta $x \oplus y$ se repite en ambas salidas, por lo que una reutilización de esta compuerta pudiera ser benéfica ya que permitiría un ahorro al momento de producirlas.

Complemento de una función.

El complemento de una función F se representa como F'. Para obtener F' a partir de la F, se puede hacer uso del teorema de DeMorgan, el cual en forma generalizada menciona que el complemento de una función se obtiene intercambiando los operadores AND por el operador OR y viceversa y complementando cada literal.

Ejemplo:
$F = x'yz' + x'y'z$ aplicando el terorema de DeMorgan $F' = (x + y' + z)(x + y + z')$

2.2.1 Forma canónica y estándar para representar las funciones booleanas.

Minitérminos y Maxitérminos.

Una variable binaria puede aparecer ya sea en su forma normal (x) o en su

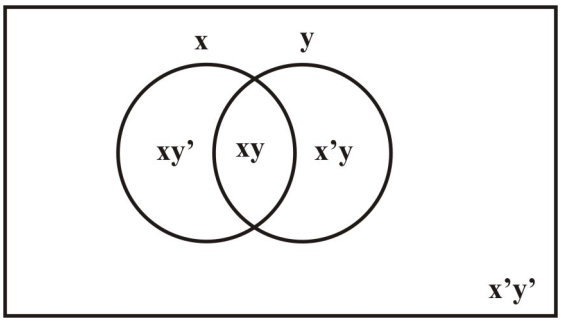

Figura 2.5: Diagrama de Venn para dos variables.

forma complementaria (x'). Ahora tomemos en cuenta las variables binarias x y y, combinadas con el operador AND. Como cada variable puede aparecer en cualquier forma, existen cuatro combinaciones posibles: x'y',x'y, xy', xy. En cada uno de los términos anteriores AND representa una de las áreas diferentes en el diagrama de Venn que se muestra en la Figura 2.5 y se denomina un minitérmino o producto estándar. En forma similar, pueden combinarse n variables para formar 2^n minitérminos.

De manera semejante, n variables forman un término OR, con cada variable vuelta prima o no, proporcionando 2^n combinaciones posibles, denominadas maxitérminos o suma estándar. Hay que hacer notar que cada maxitérmino es el complemento de su minitérmino y viceversa.

Suma de minitérminos.

Los minitérminos cuya suma define la función booleana son los que dan los 1's de la función en una tabla de verdad. Ya que la función puede tomar uno de dos valores posibles, 1 o 0, para cada minitérmino, y puesto que hay 2^n minitérminos, pueden calcularse las funciones posibles que es factible formarse con n variables para hacer 2^n. En algunas ocasiones es mejor expresar la función booleana en la forma de suma de minitérminos. Si esto no es posible, entonces puede realizarse primero por la expansión de la expresión en una suma de términos AND. Después cada término se inspecciona para ver si contiene todas las variables. Si se han perdido una o más variables, se aplica el operador AND con una expresión como $x + x'$, en donde x es una de las variables perdidas.

Producto de maxitérminos.

Para expresar la función booleana como un producto de maxitérminos, lo primero que debe hacerse es expresar dicha función a una forma de términos OR. Es posible lograr esto mediante el uso de la ley distributiva $x + yz = (x + y)(x + z)$.

2.3 Simplificación de Circuitos Lógicos.

Las funciones booleanas de salida de la tabla de verdad se simplifican por cualquier método disponible, como manipulación algebraica [23], el método de mapas de Karnaugh [23, 13], o el método de Quine-McCluskey [23, 17, 20]. Por lo común, habrá una variedad de expresiones simplificadas a elegir. No obstante, en cualquier aplicación particular ciertas restricciones, limitaciones y criterios servirán como guía en el proceso de escoger una expresión algebraica particular. Un método práctico de diseño sería tener que considerar restricciones tales como:

1. Número mínimo de compuertas,
2. Número mínimo de entradas a una compuerta,
3. Tiempo mínimo de propagación de la señal a través del circuito,
4. Número mínimo de interconexiones y
5. Limitaciones de las capacidades de impulsión de cada compuerta.

2.3.1 Simplificación Algebraica.

La expresión algebraica de una función booleana no es única. Gracias a la manipulación del álgebra booleana (haciendo uso de postulados y teoremas) es posible encontrar expresiones más simples para la misma función. Se dice que dos funciones de n variables binarias son equivalentes si tienen el mismo valor para todas las 2^n combinaciones posibles (representación de la tabla de verdad).

Una función booleana se implementa a través de compuertas lógicas (operadores binarios y unarios) y de literales (varibles binarias). Cada literal representa una entrada a una compuerta y cada término en la función booleana se implementa a través de una compuerta. Para simplificar la expresión algebraica asociada a una función booleana, se debe reducir el número de términos y de literales. Sin embargo, no siempre es posible minimizar ambos en forma simultánea. El número de literales

en una función booleana puede reducirse haciendo uso de manipulaciones algebraicas (teoremas y postulados del álgebra booleana). No obstante, no existe un conjunto de reglas que garanticen alcanzar dicho objetivo.

Ejemplo en el que se simplifica el número de literales de una función booleana:

1. $F = xy + x'z + yz$

2. $F = xy + x'z + yz(x + x')$ Postulado 5

3. $F = xy + x'z + xyz + x'yz$

4. $F = xy(1 + z) + x'z(1 + y)$ Teorema 2

5. $F = xy + x'z$

Como se puede observar en el punto número uno se tiene una expresión que se trata de reducir, el resultado de dicha reducción se puede ver en el punto número cinco. Dicho resultado es más compacto que la expresión original.

2.3.2 Método del Mapa de Karnaugh.

El mapa de Karnaugh es un diagrama compuesto por cuadros donde cada cuadro representa un minitérmino. Dado que cualquier función booleana puede expresarse como una suma de minitérminos, se deduce que una función booleana se reconoce en forma gráfica en el mapa por el área encerrada en los cuadros cuyos minitérminos se incluyen en la función. El mapa representa un diagrama visual de todas las formas posibles en que puede expresarse una función en una manera estándar. Mediante el reconocimiento de diversos patrones, el usuario puede derivar expresiones algebraicas alternas para la misma función, de las cuales puede seleccionar la más simple. Se supondrá que la expresión algebraica más simple es cualquiera en una suma de productos o producto de sumas que tiene un número mínimo de literales.

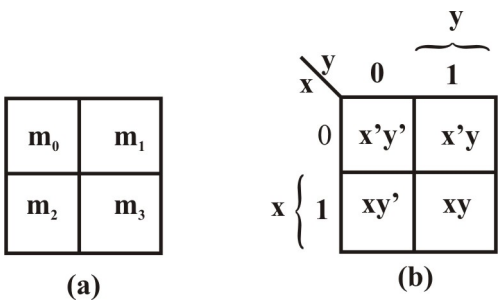

Figura 2.6: Mapa de dos variables.

En la Figura 2.6 se muestra un mapa de dos variables. Hay cuatro minitérminos para dos variables; por ende, el mapa consta de cuatro cuadros, uno para cada minitérmino. El mapa vuelve a dibujarse en (b) para mostrar las relaciones entre los cuadros y las dos variables. Los números 0's y 1's que se marcan para cada renglón y cada columna designan los valores de las variables x y y, respectivamente. Obsérvese que x aparece como prima en el renglón 0 y sin prima en el renglón 1. En forma similar, y aparece como prima en la columna 0 y sin prima en la columna 1.

Si se marcan los cuadros cuyos minitérminos pertenecen a una función dada, el mapa de dos variables se convierte en otra forma útil para representar cualquiera de las 16 funciones booleanas de dos variables. Esto se ejemplificará con la función xy que se muestra en la Figura 2.7(a). Ya que xy es igual a m_3, se coloca un 1 en el interior del cuadro que pertenece a m_3. En forma similar, la función $x+y$ se representa en el mapa de la Figura 2.7 (b) mediante tres cuadros marcados con 1. Estos cuadros se encuentran mediante los minitérminos de la función:

$$x + y = x'y + xy' + xy = m_1 + m_2 + m_3$$

Los tres cuadros pudieron haberse determinado mediante la intersección de la variable x en el segundo renglón y la variable y en la segunda columna, la cual encierra el área que pertenece a x o y.

En la Figura 2.8 se muestra un mapa de tres variables. Hay ocho minitérminos para tres variables binarias. Por lo tanto, un mapa consta de ocho cuadros. Las características de la secuencia mostrada en la figura es que sólo un bit cambia de 1 a 0 o de 0 a 1 en la secuencia listada. El mapa que se dibuja en la parte (b) se marca con números en cada renglón y cada columna para mostrar las relaciones entre los cuadros y las tres variables.

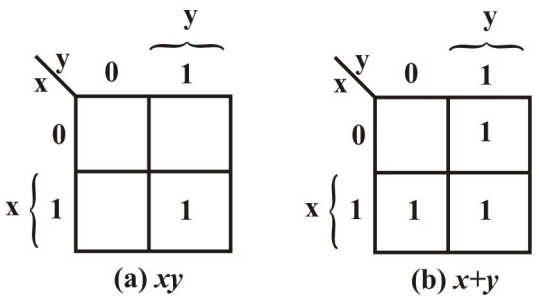

Figura 2.7: Mapa de dos variables.

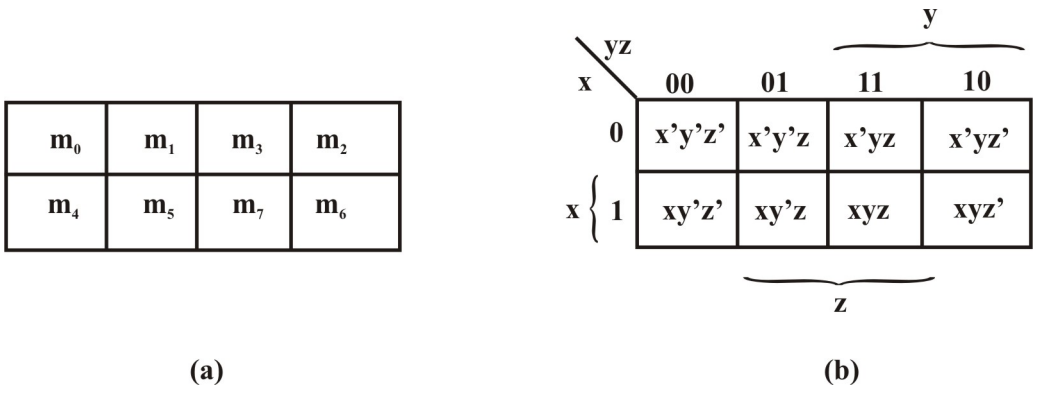

Figura 2.8: Mapa de tres variables.

Por ejemplo, el cuadro asignado a m_5 corresponde al renglón 1 y columna 01. Cuando estos dos números se concatenan, dan el número binario 101, cuyo equivalente decimal es 5. Otra forma de ver el cuadro $m_5 = xy'z$ es considerar que está en el renglón marcado con x y la columna pertecene a $y'z$ (columna 01). Obsérvese que hay cuatro cuadros donde cada variable es igual a 1 y cuatro donde cada 1 es igual a 0. La variable aparece sin prima en los cuatro cuadros donde es igual a 1 y con prima en los cuadros donde es igual a 0. Por motivos de comodidad, se escribe la variable con su símbolo de letra bajo los cuatro cuadros donde está sin prima.

Para entender la utilidad del mapa para simplificar funciones booleanas, debe reconocerse la propiedad básica que poseen los cuadros adyacentes. Cualesquiera dos cuadros adyacentes en el mapa difieren sólo en una variable que está con prima en un cuadro y sin prima en el otro. Por ejemplo, m_4 y m_7, caen en dos cuadros adyacentes. La variable y tiene prima en m_5 y no tiene prima en m_7, en tanto que las otras dos variables son las mismas en ambos cuadros. Mediante los postulados del álgebra booleana, se concluye que la suma de dos minitérminos en cuadros adyacentes puede simplificarse a un solo término AND que consta de sólo dos literales. Para aclarar lo anterior, considérese la suma de dos cuadros adyacentes como m_5 y m_7.

$$m_5 + m_7 = xy'z + xyz = xz(y' + y) = xz$$

Aquí los dos cuadros difieren por la variable y, la cual puede eliminarse cuando se forma la suma de los dos minitérminos. Por eso, cualesquiera dos minitérminos en cuadros adyacentes que se unen por el operador OR causarán una eliminación de la variable diferente.

2.3.3 Método de Tabulación o de Quine-McCluskey.

El método de mapa de simplificación es conveniente en tanto que el número de variables no exceda de cinco o seis. Conforme aumenta el número de variables, el número excesivo de cuadros evita una elección razonable de cuadros adyacentes. La desventaja obvia del mapa es que en esencia es un procedimiento de ensayo y error, que depende de la habilidad del usario humano para reconocer ciertos patrones. Para funciones de seis o más variables, es difícil tener la seguridad de que se ha hecho la mejor selección.

El método de tabulación supera esta dificultad. Es un procedimiento

específico de paso a paso que está garantizado para producir una expresión simplificada en forma estándar para una función. Puede aplicarse a problemas con muchas variables y tiene la ventaja de ser adecuado para automatizarse. Sin embargo, es bastante tedioso para el uso humano y propenso a errores debido a su proceso rutinario y monótono. El método de tabulación lo formuló por vez primera Quine y lo mejoró posteriormente McCluskey [17].

El método tabular de simplificación consta de dos partes. La primera es encontrar mediante búsqueda exhaustiva todos los términos que son candidatos para su inclusión en la función simplificada. Estos términos se denominan implicantes primos. La segunda operación es escoger entre los implicantes primos los que dan una expresión con el menor número de literales.

Determinación de los implicantes primos.

El punto de inicio del método de tabulación es la lista de los minitérminos que especifican la función. La primera operación tabular es encontrar los implicantes primos usando un proceso de comparación. Este proceso compara cada minitérmino con cada uno de los otros minitérminos. Si dos minitérminos difieren sólo en una variable, esta variable se elimina y se encuentra un término con una literal menos. Este proceso se repite para cada minitérmino hasta que se completa la búsqueda exhaustiva. El ciclo del proceso de comparación se repite para los nuevos términos que acaban de encontrarse. Los ciclos tercero y posteriores se continúan hasta que un paso único a través de un ciclo no rinde más eliminación de literales. Los términos restantes y todos los términos que no comparan durante el proceso comprenden los implicantes primos. Este método de tabulación se ilustra en el siguiente ejemplo:

Simplifique la siguiente función booleana utilizando el método de tabulación:
$F = \Sigma(0, 1, 2, 8, 10, 11, 14, 15)$
$F = w'x'y'z' + w'x'y'z + w'x'yz' + wx'y'z' + wx'y'z + wx'yz + wxyz' + wxyz$

Paso 1: Se hace la representación binaria de grupo de los minitérminos de acuerdo con el número de 1's contenidos, como se muestra en la Figura 2.9, columna (a). Esto se hace agrupando los minitérminos en cinco secciones separadas por líneas horizontales. La primera sección contiene el número sin 1 en ellas. La segunda sección contiene los números que tienen sólo un

(a)

w	x	y	z	
0	0	0	0	✓
1	0	0	0	1 ✓
2	0	0	1	0 ✓
8	1	0	0	0 ✓
10	1	0	1	0 ✓
11	1	0	1	1 ✓
14	1	1	1	0 ✓
15	1	1	1	1 ✓

(b)

	w	x	y	z	
0,1	0	0	0	-	
0,2	0	0	-	0	✓
0,8	-	0	0	0	✓
2,10	-	0	1	0	✓
8,10	1	0	-	0	✓
10,11	1	0	1	-	✓
10,14	1	-	1	0	✓
11,15	1	-	1	1	✓
14,15	1	1	1	-	✓

(c)

	w	x	y	z
0,2,8,10	-	0	-	0
0,8,2,10	-	0	-	0
10,11,14,15	1	-	1	-
10,14,11,15	1	-	1	-

Figura 2.9: Determinación de implicantes primos.

1. La tercera, cuarta y quinta secciones contienen los números binarios con dos, tres y cuatro números 1, respectivamente. Los equivalentes decimales de los minitérminos también se llevan para identificación.

Paso 2: Cualesquiera dos minitérminos que difieran uno del otro sólo por una variable pueden combinarse, y la variable que no compara se elimina. Dos números minitérminos caen en esta categoría si ambos tienen el mismo valor de bit en todas las posiciones excepto una. Los minitérminos en una sección se comparan con los de la siguiente hacia abajo solamente, debido a que dos términos que difieren por más de un bit no pueden compararse entre sí. El minitérmino en la primera sección se compara con cada uno de los tres minitérminos en la segunda sección. Si dos números cualquiera son los mismos en cada posición excepto una, se coloca una marca a la derecha de ambos minitérminos para mostrar que se han utilizado. El mismo resultante, junto con los equivalente decimales, se lista en la columna (b) de la Figura 2.9. La variable eliminada durante la comparación se indica con un guión en su posición original.

En este caso $m_0(0000)$ combina con m_1 para formar (000-). Esta combinación es equivalente a la operación algebraica:

$$m_0 + m_1 = w'x'y'z' + w'x'y'z$$

El minitérmino m_0 también se combina con m_2 para formar (00-0) y con m_8 para formar (-000). El resultado de esta comparación se coloca en la primera sección de la columna (b). Los minitérminos de las secciones dos y tres de la columna (a) se comparan a continuación para producir los términos que se listan en la sección de la columna (b). Todas las otras secciones de (a)

se comparan en forma similar y se forman las secciones subsecuentes en (b). Este proceso de comparación exhaustiva resulta en las cuatro secciones de (b).

Paso 3: Los términos de la columna (b) tienen sólo tres variables. Un 1 bajo la variable indica que no tiene prima, un 0 significa que tiene prima, y un guión significa que la variable no se incluye en el término. El proceso de búsqueda y comparación se repite para los términos en la columna (b) para formar los términos de dos variables de la columna (c). De nuevo, los términos en cada sección necesitan compararse sólo si tienen guiones en la misma posición. Observe que el término (000-) no compara con ningún otro término. Por tanto, no tiene marca de verificación a la derecha. Los equivalentes decimales se escriben en el lado izquierdo de cada entrada para propósitos de identificación. El proceso de comparación debe llevarse a cabo otra vez en la columna (c) y en las columnas subsecuentes, en tanto se encuentre una comparación apropiada. En este ejemplo, la operación se detiene en la tercera columna.

Paso 4: Los términos que no están marcados en la tabla forman los implicantes primos. En este ejemplo, se tiene el término w'x'y' (000-) en la columna (b), y los términos x'z' (-0-0) y wy (1-1-) en la columna (c). Cada término en la columna (c) aparece dos veces en la tabla y en tanto el término forma un implicante primo, es innecesario usar el término dos veces. La suma de los implicantes primos da una expresión simplificada de la función. Esto se debe a que cada término marcado en la tabla ha tomado en cuenta una entrada en un término más simple en una columna subsecuente. Por ende, las entradas no marcadas (implicantes primos) son los términos que se dejan para formar la función. Para este ejemplo la suma de los implicantes primos minimizada es una suma de productos:

$$F = w'x'y' + x'z' + wy$$

En este capítulo se han presentado los métodos con los cuales el ser humano pueden reducir expresiones de circuitos lógicos combinatorios. Vimos que el método de Karnaugh no es conveniente cuando se trata de reducir expresiones de más de cinco variables, por lo que se recomienda el uso del método de Quine-McCluskey para estos casos. Este método tiene la ventaja de seguir un procedimiento paso a paso; en cambio su desventaja de este otro método es que es más propenso a errores debido a lo monótono que puede llegar a ser, entre otras cosas.

Lo que se propone en este trabajo de tesis es una técnica de reducción en la que el humano no intervenga. Esta técnica se explica en el capítulo 4.

Capítulo 3

Programación Genética

La teoría de la evolución [10] fue formulada por Charles Darwin para explicar el proceso de adaptación de las especies por medio del principio de selección natural, el cual favorece a aquellos individuos que se adaptan más fácilmente a su entorno ambiental y por ende, serán los individuos que sobrevivirán. La teoría Neo-Darwinista considera también el concepto de herencia, siendo la fuente de inspiración para los algoritmos evolutivos.

Este interesante campo de la biología ha sido fuente de inspiración en el campo de las Ciencias de la Computación e Ingeniería para el desarrollo de métodos de optimización y búsqueda alternativos. La Computación Evolutiva (EC, por sus siglas en inglés), aplica la teoría de la evolución natural y la genética en la adaptación evolutiva de estructuras computacionales, proporcionando un medio alternativo para atacar problemas complejos en diversas áreas como en la ingeniería. Una población de posibles soluciones de un problema dado es análoga a una población de organismos vivos que evoluciona cada generación al recombinar los mejores individuos de la población y transmitir sus características de dichos individuos padres a sus descendientes. En este campo, diferentes esquemas de métodos evolutivos se han desarrollado, los cuales difieren en el tipo de estructuras que conforman la población.

Los algoritmos genéticos, propuestos por John Holland [25], son los más populares de los métodos evolutivos. El objetivo primordial de los algoritmos genéticos desarrollados por John Holland fue el estudio formal de los procesos de adaptación natural y de cómo estos mecanismos podrían ser trasladados al área del aprendizaje de máquina.

La Programación Genética [24] (GP, por sus siglas en inglés) es otra técnica evolutiva que goza de gran popularidad en la actualidad. Esta técnica fue propuesta por John Koza [14, 16] a finales de los 80's. John Koza en su libro titulado *"Genetic Programming: On the Programming of Computers by Means of Natural Selection"*, describe un método para la evolución de estructuras complejas como son los programas de computadora. El hecho de que muchos problemas prácticos de diferentes dominios de aplicación puedan ser formulados como un problema de determinación de un *"programa solución"* que produzca una salida deseada cuando se tienen presentes ciertas entradas particulares, hace a la Programación Genética una novedosa línea de investigación. Dentro de algunas de las aplicaciones prácticas de la Programación Genética se tienen el modelado e identificación de sistemas, procesamiento de señales e imágenes, **diseño de circuitos electrónicos**, control y robótica y predicción, entre otras.

3.1 Fundamentos de la Programación Genética

En Programación Genética, un programa esta formado a su vez por programas, individuos, cuya estructura depende de un conjunto de funcionales y de terminales. Los individuos varían de una generación a otra, ya que los operadores genéticos juegan un papel fundamental en la evolución de los mismos.

3.1.1 Conceptos Básicos.

La mayoría de los sistemas de programación genética tienen en común las siguientes características:

- *Decisión Estocástica.* La programación genética usa números pseudo-aleatorios para emular la evolución natural. Como resultado, la programación genética utiliza procesos estocásticos y decisiones probabilísticas.

- *Estructura del Programa.* La programación genética integra dos tipos de conjuntos: símbolos funcionales y terminales. Los funcionales ejecutan operaciones desde sus entradas, las cuales pueden ser terminales o salidas de otros funcionales. Desde el inicio de la ejecución del algoritmo, la población es inicializada con programas que son formados a partir de símbolos funcionales y terminales.

- *Operadores Genéticos.* La programación genética transforma los programas de la población inicial usando operadores genéticos. La cruza entre dos individuos es uno de los principales operadores genéticos en programación genética. Otros operadores genéticos importantes son la mutación y la reproducción.

- *Evolución de la Población.* La selección basada en la aptitud, determina qué programas de la población se eligen para fungir como los padres de la siguiente generación.

3.2 Terminales y Funcionales -Las primitivas de la Programación Genética.

Los símbolos funcionales y terminales, son las primitivas a partir de las cuales se construye un programa en programación genética. Los símbolos funcionales y terminales juegan diferentes papeles. A grandes rasgos podemos decir que los terminales proveen un valor al sistema, mientras que los funcionales procesan dicho valor. Tanto a los símbolos funcionales como a los terminales, se les denomina nodos cuando la representación de los programas sea en forma de árboles.

3.2.1 El conjunto de los Terminales.

El conjunto de los símbolos terminales está constituido por:

- Las entradas al programa: un individuo de la población puede ser $f = x + 2y$, en el cual las entradas de este programa son las variables x y y las cuales forman parte del conjunto de los símbolos terminales.

- Constantes que son suministradas a un programa: estas constantes no cambian su valor durante la ejecución del programa.

- Las funciones que no tienen ningún argumento o aridad 0.

En una representación basada en árboles, los símbolos terminales reciben este nombre ya que constituyen las hojas o nodos terminales de las ramas de los árboles.

3.2.2 El conjunto de los Funcionales.

El conjunto de los funcionales está compuesto por sentencias, operadores y funciones disponibles para los sistemas de programación genética.

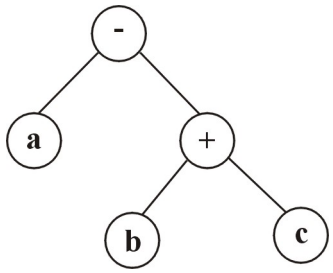

Figura 3.1: Rama de un árbol

Algunos ejemplos del conjunto de los funcionales son los siguientes:

- *Funciones Booleanas.*
  Ejemplo: `AND, OR, NOT, XOR`.
- *Funciones Aritméticas.*
  Ejemplo: `SUMA, RESTA, MULTIPLICACIÓN, DIVISIÓN.`
- *Funciones Trascendentes.*
  Ejemplo: `TRIGONOMÉTRICA Y FUNCIONES LOGARÍTMICAS.`
- *Funciones de Asigación a variables.*
  Ejemplo: `Asignación de valores a las variables.`
- *Sentencias Condicionales.*
  Ejemplo: `If, Then, Else; Case o Switch.`
- *Sentencias de Control de Transferencia.*
  Ejemplo: `Go to, Call, Jump.`
- *Sentencias de Ciclos.*
  Ejemplo: `While...Do, Repeat...Until, For...Do.`

3.2.3 Selección del conjunto de los Terminales y Funcionales.

El conjunto de los terminales y funcionales usados en la programación genética deben ser lo suficientemente poderosos para ser capaces de representar la solución de un problema. Por ejemplo, el conjunto de los funcionales que contenga sólo el operador de adición muy probablemente no podrá resolver problemas complejos. Por otra parte, es mejor no tener un conjunto de funcionales muy grande, ya que esto podría aumentar el espacio

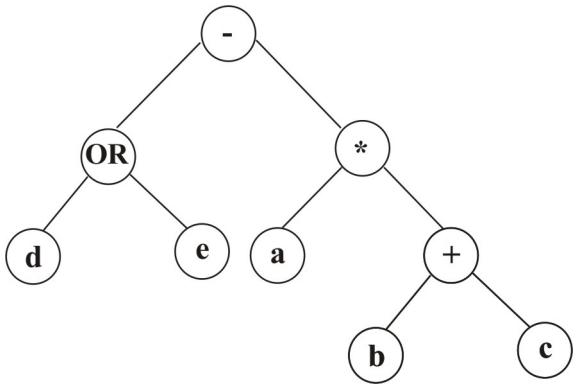

Figura 3.2: Estructura de un árbol

de búsqueda y por ende, hacer que la solución sea más difícil de encontrar.

Una propiedad importante de los conjuntos de los funcionales es que cada función sea capaz de manejar todos los valores que recibe desde las entradas. Esto es conocido como la propiedad de *Cierre*. Todas las funciones deben ser capaces de aceptar todas las posibles entradas porque de no ser así se colapsaría el sistema.

3.3 Estructura de un Programa Ejecutable.

Las primitivas de la programación genética -funcionales y terminales- no son programas sino más bien componentes de los mismos. Los funcionales y terminales deben ser integrados en una estructura antes de que éstos puedan ser ejecutados como programas. La selección de la estructura de un programa en programación genética afecta el orden de ejecución, uso de la memoria y la aplicación de los operadores genéticos al programa. Las tres principales estructuras utilizadas en programación genética son: estructuras de árboles, lineales y grafos.

3.3.1 Ejecución de la Estructura de un árbol.

La Figura 3.2 muestra la estructura de un árbol. En esta figura se tienen diferentes símbolos que pueden ser ejecutados en cualquier orden. Pero existen convenciones para la ejecución de estructuras de árboles.

La ejecución convencional de la estructura de un árbol consiste en evaluar

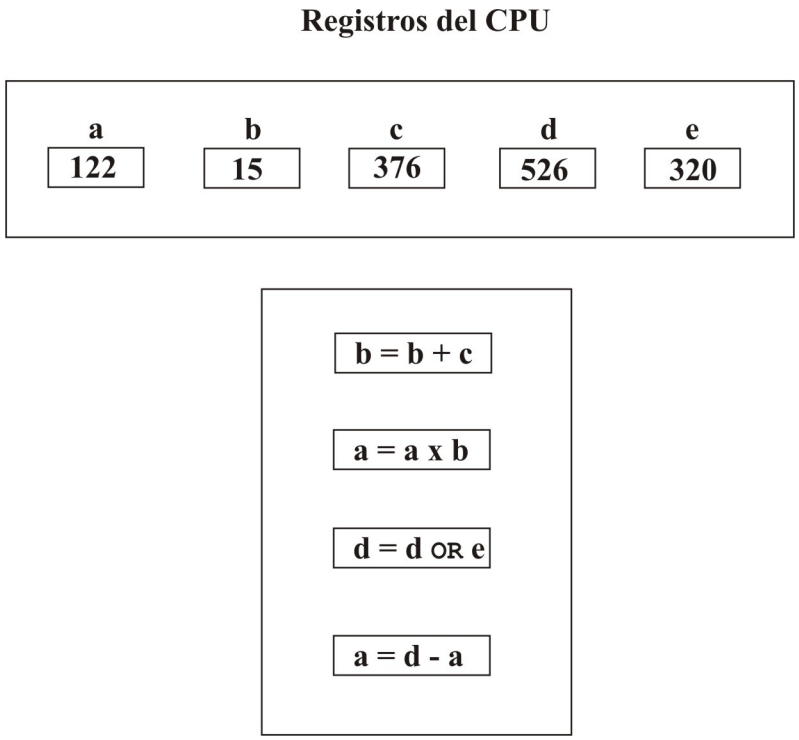

Figura 3.3: Estructura Lineal.

repetidamente el nodo que está más a la izquierda para todas las entradas disponibles. Este orden de ejecución es denominado **Orden Postfijo**, llamado así porque los operadores aparecen después de los operandos. Otra ejecución convencional es el llamado **Orden Prefijo**, y se le llama así porque los operandos aparecen después de los operadores.

Aplicando el Orden Postfijo a la Figura 3.2, el orden de ejecución de los nodos es el siguiente:
d→ e → or → a → b → c → + → mul → -.

3.3.2 Ejecución de la Estructura Lineal.

Una estructura lineal es simplemente una cadena de instrucciones que se ejecuta de izquierda a derecha, o bien, de arriba hacia abajo dependiendo de cómo se interprete dicha representación.

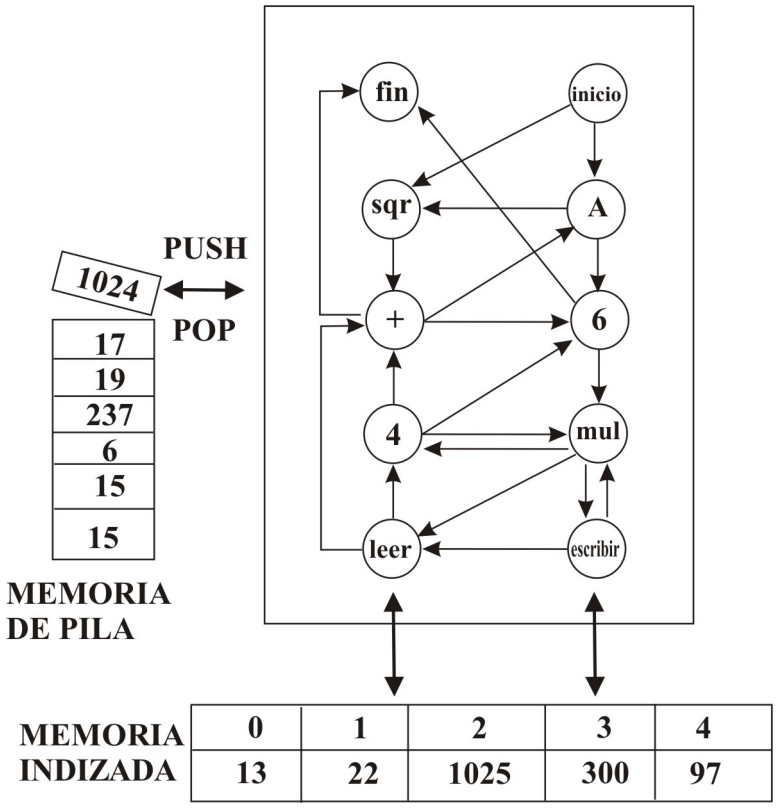

Figura 3.4: Estructura de un Grafo.

Un programa lineal empieza su ejecución en la instrucción del tope o de la izquierda, según esté representado y así continúa hacia abajo o hacia la derecha. Este tipo de representaciones permite ejecutar instrucciones de salto por lo que los hace flexibles en su ejecución. Al finalizar la ejecución del programa representado en la Figura 3.3 el resultado es almacenado en el registro a.

3.3.3 Ejecución de la Estructura de Grafo.

Los grafos son capaces de representar un programa muy complejo con estructuras compactas. La estructura de un grafo no es otra cosa que nodos conectados por aristas. Una arista es la conexión entre dos nodos, indicando la dirección del flujo del programa.

Las estructuras de árboles y programas lineales, son también grafos, sólo que éstos presentan restricciones particulares para la forma en cómo se conectan los nodos a través de las aristas.

PADO es el nombre de un sistema en PG basado en grafos. Aquí no se permiten los ciclos ni las recursiones por las dificultades que pueden causar, como el hecho de que se puede ciclar el programa.

La ejecución del programa empieza con el nodo etiquetado con `START` y se sigue la dirección de las aristas las cuales determinan el flujo de la ejecución del programa, hasta llegar al nodo etiquetado con `END`.

3.4 Inicialización de la Población.

El primer paso para llevar a cabo la ejecución de un programa de la programación genética es inicializar la población. Esto significa crear una variedad de estructuras de programas para una evolución posterior. Este proceso es diferente para cada uno de los tres genomas mencionados en la sección 3.3. Uno de los principales parámetros de la programación genética es el tamaño máximo permitido para un programa. Para los árboles en programación genética, los parámetros están expresados en la máxima profundidad del árbol o el número máximo total de los nodos de un árbol.

El Parámetro de Máxima Profundidad (PMP) es la profundidad más larga permitida entre el nodo raíz y los nodos terminales. En general, para los nodos de aridad 2, el tamaño del árbol tendrá un número máximo de 2^{PMP} nodos. En la representación lineal, el parámetro es llamado longitud máxima y se traduce en el número máximo de instrucciones permitidas en un programa. Para los grafos en programación genética, el número máximo de nodos es equivalente al tamaño del programa.

3.4.1 Inicialización de las Estructuras con árboles.

Hay que recordar que los árboles son construidos por unidades llamadas funcionales y terminales. Por el momento, supondremos que los terminales y funcionales disponibles en los árboles del programa son los siguientes:

- T = {a,b,c,d,e}
- F = {+,-,x,%}

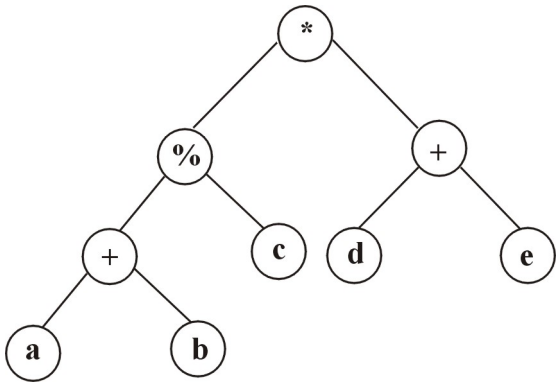

Figura 3.5: Máxima Profundidad de 4, inicializado con el método "grow"

Existen dos métodos diferentes para la inicialización de las estructuras de los árboles:

- *Full:* Mediante este método los árboles tiene forma regular, es decir, es un árbol balanceado.
- *Grow:* Mediante este método los árboles tienen forma irregular.

En la Figura 3.5, la rama que termina con la entrada **d** tiene una profundidad de 3. Esto se debe a la incidencia de seleccionar terminales de forma aleatoria.

Con el método completo, se seleccionan nodos aleatorios desde el conjunto de terminales y funcionales. En este método se seleccionan sólo funcionales hasta el nodo que representa la máxima profundidad, y cuando se llega a este punto entonces se seleccionan sólo terminales. El resultado de este método es que cada rama del árbol cumple con la máxima profundidad.

En la Figura 3.6 se muestra un árbol que ha sido inicializado por el método completo con una profundidad máxima de 3.

3.4.2 El método de mitad y mitad.

La diversidad es valiosa en las poblaciones de la programación genética. Cuando se usa el método completo, puede ocurrir que se produzca un conjunto uniforme de estructuras en la población inicial ya que la rutina es la misma para todos los individuos. Para prevenir esto, se ha diseñado la

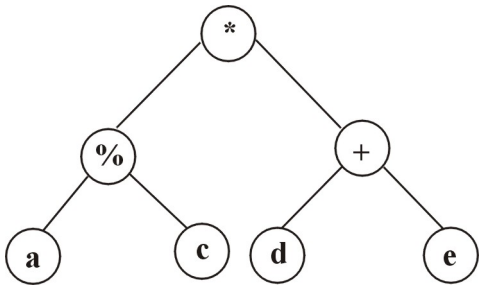

Figura 3.6: Máxima Profundidad de 3, inicializado con el método "full"

técnica llamada "mitad y mitad".

La técnica consiste en lo siguiente: Supongamos que el parámetro de máxima profundidad es de 6. La población es dividida entre todos los individuos para ser inicializada con árboles que tengan profundidades de 2, 3, 4, 5 y 6. Para cada una de estas profundidades, la mitad de los árboles serán inicializados con la técnica completa y la otra mitad con la técnica que varían su forma *("grow").*

3.5 Operadores Genéticos.

Una vez que la población ha sido inicializada, usualmente la aptitud de la misma es muy baja pues estas soluciones son generadas al azar. A partir de esta población se procede a emular el proceso evolutivo transformando dicha población inicial con los operadores genéticos.

Entre los operadores podemos encontrar:

- Cruza,
- Mutación, y
- Reproducción.

3.5.1 Cruza.

El operador genético denominado cruza combina el material genético de dos padres intercambiando una parte de uno de ellos con otra parte del otro. La cruza basada en árboles se muestra en la Figura 3.7. Los padres son

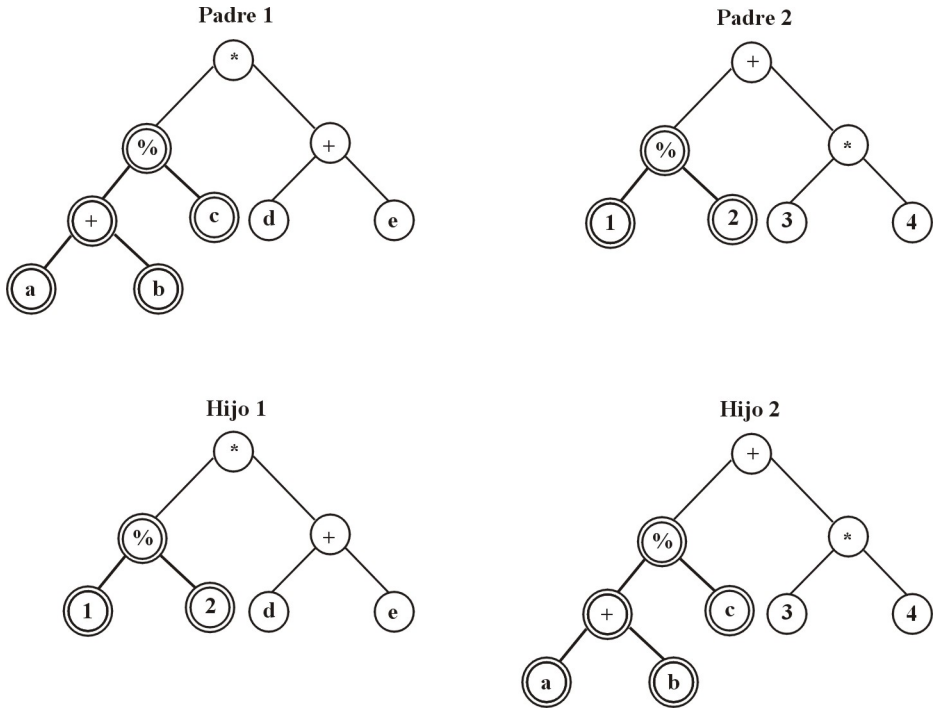

Figura 3.7: Representación de cruza entre dos individuos.

mostrados en la parte superior, mientras que los hijos son los que están en la parte inferior. La cruza en los árboles funciona de la siguiente manera:

- Se seleccionan dos individuos como padres, utilizando un mecanismo generalmente probabilístico. Los dos padres se muestran en la parte superior de la Figura 3.7.
- Se seleccionan aleatoriamente subárboles de cada padre. En la Figura 3.7 los subárboles seleccionados son mostrados con dobles círculos y con líneas más gruesas. La selección de los subárboles puede ser sesgada, de forma que los subárboles que sean terminales tengan baja probabilidad de ser seleccionados.
- Se intercambian los subárboles seleccionados entre los dos padres. El resultado de estos individuos son los hijos. Éstos se muestran en la Figura 3.7

3.5.2 Mutación.

El operador de mutación es aplicable a un solo individuo. Cada hijo producido por la cruza es sometido a mutación con una baja probabilidad definidida por el usuario. Una aplicación separada de cruza y mutación, sin embargo, también es posible y puede resultar adecuada en algunos casos.

La mutación para árboles se muestra en la Figura 3.8. El operador genético denominado Mutación reemplaza un subárbol por otro, pero cuidando que el nuevo individuo no exceda la profundidad máxima permitida.

La mutación aplicada a un individuo se realiza de la siguiente manera:

- El individuo es seleccionado de acuerdo a un porcentaje, el cual generalmente es bajo. El individuo original es el que se muestra en la parte superior, mientras que el resultante es el que se muestra en la parte inferior de la Figura 3.8.
- Se selecciona aleatoriamente un subárbol del individuo. En la figura 3.8 el subárbol seleccionado es mostrado con dobles círculos y con líneas más gruesas.
- Se reemplaza el subárbol seleccionado por otro que se crea aleatoriamente.
- Se verifica la profundidad máxima permitida y si ésta es rebasada, entonces se procede de nuevo con la mutación. De lo contrario, el nuevo individuo será aquel que tenga el subárbol generado aleatoriamente.

3.5.3 Reproducción.

Un individuo es seleccionado. Éste es copiado y la copia se coloca en la población nueva.

3.6 Aptitud y Selección.

La programación genética es un tipo de búsqueda dirigida. La PG debe seleccionar cuáles miembros de la población serán afectados por los operadores genéticos (cruza, mutación, reproducción). Una vez que se ha realizado lo anterior, la PG implementa una de las partes más importantes del modelo de aprendizaje evolutivo, la selección basada en la aptitud. La

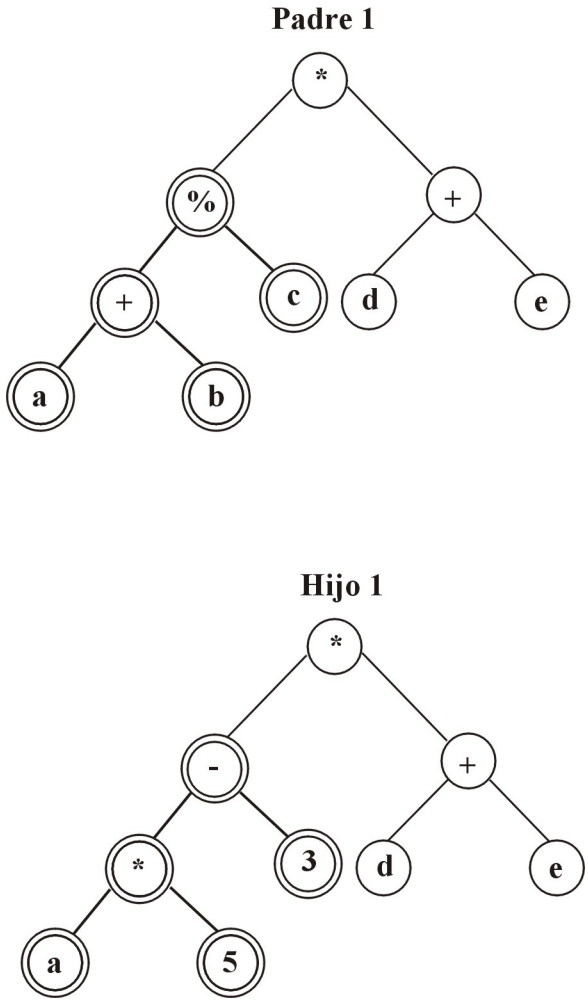

Figura 3.8: Mutación a un individuo.

métrica de evaluación en PG es llamada función de aptitud. Ésta juega el papel del ambiente, determinando qué individuos sobrevivirán y cuáles no.

3.6.1 Función de Aptitud.

La aptitud es la medida usada por la PG durante la simulación del proceso de evolución, y determina qué tan bien ha aprendido el programa a predecir las salidas a partir de las entradas. Esto es, las características del dominio de aprendizaje.

La evaluación de la función de aptitud tiene como objeto dar una retroalimentación al algoritmo de aprendizaje relativo a cuales individuos tendrán una mayor probabilidad de multiplicarse y reproducirse y cuáles individuos tendrán mayor probabilidad de ser removidos de la población.

Los problemas que resuelve de manera directa la PG son de regresión simbólica. En este tipo de problemas, se pretende generar una expresión matemática (combinando los operadores y los terminales disponibles) que aproxime la curva definida por los datos proporcionados por el usuario. Ejemplos de regresión simbólica:

- Número de ejemplos clasificados correctamente en una tarea de clasificación.
- La desviación entre lo pronosticado y realidad en una aplicación de predicción.
- La cantidad de comida encontrada y consumida por un agente artificial en una aplicación de vida artificial.

3.6.2 El Algoritmo de Selección.

Después de que se ha determinado la calidad de un individuo aplicando la función de aptitud, se decidirá si se aplican o no los operadores genéticos a un individuo y si se debe o no mantener en la población o permitir que éste sea reemplazado. Lo anterior se denomina, Operador de Selección.

Existen diferentes métodos de selección, y es tarea del usuario de la PG el decidir el método de selección que se aplicará tomando en cuenta las circunstancias específicas del problema. La selección es la responsable de la velocidad de evolución y, de no manejarse adecuadamente, puede provocar

convergencia prematura. La selección en general, es una consecuencia de la competencia entre los individuos de una población.

3.7 Algoritmo Básico de PG.

Existen dos maneras de efectuar el reemplazo poblacional en la PG: el método generacional y el de estado uniforme (o no generacional).

En el método generacional, la nueva población se forma a partir de la anterior y reemplaza totalmente a ésta.

En el método de estado uniforme, sólo unos pocos individuos son seleccionados y sujetos a cruza y mutación. Los descendientes producidos reemplazan a un número igual de la población (tal vez a los peores). El concepto de generación no existe en este caso.

A continuación se dan los pasos preliminares de la PG:

- Definir el conjunto de los terminales.
- Definir el conjunto de los funcionales.
- Definir la función de aptitud.
- Definir los parámetros, tales como: el tamaño de la población, tamaño máximo del individuo, probabilidad de cruza, método de selección y el criterio de detención.

Versión Generacional de la PG.

1. Inicializar la población.
2. Evaluar a los individuos existentes en la población. Calcular la aptitud de cada individuo.
3. Hasta que la nueva población no esté completa, repetir los siguientes pasos:
    - Seleccionar un individuo o individuos en la población utilizando el algoritmo de selección elegido.
    - Aplicar los operadores genéticos sobre el individuo o individuos seleccionados.

- Colocar a los descendientes producto de cruza y mutación en la nueva población.

4. Si el criterio de término se satisface se detiene el proceso; en caso contrario, reemplazar a la población existente con la nueva población y se repiten los pasos 2-4.

5. Reportar al mejor individuo de la población.

Versión de Estado Uniforme de la PG.

1. Inicializar la población.

2. Seleccionar aleatoriamente un subconjunto de la población para que ésta tome parte en un torneo (competidores).

3. Evaluar la aptitud de cada competidor en el torneo.

4. Seleccionar al ganador o ganadores del torneo utilizando el algoritmo de selección.

5. Aplicar los operadores genéticos sobre el ganador o ganadores en el torneo.

6. Reemplazar a los perdedores del torneo con los resultados de la aplicación de los operadores genéticos por los ganadores de los torneos.

7. Repetir los pasos 2-7 hasta que el criterio de detención sea satisfecho.

8. Reportar al mejor individuo de la población.

En este trabajo se utiliza la programación genética con una representación lineal con expresiones postfijas, que permiten el diseño, la prueba y evolución para producir expresiones que representen el circuito objetivo. De esta forma se ha desarrollado un método de reducción de expresiones de circuitos lógicos combinatorios, que no requieren de la intervención del humano.

Capítulo 4

Descripción de la Técnica.

En este capítulo se explica como se aborda el problema de diseñar circuitos lógicos combinatorios utilizando la Programación Genética con expresiones Postfijas.

A continuación se presentará la representación propuesta en este trabajo, así como la forma en la que se aborda el problema de manejar circuitos con más de una salida. Adicionalmente se describe cómo se lleva a cabo la evaluación de la función de aptitud de cada uno de los individuos.

4.1 Representación de los individuos.

La representación de circuitos lógicos combinatorios puede ser representado a través de tablas de verdad, diagramas, funciones booleanas, etc. Por ejemplo si se quiere representar un circuito en forma de función booleana, quedaría de la siguiente forma:

$$F = x \cdot y$$

Donde $\cdot$ representa la operación lógica AND. De tal forma, que la representación gráfica se muestra en la Figura 4.1.

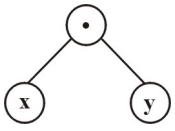

Figura 4.1: Representación infija de una expresión booleana.

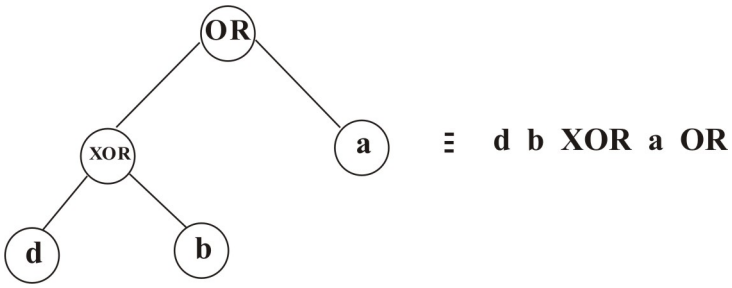

Figura 4.2: Representación postfija de una expresión booleana.

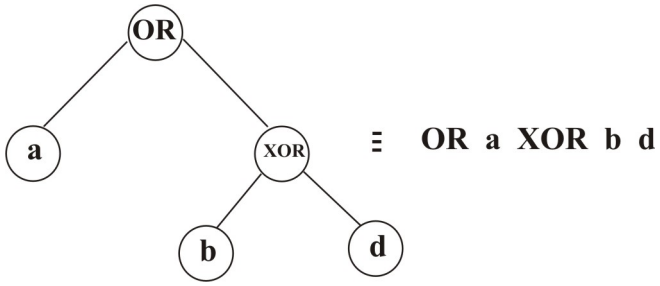

Figura 4.3: Representación prefija de una expresión booleana.

La función booleana $F = x \cdot y$ esta basada en una expresión infija. Este tipo de expresiones no son interpretadas fácilmente por los ordenadores, por lo que se debe basar en otro tipo de expresiones, como por ejemplo:

- **Prefija.** En este tipo de representaciones los operadores van antes que los operandos.
- **Postfija.** En este tipo de representaciones los operadores van después que los operandos.

Considérese la siguiente función booleana:

$$F = (d\ XOR\ b)\ OR\ a$$

En la Figura 4.2 se muestra el árbol de la expresión anterior en forma postifja mientras que en la Figura 4.3 se muestra el árbol de la expresión prefija. En ambas figuras, del lado derecho, también se muestra la expresión equivalente al árbol.

Para representar un individuo en Programación Genética se debe definir un conjunto de símbolos funcionales y terminales. Para representar a

Operadores Booleanos	*Aridad*
NOT	1
OR	2
AND	2
OR-exclusivo	2

Tabla 4.1: Símbolos funcionales utilizados en la implementación.

los circuitos lógicos el cojunto de símbolos funcionales está fomado por operadores booleanos, los cuales se muestran la Tabla 4.1.

El conjunto de los terminales está representado por las literales, que representan las entradas de los circuitos. Por ejemplo para un circuito de 3 entradas el conjunto de los terminales está formado por las literales: a,b,c. Ya que se ha definido el conjunto de los símbolos funcionales y terminales, un individuo puede ser representado de la siguiente forma: $db3a1R$. Esta representación está hecha con base en la representación postfija. En este caso los símbolos terminales están representados por las letras minúsculas, mientras que los símbolos funcionales están representados por los números; la letra R indica la raíz del individuo representado en forma de árbol. Ahora bien, los números a su vez representan las operadores booleanos mostrados en la Tabla 4.1.

La representación anterior es de un individuo que trata de resolver un circuito lógico combinatorio de una salida, pero si el circuito que se trata de resolver tiene más de una salida, entonces la representación del individuo varía en el número de R's que contenga. Esto es, el individuo tendrá tantas R's como salidas tenga el circuito que se trate de resolver. Por ejemplo, un individuo que trate de resolver un circuito de tres salidas tendrá la siguiente representación: $db3a1Rab2c1Rdb1R$.

4.2 Evaluador de expresiones.

Para poder evaluar cada individuo, es necesario el uso de un *evaluador de expresiones* (intérprete de genomas). Este evaluador de expresiones está basado en una pila, cuya representación gráfica se muestra en la Figura 4.4.

El funcionamiento del evaluador de expresiones se explica a continuación:

- Si el caracter a leer es una letra, entonces se almacena en la pila su valor correspondiente con la tabla de verdad.

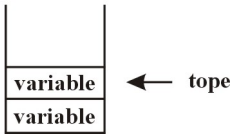

Figura 4.4: Representación gráfica de un evaluador de expresiones.

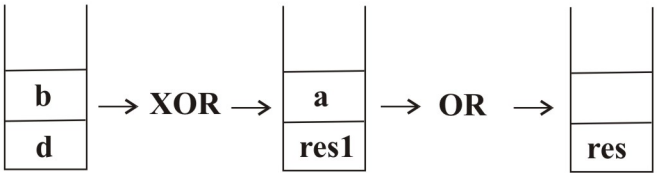

Figura 4.5: Funcionamiento del evaluador de expresiones.

- Si el caracter a leer es un número, entonces se sacan de la pila los valores que se evaluarán y el resultado es almacenado en el tope (parte superior) de la pila.
- Este procedimiento continúa hasta que se haya terminado de evaluar la expresión.

En la Figura 4.5 se muestra gráficamente la forma en cómo funcionará el evaluador de genomas para la expresión *d b XOR a OR*

4.3 Función de Aptitud.

Una vez que los individuos han sido generados se debe de determinar qué individuos sobrevivirán y cuáles no, por lo que se debe aplicar una métrica de evaluación, que en programación genética es llamada función de aptitud.

Ahora bien, la aptitud de cada uno de los individuos se determina contando el número de bits acertados con respecto a la tabla de verdad. Mientras más bits sean acertados, mayor será la aptitud del individuo.

$$Aptitud = Acertados + Bonificación.$$

Una vez que la aptitud del individuo es igual al número de bits correspondientes a la tabla de verdad, lo que se hace es bonificar a dicho

individuo; ésto se hace con la finalidad de optimizarlo.

La bonificación se realiza de la siguiente manera: se resta de la longitud máxima permitida el tamaño del individuo, y a esto se le resta el número de operadores que contiene dicho individuo. Esto es porque entre menor número de operadores contenga el individuo, menor número de compuertas tendrá.

Bonificación = Lmax - Lind - No. de operadores.

4.4 Operadores Genéticos.

Dentro de programación genética existen diversos operadores genéticos, entre los que podemos encontrar los siguientes:

- *Selección.* Existen diversos métodos de selección, como por ejemplo: Selección de Boltzmann, Selección mediante Torneo, Selección de Estado Uniforme, entre otras. En esta implementación se decidió utilizar la *selección mediante torneo (versión determinística).* Este tipo de selección se aplica de la siguiente manera:

  1. Barajar los individuos de la población.
  2. Escoger un número P de individuos (típicamente dos).
  3. Compararlos con base en su aptitud.
  4. El ganador del "torneo"es el individuo más apto.
  5. Se debe barajar la población un total de P veces para seleccionar N padres.

  En esta selección se garantiza que el mejor individuo será seleccionado P veces. Cada competencia requiere la selección aleatoria de un número constante de individuos de la población. La comparación entre estos individuos puede realizarse en tiempo constante. Se requieren n competencias de este tipo para completar una generación, por lo que el algoritmo es de $O(n)$.

- *Cruza.* La cruza es uno de los operadores genéticos más importantes dentro de la programación genética. Este operador es el encargado de explotar el espacio de búsqueda y funciona de la siguiente forma:

  1. Se seleccionan dos individuos de la población.

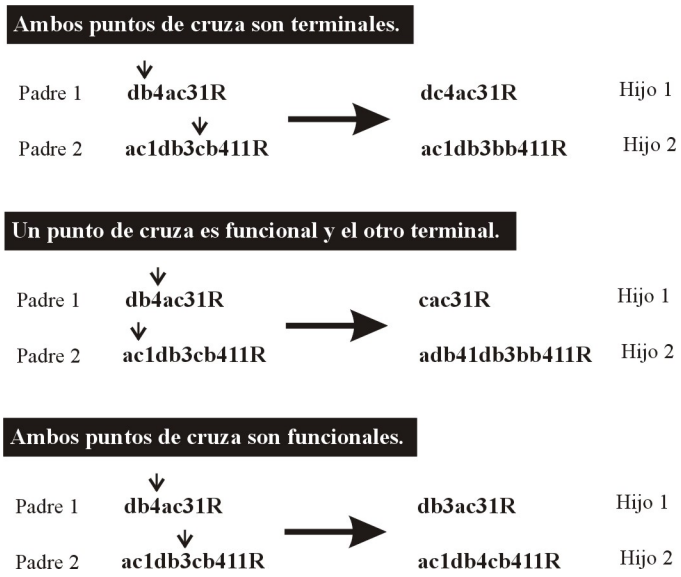

Figura 4.6: Diferentes puntos de Cruza.

2. Se generan puntos de cruza al azar dentro de las cadenas.
3. Se determina la longitud de las subcadenas que serán intercambiadas en cada uno de los dos individuos seleccionados; si la longitud no excede la longitud máxima permitida entonces se procede a dicho intercambio; de lo contrario se repite el proceso desde el punto 2.

En el proceso de cruza se consideran tres casos, los cuales son:

1. Cuando ambos puntos de cruza son terminales.
2. Cuando un punto de cruza es terminal y el otro punto es funcional.
3. Cuando ambos puntos de cruza son funcionales.

En el caso 1 se asigna un bajo porcentaje para que ambos puntos de cruza sean terminales, ya que la explotación con este tipo de puntos de cruza no es significativa. En la Figura 4.6 se muestran los diferentes casos que se deben considerar.

- *Mutación.* Este operador genético es el encargado de llevar a cabo la exploración en el espacio de búsqueda y funciona de la siguiente forma:

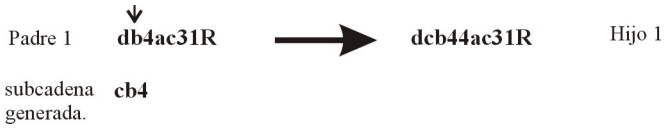

Figura 4.7: Aplicación de mutación a un individuo.

1. Se selecciona un individuo de la población.
2. Se genera un punto de mutación al azar dentro de la cadena.
3. Se determina la longitud de la subcadena que reemplazará a la otra subcadena. En caso de que no exceda la longitud máxima permitida, entonces se procede a hacer dicho reemplazo. En caso contrario, se repite el proceso desde el punto 2.

Al igual que en la cruza, en la mutación se asigna un bajo porcentaje para que el punto de mutación sea terminal, ya que la exploración con este tipo de punto de mutación no es significativa. En la Figura 4.7 se muestra cómo funciona la mutación.

- *Elitismo.* Este operador genético es el encargado de garantizar la transferencia genética del mejor individuo de la población de la actual generación a la siguiente generación. Por mejor individuo entiéndase aquél que tenga mayor aptitud.

4.5 Método de Encapsulamiento.

Como el objetivo principal de esta tesis es resolver circuitos lógicos combinatorios de más de una salida se propuso la reutilización de código de circuitos, para lo cual se planteó la siguiente idea:

- Se generan individuos en forma postfija, con base en terminales y funcionales. Además de las letras que se contemplan en el conjunto de los terminales para mapear las entradas con las que cuente la tabla de verdad, se agregará una letra más que es la p y ésta se emplea para llevar a cabo dicha reutilización de código. La Figura 4.8 muestra la forma en que la letra p lleva a cabo el encapsulamiento.

- Una vez que se han generado los individuos de la población, se verifica si éstos cuentan con *p's*. En caso de contener en sus cadenas dicha letra, se procede a asignar un punto dentro de la misma que es donde

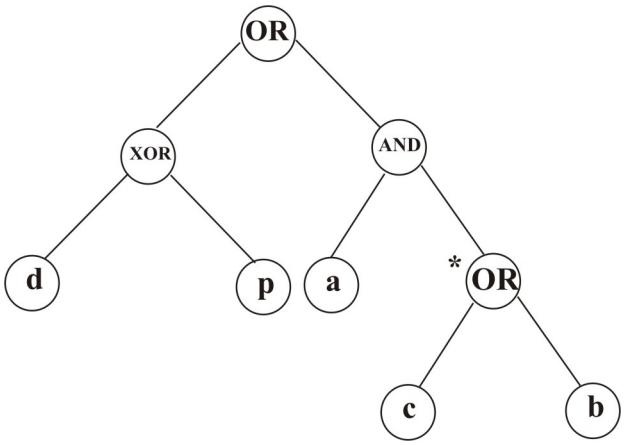

Figura 4.8: Encapsulamiento representado con la letra p.

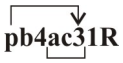

Figura 4.9: Asignación de puntos para las *p's* de circuitos con una salida.

va a hacer referencia la letra *p*, lo que da como resultado la reutilización de código. En caso de que se haga referencia a un terminal sólo se sustituye la *p* por dicho terminal; en caso de que se haga referencia a un funcional, entonces se sustituirá por la subcadena que abarque ese funcional. En la Figura 4.9 se muestra un ejemplo de cómo se asignan los puntos a los que hace referencia una p y en la Figura 4.10 se muestra un ejemplo de cómo se asignan los puntos a los que hacen referencias las *p's* en el caso de múltiles salidas.

- En caso de que el punto al que hace referencia la *p* contenga otra *p*, se procede a hacer el mismo procedimiento descrito en el punto anterior. Este proceso termina hasta que a todas las *p's* que estén contenidas dentro de un individuo se les haya asignado un punto dentro de la cadena.

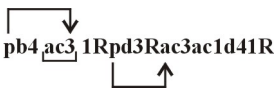

Figura 4.10: Asignación de puntos para las *p's* de circuitos con múltiples salidas.

Otros puntos importantes que se tomaron en cuenta para la implementación son los siguientes:

- La función de aptitud utilizada para esta implementación es la descrita en la sección 4.3.
- Se utilizó la selección mediante torneo (versión determinística) descrita en la sección 4.4.
- Se aplica un operador genético denominado *EGL* cuya idea básica es la siguiente: antes de aplicar elitismo, se hacen tres asignaciones diferentes a las posiciones a las que hacen referencias las letras *p's*. Se evalúa el individuo y aquella asignación que haya sido la mejor, en términos de aptitud, será la que se guardará.
- Se aplica elitismo a un solo individuo de la población.

4.6 Método Poblacional y Adaptación en Línea.

El método poblacional toma como base el método de encapsulamiento. Las diferencias se describen a continuación:

- Se utiliza una función de aptitud parecida a la descrita en la sección 4.3. La única diferencia estriba en que en este método se evalúa columna por columna de la tabla de verdad, en vez de evaluar toda la tabla de verdad, que es lo que se hace en el método de encapsulamiento. Esto da como resultado tener un enfoque poblacional, ya que se tendrán $n + 1$ supoblaciones, donde n denota el total de columnas de la tabla de verdad.
- Cada subpoblación se encargará de cada una de las columnas de la tabla de verdad, y la última subpoblación (o primera, según se quiera ver) se encargará de resolver toda la tabla de verdad.
- Se implementó la adaptación en línea en este método, la cual consiste en lo siguiente: se comienza con una cierta longitud máxima permitida, si cierto número de generaciones no se mejora el desempeño (en términos de aptitud) entonces se deja crecer la longitud 20 alelos más, y así sucesivamente hasta llegar a la longitud global máxima permitida que se haya definido, al llegar a este punto se incrementa el porcentaje de mutación en 0.025 hasta llegar a .4, y una vez que se llega hasta este porcentaje de mutación lo que hace es incrementar el porcentaje

de cruza en 0.025 y esto se hace sucesivamente hasta que se llega a un porcentaje de cruza de 1. Si se llega a este punto, se reinician los valores de los porcentajes de cruza y mutación.

Los dos métodos anteriormente descritos constituyen la metodología con que se aborda el problema de resolver circuitos lógicos combinatorios de más de una salida. En el capítulo 6, se muestran los resultados alcanzados por cada una de las dos técnicas.

Capítulo 5

Circuitos de una Salida

En este capítulo se presentan los resultados de 3 circuitos de diferente grado de complejidad, dichos circuitos son de x entradas y y salidas, donde x es mayor a 1 y y es igual a 1. Los circuitos seleccionados en este capítulo, son problemas even-n-parity, donde n va de 3 a 5. El conjunto de símbolos funcionales utilizados son: *AND, OR, NAND y NOR*; mientras que el conjunto de símbolos terminales consiste de tantas letras como entradas contenga el circuito a resolver.

El problema del even-n-parity ha sido seleccionado porque es un problema difícil y ha sido utilizado frecuentemente por investigadores de computación evolutiva. Mientras más entradas tenga el circuito, más difícil se vuelve el problema.

5.1 Experimentos.

Se realizaron 20 corridas para ver el desempeño que tuvo el algoritmo en estos circuitos even-n-parity, se utilizó un porcentaje de cruza de 90% y un porcentaje de mutación de 10%. En lo que se refiere al número de generaciones, tamaño de la población y la longitud máxima permitida se mencionarán en cada uno de los ejemplos de los circuitos de una salida. Estos últimos parámetros se determinaron con base en distintas pruebas. **Cabe hacer mención que el parámetro de longitud máxima permitida determina en gran medida el desempeño que tendrá el algoritmo.**

a	*b*	*c*	*S1*
0	0	0	1
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 5.1: Tabla de verdad para el even-*3*-parity.

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	240
Número máximo de generaciones	400
Longitud máxima permitida	180
Zona factible	65%
Mejores resultados	10% con 9 compuertas

Tabla 5.2: Parámetros utilizados en el método sin encapsulamiento para el ejemplo even-*3*-parity.

5.2 Método sin Encapsulamiento.

5.2.1 Ejemplo 1. Even-*3*-Parity.

El primer ejemplo consiste en el circuito even-*3*-parity, que consta de 3 entradas y 1 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 5.1. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la tabla 5.2.

Las estadísticas son mostradas en la Tabla 5.3 y en la Tabla 5.4 se muestra el análisis de las 20 corridas. El circuito lógico de 9 compuertas es mostrado en la Figura 5.1 mientras que el desempeño se muestra en la Figura 5.2. Estos resultados son comparados contra los reportados por

Media	*D.E.*	*Mediana*
7.6347	0.4893	8

Tabla 5.3: Estadísticas del even-*3*-parity con el método sin encapsulamiento.

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	7	ND
2	8	9
3	7	ND
4	7	ND
5	8	12
6	8	11
7	8	9
8	8	12
9	7	ND
10	8	11
11	7	ND
12	8	10
13	8	10
14	8	10
15	6	ND
16	8	10
17	8	11
18	8	12
19	7	ND
20	8	10

Tabla 5.4: Desempeño del even-*3*-parity con el método sin encapsulamiento. ND = No se alcanzó la Zona Factible.

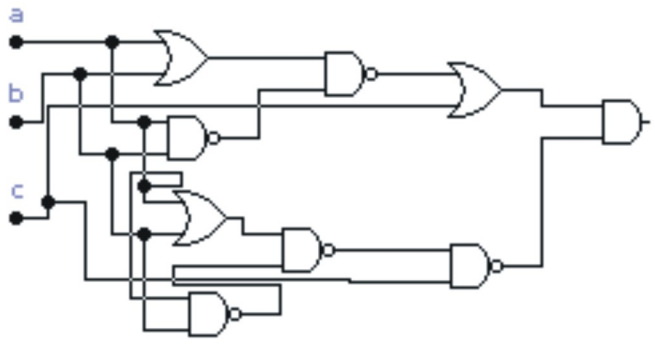

Figura 5.1: Even-*3*-parity con el método sin encapsulamiento.

Técnica	*No. Compuertas*	*Compuertas*	*Evaluaciones*
PG Postfija	9	AND, OR, NAND, NOR	96,000
PG De Jong	9	AND, OR, NAND, NOR	72,044
MO De Jong	9	AND, OR, NAND, NOR	42,965

Tabla 5.5: Comparación de los resultados encontrados por la PG Postfija y la PG Prefija Multiobjetivo de De Jong para el even-*3*-parity.

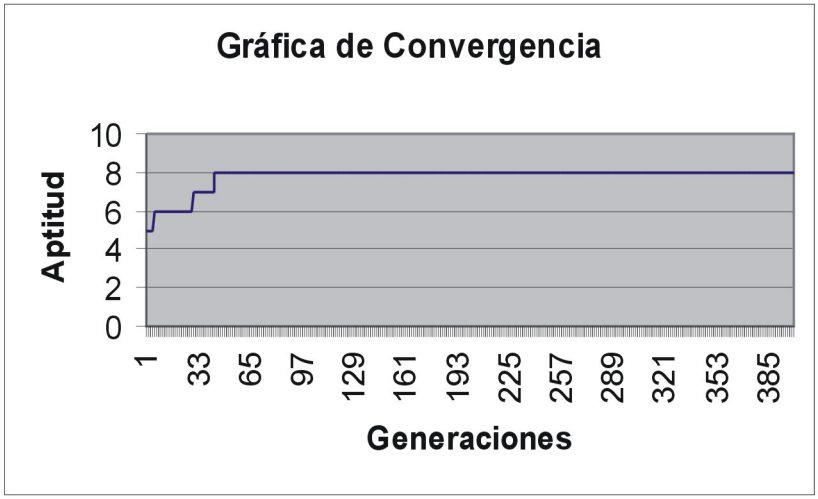

Figura 5.2: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del even-*3*-parity con el método sin encapsulamiento.

Edwing D. de Jong [27] los cuales se muestran en la Tabla 5.5.

En la Figura 5.3 se muestra el circuito correspondiente al even-*3*-parity encontrado por De Jong con su método multiobjetivo.

5.2.2 Ejemplo 2. Even-*4*-Parity.

El segundo ejemplo consiste en el circuito even-*4*-parity, que consta de 4 entradas y 1 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 5.6. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la tabla 5.7.

Las estadísticas son mostradas en la Tabla 5.8 y en la Tabla 5.9 se muestra el análisis de las 20 corridas. El circuito lógico con 26 compuertas es mostrado en la Figura 5.4 mientras que el desempeño se muestra en la Figura 5.5. Estos resultados son comparados contra los reportados por

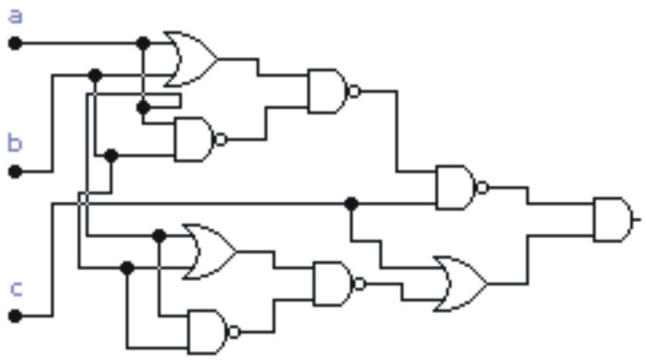

Figura 5.3: Even-*3*-parity con el método multiobjetivo encontrado por De Jong.

a	*b*	*c*	*d*	*S1*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Tabla 5.6: Tabla de verdad para el even-*4*-parity.

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	800
Número máximo de generaciones	800
Longitud máxima permitida	380
Zona factible	20%
Mejores resultados	5% con 26 compuertas

Tabla 5.7: Parámetros utilizados en el método sin encapsulamiento para el ejemplo even-*4*-parity.

Media	*D.E.*	*Mediana*
14.2303	1.4179	15

Tabla 5.8: Estadísticas del even-*4*-parity con el método sin encapsulamiento.

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	15	ND
2	15	ND
3	15	ND
4	14	ND
5	16	32
6	14	ND
7	15	ND
8	16	30
9	16	37
10	15	ND
11	12	ND
12	14	ND
13	12	ND
14	13	ND
15	16	26
16	14	ND
17	12	ND
18	15	ND
19	15	ND
20	12	ND

Tabla 5.9: Desempeño del even-*4*-parity con el método sin encapsulamiento. ND = No se alcanzó la zona factible.

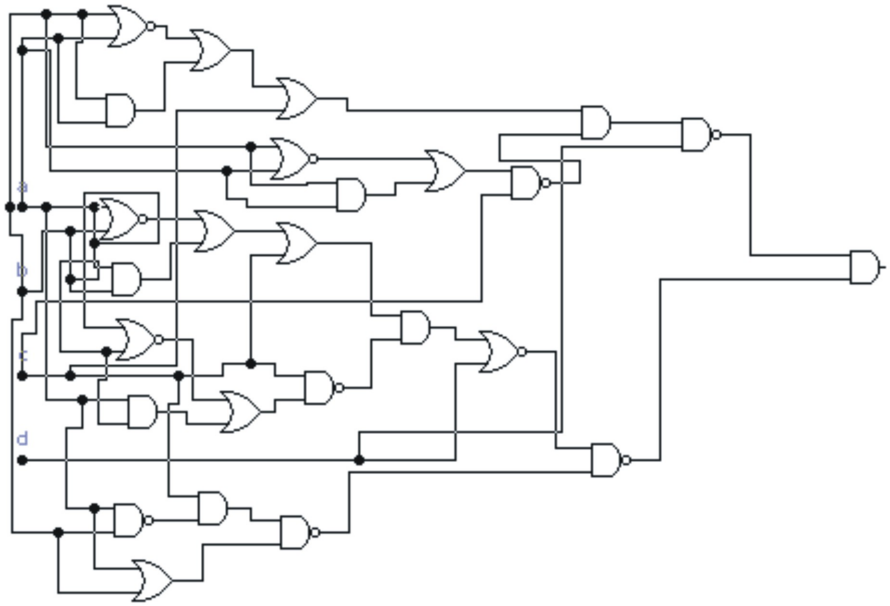

Figura 5.4: Even-4-parity con el método sin encapsulamiento.

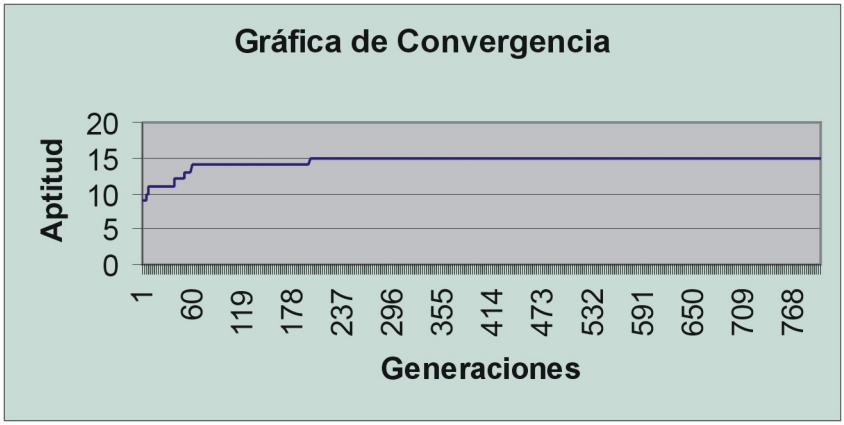

Figura 5.5: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del even-*4*-parity con el método sin encapsulamiento.

Técnica	*No. Compuertas*	*Compuertas*	*Evaluaciones*
PG Postfija	26	AND, OR, NAND, NOR	640,000
PG De Jong	15	AND, OR, NAND, NOR	5,410,550
MO De Jong	15	AND, OR, NAND, NOR	238,856

Tabla 5.10: Comparación de los resultados encontrados por la PG Postfija y la PG de De Jong para el even-*4*-parity.

Edwing D. de Jong [27] que se muestra en la Tabla 5.10.

En la Figura 5.6 se muestra el circuito correspondiente al even-*4*-parity encontrado por De Jong con su método multiobjetivo.

5.2.3 Ejemplo 3. Even-*5*-Parity

El tercer ejemplo consiste en el circuito even-*5*-parity, que consta de 5 entradas y 1 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 5.11. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la tabla 5.12.

En ninguna corrida se alcanzó la zona factible. Las estadísticas son mostradas en la Tabla 5.13 y en la Tabla 5.14 se muestra el análisis de las 20 corridas. El desempeño se muestra en la Figura 5.7. Estos resultados son comparados contra los reportados por Edwing D. de Jong [27] que se muestra en la Tabla 5.15.

En la Figura 5.8 se muestra el circuito correspondiente al even-*5*-parity

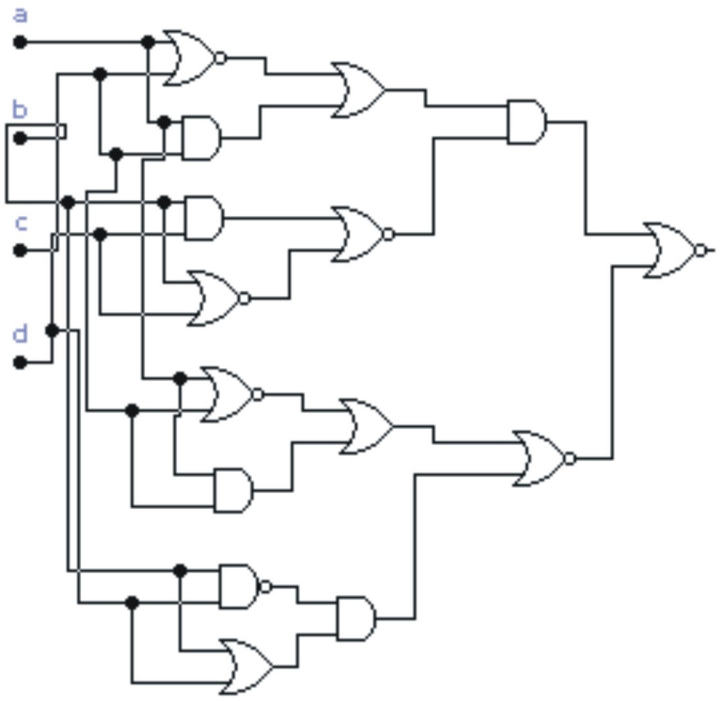

Figura 5.6: Even-*4*-parity con el método multiobjetivo encontrado por De Jong.

a	b	c	d	e	$S1$
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	1
1	0	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	0	1	0
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	1
1	1	1	1	1	0

Tabla 5.11: Tabla de verdad para el even-5-parity.

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	1,200
Número máximo de generaciones	1,600
Longitud máxima permitida	720
Zona factible	0%
Mejores resultados	ND

Tabla 5.12: Parámetros utilizados en el método sin encapsulamiento para el ejemplo even-*5*-parity.ND = No se alcanó la zona factible.

Media	*D.E.*	*Mediana*
26.3481	2.3502	26

Tabla 5.13: Estadísticas del even-*5*-parity con el método sin encapsulamiento.

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	26	ND
2	30	ND
3	25	ND
4	27	ND
5	27	ND
6	29	ND
7	26	ND
8	25	ND
9	28	ND
10	28	ND
11	26	ND
12	30	ND
13	27	ND
14	26	ND
15	30	ND
16	25	ND
17	24	ND
18	23	ND
19	21	ND
20	26	ND

Tabla 5.14: Desempeño del even-*5*-parity con el método sin encapsulamiento. ND = No se alcanzó la Zona Factible.

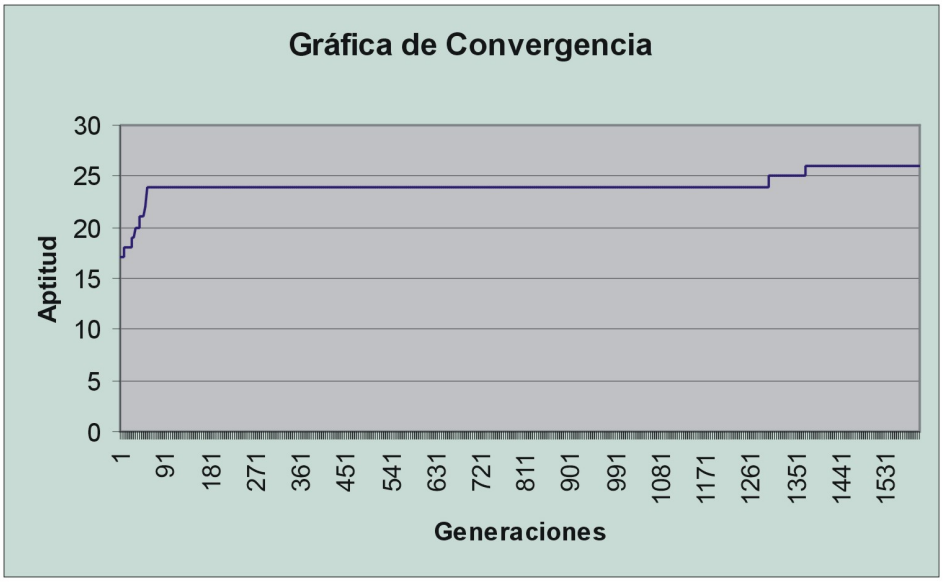

Figura 5.7: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del even-*5*-parity con el método sin encapsulamiento.

Técnica	*No. Compuertas*	*Compuertas*	*Evaluaciones*
PG Postfija	ND	AND, OR, NAND, NOR	1,920,000
PG De Jong	ND	AND, OR, NAND, NOR	1,000,000
MO De Jong	27	AND, OR, NAND, NOR	1,140,000

Tabla 5.15: Comparación de los resultados encontrados por la PG Postfija y la PG de De Jong para el even-*5*-parity. ND = No se alcanzó la zona factible.

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	240
Número máximo de generaciones	400
Longitud máxima permitida	180
Zona factible	80%
Mejores resultados	10% con 6 compuertas

Tabla 5.16: Parámetros utilizados en el método de encapsulamiento para el ejemplo even-*3*-parity.

Media	*D.E.*	*Mediana*
7.7817	0.4103	8

Tabla 5.17: Estadísticas del even-*3*-parity con el método de encapsulamiento.

encontrado por De Jong con su método multiobjetivo.

5.3 Método de Encapsulamiento.

5.3.1 Ejemplo 1. Even-*3*-Parity.

El primer ejemplo consiste en el circuito even-*3*-parity, que consta de 3 entradas y 1 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 5.1. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la tabla 5.16.

Las estadísticas son mostradas en la Tabla 5.17 y en la Tabla 5.18 se muestra el análisis de las 20 corridas. El circuito lógico de 6 compuertas es mostrado en la Figura 5.9 mientras que el desempeño se muestra en la Figura 5.10. Estos resultados son comparados contra los reportados por Edwing D. de Jong [27] que se muestra en la Tabla 5.19. En la Figura 5.3 se muestra el circuito correspondiente al even-*3*-parity encontrado por De Jong con su método multiobjetivo.

5.3.2 Ejemplo 2. Even-*4*-Parity.

El segundo ejemplo consiste en el circuito even-*4*-parity, que consta de 4 entradas y 1 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 5.6. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la Tabla 5.20.

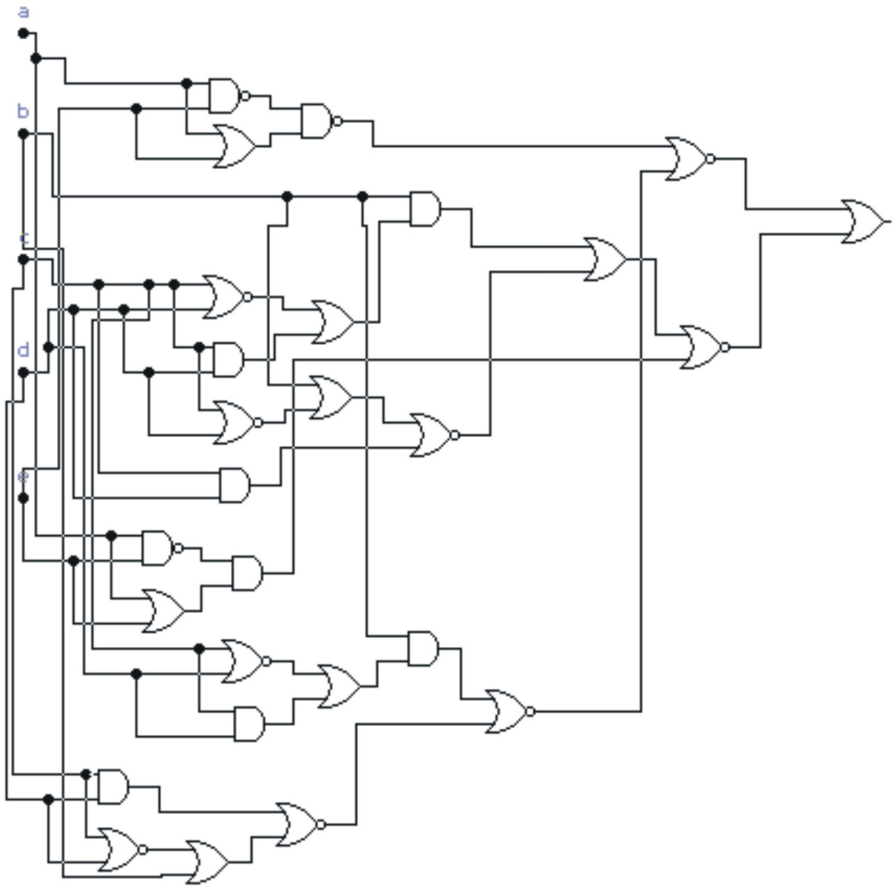

Figura 5.8: Even-*5*-parity con el método multiobjetivo encontrado por De Jong.

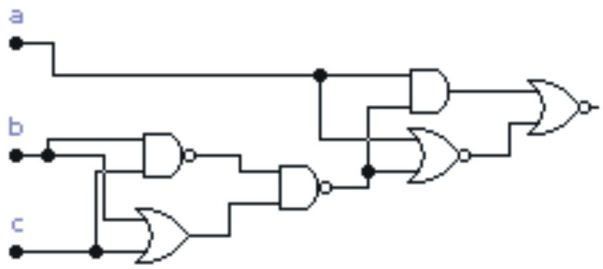

Figura 5.9: Even-*3*-parity con el método de encapsulamiento.

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	7	ND
2	8	8
3	8	10
4	8	10
5	8	10
6	7	ND
7	7	ND
8	7	ND
9	8	8
10	8	10
11	8	15
12	8	8
13	8	6
14	8	11
15	8	10
16	8	12
17	8	7
18	8	10
19	8	10
20	8	6

Tabla 5.18: Desempeño del even-*3*-parity con el método de encapsulamiento.
ND = No se alcanzó la zona factible.

Técnica	*No. Compuertas*	*Compuertas*	*Evaluaciones*
PG Postfija	6	AND, OR, NAND, NOR	96,000
PG De Jong	9	AND, OR, NAND, NOR	72,044
MO De Jong	9	AND, OR, NAND, NOR	42,965

Tabla 5.19: Comparación de los resultados encontrados por la PG Postfija y la PG de De Jong para el even-*3*-parity.

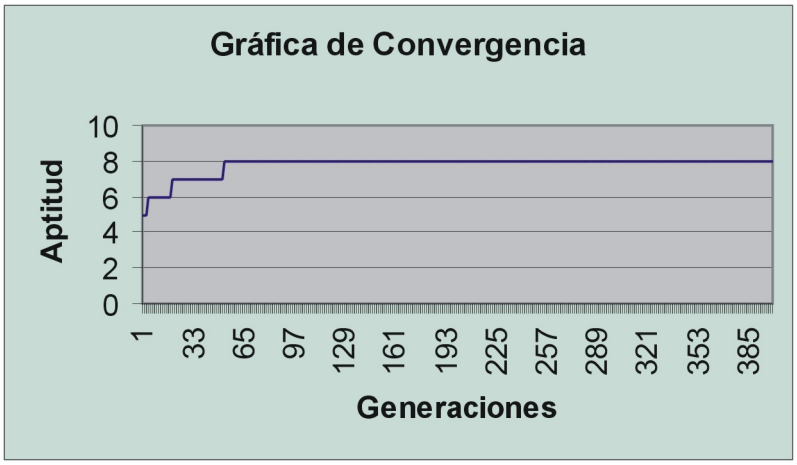

Figura 5.10: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del even-*3*-parity con el método de encapsulamiento.

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	800
Número máximo de generaciones	800
Longitud máxima permitida	380
Zona factible	55%
Mejores resultados	5% con 12 compuertas

Tabla 5.20: Parámetros utilizados en el método de encapsulamiento para el ejemplo even-*4*-parity.

Media	*D.E.*	*Mediana*
15.3290	0.8127	16

Tabla 5.21: Estadísticas del even-*4*-parity con el método de encapsulamiento.

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	16	12
2	14	ND
3	16	28
4	14	ND
5	16	34
6	16	23
7	16	23
8	16	26
9	15	ND
10	15	ND
11	14	ND
12	16	35
13	16	29
14	15	ND
15	14	ND
16	15	ND
17	16	31
18	16	24
19	15	ND
20	16	29

Tabla 5.22: Desempeño del even-*4*-parity con el método de encapsulamiento. ND = No se alcanzó la zona factible.

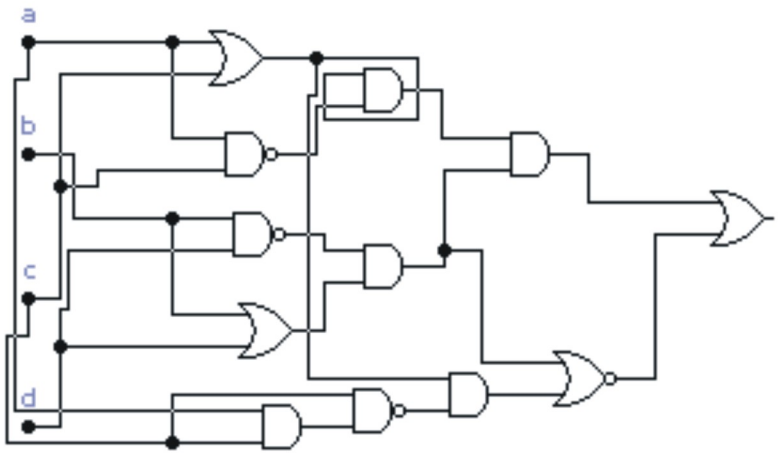

Figura 5.11: Even-*4*-parity con el método de encapsulamiento.

Técnica	*No. Compuertas*	*Compuertas*	*Evaluaciones*
PG Postfija	12	AND, OR, NAND, NOR	640,000
PG De Jong	15	AND, OR, NAND, NOR	5,410,550
MO De Jong	15	AND, OR, NAND, NOR	238,856

Tabla 5.23: Comparación de los resultados encontrados por la PG Postfija y la PG de De Jong para el even-*4*-parity.

Las estadísticas son mostradas en la Tabla 5.21 y en la Tabla 5.22 se muestra el análisis de las 20 corridas. El circuito lógico de 12 compuertas es mostrado en la Figura 5.11 mientras que el desempeño se muestra en la Figura 5.12. Estos resultados son comparados contra los reportados por Edwing D. de Jong [27] que se muestra en la Tabla 5.23. En la Figura 5.6 se muestra el circuito correspondiente al even-*4*-parity encontrado por De Jong con su método multiobjetivo.

5.3.3 Ejemplo 3. Even-*5*-Parity

El tercer ejemplo consiste en el circuito even-*5*-parity, que consta de 5 entradas y 1 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 5.11. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la Tabla 5.24.

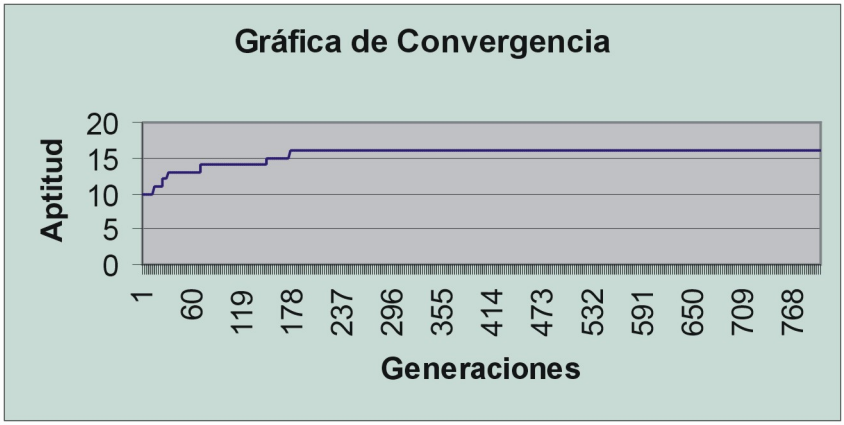

Figura 5.12: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del even-*4*-parity con el método de encapsulamiento.

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	1,200
Número máximo de generaciones	1,600
Longitud máxima permitida	720
Zona factible	10%
Mejores resultados	5% con 30 compuertas

Tabla 5.24: Parámetros utilizados en el método de encapsulamiento para el ejemplo even-*5*-parity.

Media	*D.E.*	*Mediana*
28.0829	2.5874	28

Tabla 5.25: Estadísticas del even-*5*-parity con el método de encapsulamiento.

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	32	30
2	29	ND
3	31	ND
4	25	ND
5	26	ND
6	22	ND
7	27	ND
8	28	ND
9	29	ND
10	32	44
11	30	ND
12	26	ND
13	31	ND
14	28	ND
15	25	ND
16	28	ND
17	28	ND
18	28	ND
19	28	ND
20	31	ND

Tabla 5.26: Desempeño del even-*5*-parity con el método de encapsulamiento. ND = No se alcanzó la Zona Factible.

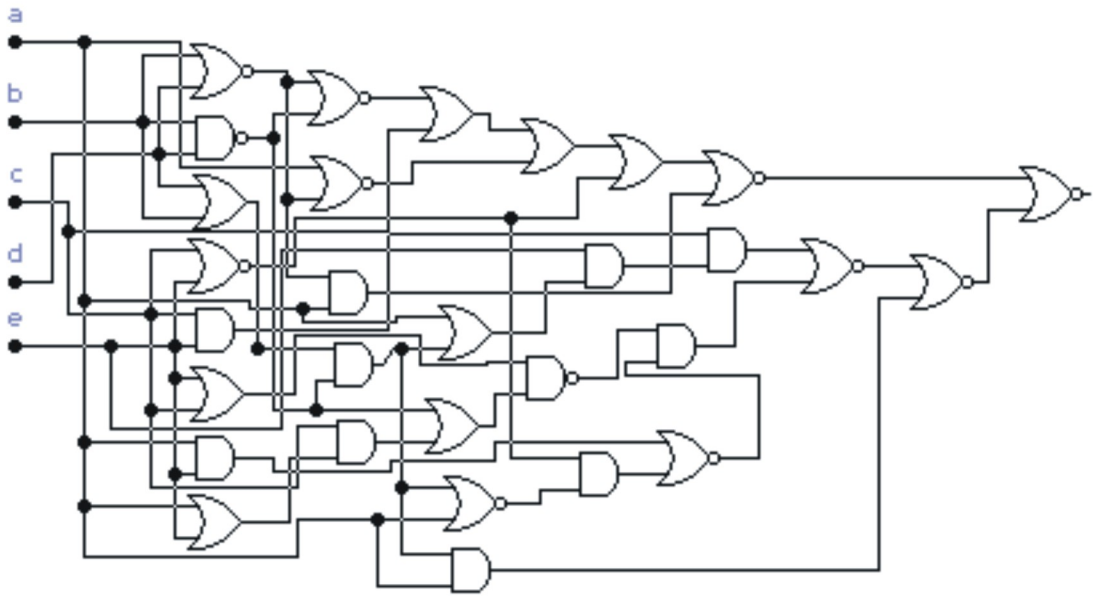

Figura 5.13: Even-*5*-parity con el método de encapsulamiento.

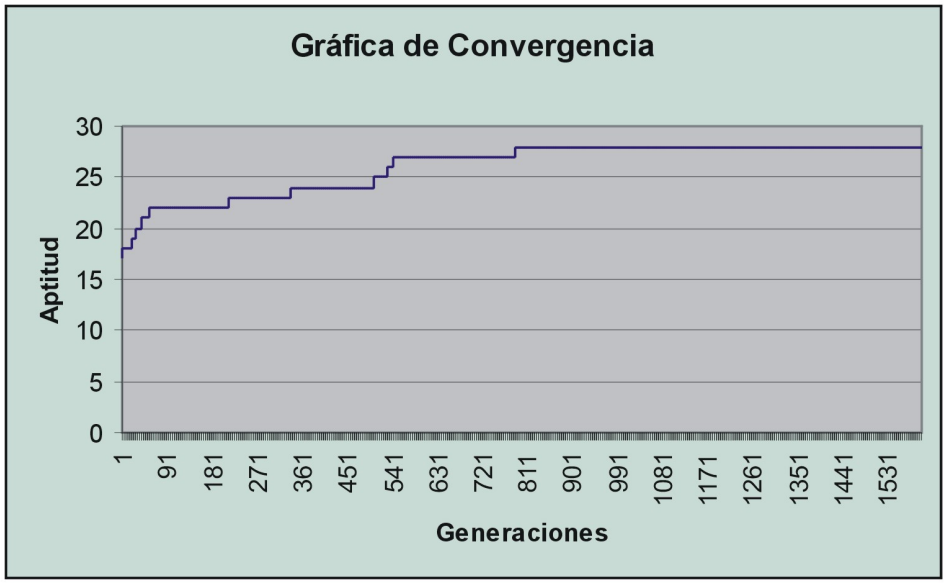

Figura 5.14: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del even-*5*-parity con el método de encapsulamiento.

Técnica	*No. Compuertas*	*Compuertas*	*Evaluaciones*
PG Postfija	30	AND, OR, NAND, NOR	1,920,000
PG De Jong	ND	AND, OR, NAND, NOR	1,000,000
MO De Jong	27	AND, OR, NAND, NOR	1,140,000

Tabla 5.27: Comparación de los resultados encontrados por la PG Postfija y la PG de De Jong para el even-*5*-parity. ND = No se alcazó la zona factible.

Las estadísticas son mostradas en la Tabla 5.25 y en la Tabla 5.26 se muestra el análisis de las 20 corridas, el desempeño se muestra en la Figura 5.14. El circuito con 30 compuertas es mostrado en la Figura 5.13. Estos resultados son comparados contra los reportados por Edwing D. de Jong [27] que se muestra en la Tabla 5.27.

En la Figura 5.8 se muestra el circuito correspondiente al even-*5*-parity encontrado por De Jong con su método multiobjetivo.

even-3-*parity.*	
20 corridas Tamaño de Población: 240 individuos. Número máximo de generaciones: 400.	
M. Sin Encapsulamiento	*M. de Encapsulamiento*
65% Zona Factible 10% Mejor Solución con 9 compuertas	80% Zona Factible 10% Mejor Solución con 6 compuertas
even-4-*parity.*	
20 corridas Tamaño de Población: 800 individuos. Número máximo de generaciones: 800.	
M. sin Encapsulamiento	*M. de Encapsulamiento*
20% Zona Factible Mejor solución con 26 compuertas	55% Zona Factible Mejor solución con 12 compuertas
even-5-*parity.*	
20 corridas Tamaño de la Población: 1200 individuos. Número máximo de generaciones: 1600.	
M. sin Encapsulamiento	*M. de Encapsulamiento*
0% Zona Factible ND	10% Zona Factible Mejor solución con 30 compuertas

Tabla 5.28: Tabla comparativa del desempeño de los métodos sin y de encapsulamiento. ND = No se alcanzó zona factible.

5.4 Análisis de Resultados.

En la Tabla 5.28 se hace una comparación de los resultados obtenidos por cada uno de los métodos propuestos. Como se observa, el método de encapsulamiento tiene mejor desempeño que el método sin encapsulamiento. En todos los circuitos, se obtienen mejores resultados con el método de encapsulamiento.

Capítulo 6

Circuitos con múltiples salidas.

6.1 Circuitos de más de una salida.

Existen diferentes técnicas evolutivas que han atacado el problema de resolver circuitos de múltiples salidas, entre las que podemos mencionar: el Algoritmo Genético Multiobjetivo [6, 7], el Sistema de Colonia de Hormigas [8, 9], el NGA [1, 4, 5] y la Programación Genética Prefija [28]. Esta última técnica reporta buenos resultados en lo que se refiere a los circuitos de una salida, no así para los circuitos de múltiples salidas, ya que este método lo que hace es resolver las salidas por separado.

Los operadores genéticos que se utilizaron en la implementación fueron:

- *Cruza.*
- *Mutación.*
- *EGL.*
- *Elitismo.*

Para la reutilización de compuertas se introdujo un símbolo terminal que hace referencia a partes del mismo individuo. Esto se explica detalladamente en el capítulo 4. El conjunto de símbolos funcionales utilizados son: *AND, OR, NOT, OR-exclusivo*; mientras que el conjunto de símbolos terminales son tantas letras como número de entradas contenga el circuito a resolverse.

6.2 Experimentos.

Los siguientes circuitos a resolverse, son circuitos de m entradas y n salidas, donde n es mayor a 1. Se seleccionaron 5 circuitos de diferente grado de complejidad para ser probados con la Programación Genética Postfija. Los resultados serán comparados con ambos Métodos propuestos en esta tesis (Método de Encapsulamiento y Método Poblacional/Autoadaptativo).

Para los primeros cuatro ejemplos se realizaron 20 corridas, mientras que para el último se realizaron 5 corridas en forma aleatoria. El tamaño de cada individuo varía según la complejidad de cada circuito, así como el número de generaciones y tamaño de la población.

En ambos métodos se utilizó un porcentaje de cruza de 70% y un porcentaje de mutación de 30%. En el Método Poblacional y Autoadaptativo, estos porcentajes varían durante la ejecución del programa.

El tamaño de la población, el número máximo de generaciones así como la longitud máxima permitida se determinaron con base en distintas pruebas que se hicieron. **Cabe aclarar que el parámetro de longitud máxima permitida determina en gran medida el desempeño que tendrá el algoritmo.** En cada uno de los ejemplos se mencionarán los valores de éstos tres parámetros. La longitud permitida varía cada cierto número de generaciones, entre más complejo sea el circuito mayor será el número de generaciones que transcurran para que la longitud aumente.

6.3 Método de Encapsulamiento.

6.3.1 Ejemplo 1. Sumador de 2 bits.

El primer ejemplo consiste en el sumador de 2 bits, que consta de 4 entradas y 3 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 6.1. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la Tabla 6.2.

Las estadísticas son mostradas en la tabla 6.3 y en la tabla 6.4 se muestra el análisis de las 20 corridas. El circuito lógico óptimo es mostrado en la figura 6.1. El desempeño es mostrado en la figura 6.2.

En la tabla 6.5 se compara el resultado generado con Programación Genética contra el de un Diseñador Humano, Algoritmo Genético Binario y el Sistema de Colonia de Hormigas [8, 9].

a	*b*	*c*	*d*	*S1*	*S2*	*S3*
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	0
1	0	0	1	1	1	0
1	0	1	0	0	0	1
1	0	1	1	1	0	1
1	1	0	0	1	1	0
1	1	0	1	0	0	1
1	1	1	0	1	0	1
1	1	1	1	0	1	1

Tabla 6.1: Tabla de verdad para el Sumador de 2 bits.

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	560
Número máximo de generaciones	700
Longitud máxima permitida	220
Zona factible	100%
Mejores resultados	10% con 7 compuertas

Tabla 6.2: Parámetros utilizados en el método de encapsulamiento para el ejemplo del sumador de 2 bits.

Media	*D.E.*	*Mediana*
48	0	48

Tabla 6.3: Estadísticas del Sumador de 2 bits con el Método de Encapsulamiento.

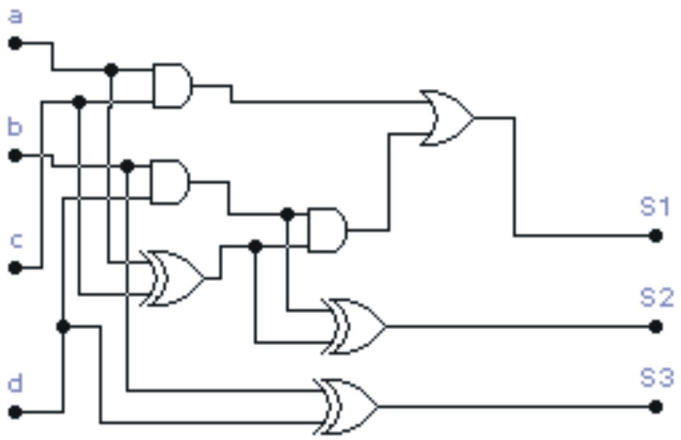

Figura 6.1: Sumador de 2 bits con el Método de Encapsulamiento.

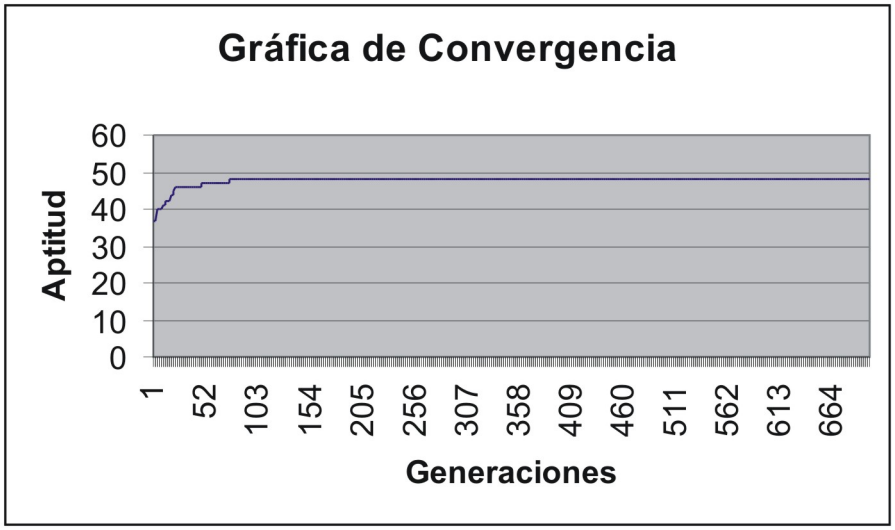

Figura 6.2: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del Sumador de 2 bits con el Método de Encapsulamiento.

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	48	7
2	48	40
3	48	12
4	48	11
5	48	7
6	48	19
7	48	10
8	48	22
9	48	9
10	48	11
11	48	14
12	48	9
13	48	10
14	48	13
15	48	22
16	48	10
17	48	12
18	48	11
19	48	19
20	48	13

Tabla 6.4: Desempeño del Sumador de 2 bits con el Método de Encapsulamiento.

PG Postfija
S1 = bd⊕ S2 = (bc)(ac⊕)⊕ S3 = (ac)((bc)(ac⊕)) + 7 Compuertas 1 OR, 3 AND, 3 XOR.
Diseñador Humano
S1 = b ⊕ d S2 = (a ⊕ c)d' + ((a ⊕ c) ⊕ b)d S3 = ac + bd(a + c) 12 Compuertas 5 AND, 3 OR, 3 XOR, 1 NOT
BGA
S1 = b ⊕ d S2 = (a ⊕ c) ⊕ bd S3 = ac + bd(a ⊕ c) 7 Compuertas 1 OR, 3 AND, 3 XOR
AS
S1 = d ⊕ b S2 = (db(a ⊕ c)) ⊕ ab S3 = db ⊕ (a ⊕ c) 7 Compuertas 3 AND, 4 XOR.

Tabla 6.5: Tabla comparativa de las mejores soluciones obtenidas en Programación Genética Postfija, Diseñador Humano, Algoritmo Genético Binario y la Colonia de Hormigas para el Sumador de 2 bits.

a	*b*	*c*	*d*	*S1*	*S2*	*S3*	*S4*
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

Tabla 6.6: Tabla de verdad para el Multiplicador de 2 bits.

6.3.2 Ejemplo 2. Multiplicador de 2 bits.

El segundo ejemplo consiste en el multiplicador de 2 bits, que consta de 4 entradas y 4 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 6.6. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la Tabla 6.7.

El circuito lógico óptimo es mostrado en la figura 6.3. El desempeño es mostrado en la figura 6.4.

En la tabla 6.10 se muestran los resultados de otras técnicas, como el de un Diseñador Humano, Algoritmo Genético Multiobjetivo, el presentado por Miller et al. [19] y el Sistema de la Colonia de Hormigas [8, 9]

6.3.3 Ejemplo 3. Circuito FDDI.

El tercer ejemplo consiste en el circuito FDDI, que consta de 4 entradas y 5 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 6.11. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la Tabla 6.12.

Las estadísticas son mostradas en la tabla 6.13 y en la tabla 6.14 se muestra el análisis de las 20 corridas. El circuito con 16 compuertas es

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	450
Número máximo de generaciones	500
Longitud máxima permitida	240
Zona factible	100%
Mejores resultados	15% con 7 compuertas

Tabla 6.7: Parámetros utilizados en el método de encapsulamiento para el ejemplo del multiplicador de dos bits.

Media	*D.E.*	*Mediana*
64	0	64

Tabla 6.8: Estadísticas del Multiplicador de 2 bits con el Método de Encapsulamiento.

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	64	11
2	64	8
3	64	10
4	64	9
5	64	10
6	64	10
7	64	11
8	64	9
9	64	12
10	64	7
11	64	8
12	64	8
13	64	7
14	64	10
15	64	10
16	64	9
17	64	11
18	64	10
19	64	10
20	64	7

Tabla 6.9: Desempeño del Multiplicador de 2 bits con el Método de Encapsulamiento.

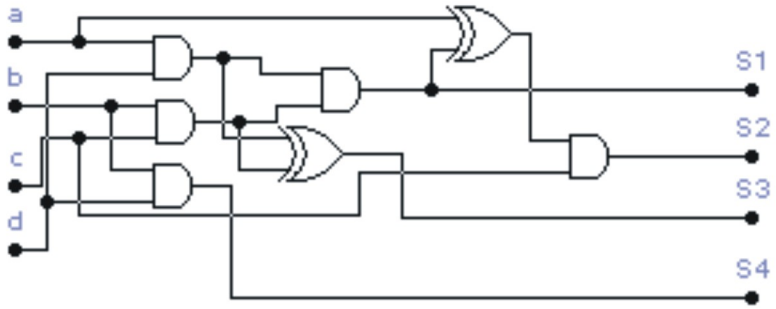

Figura 6.3: Multiplicador de 2 bits encontrado por el método de encapsulamiento.

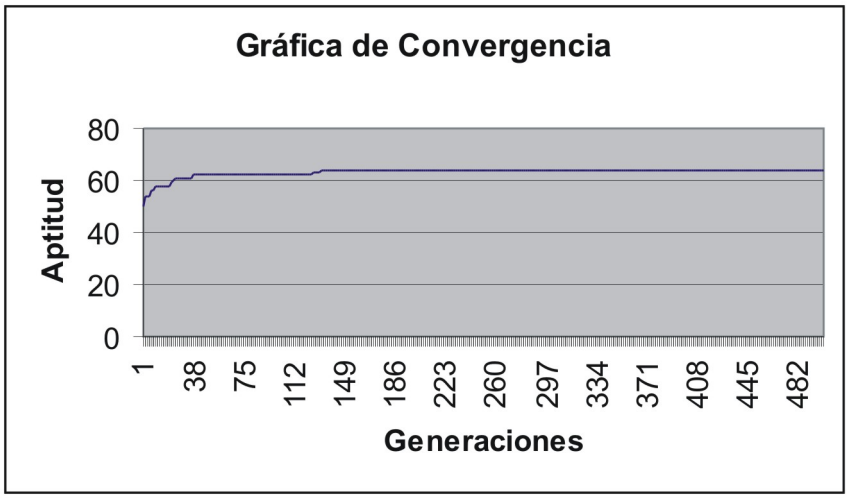

Figura 6.4: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del Multiplicador de 2 bits con el Método de Encapsulamiento.

PG Postfija
S1 = bd
S2 = (ad)(bc)⊕
S3 = (((ad)(bc))a⊕)c
S4 = (ad)(bc)
7 Compuertas
5 AND, 2 XOR.
Diseñador Humano
S1 = bd
S2 = bc ⊕ ad
S3 = ac(bd)'
S4 = bdac
8 Compuertas
6 AND, 1 XOR, 1 NOT.
MGA
S1 = bd
S2 = bc ⊕ ad
S3 = ac ⊕ (bdac)
S4 = bdac
7 Compuertas
5 AND, 2 XOR.
Miller et al.
S1 = bd
S2 = ad ⊕ bc
S3 = (bd)'ac
S4 = (ad ⊕ bc)' (ad)
9 Compuertas
6 AND, 1 XOR, 2 NOT.
AS
S1 = bd
S2 = ad ⊕ cb
S3 = ((ac)d ⊕ (ac)
S4 = acd
7 Compuertas
5 AND, 2 XOR.

Tabla 6.10: Tabla comparativa de las mejores soluciones obtenidas en Programación Genética Postfija, Diseñador Humano, Algoritmo Genético Multiobjetivo, Miller et al. y el Sistema de la Colonia de Hormigas para el ejemplo del Multiplicador de 2 bits.

a	*b*	*c*	*d*	*S1*	*S2*	*S3*	*S4*	*S5*
0	0	0	0	1	1	1	1	0
0	0	0	1	0	1	0	0	1
0	0	1	0	1	0	1	0	0
0	0	1	1	1	0	1	0	1
0	1	0	0	0	1	0	1	0
0	1	0	1	0	1	0	1	1
0	1	1	0	0	1	1	1	0
0	1	1	1	0	1	1	1	1
1	0	0	0	1	0	0	1	0
1	0	0	1	1	0	0	1	1
1	0	1	0	1	0	1	1	0
1	0	1	1	1	0	1	1	1
1	1	0	0	1	1	0	1	0
1	1	0	1	1	1	0	1	1
1	1	1	0	1	1	1	0	0
1	1	1	1	1	1	1	0	1

Tabla 6.11: Tabla de verdad para el circuito de FDDI.

mostrado en la figura 6.5. En la figura 6.6 se muestra la gráfica de convergencia.

6.3.4 Ejemplo 4. Circuito de Katz.

El cuarto ejemplo consiste en el circuito de Katz, que consta de 4 entradas y 3 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 6.15. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la Tabla 6.16.

Las estadísticas son mostradas en la tabla 6.17 y en la tabla 6.18 se muestra el análisis de las 20 corridas. El circuito con 19 compuertas es mostrado en la figura 6.7. En la figura 6.8 se muestra la gráfica de convergencia.

En la tabla 6.19 se compara el resultado obtenido con la Programación Genética contra el de un Diseñador Humano 1 que utiliza mapas de Karnaugh y álgrebra booleana, el Diseñador Humano 2 utiliza el método de Quine-McCluskey, el Algoritmo Genético Multiobjetico y el Sistema de Colonia de Hormigas [8, 9].

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	600
Número máximo de generaciones	2,000
Longitud máxima permitida	260
Zona factible	100%
Mejores resultados	5% con 16 compuertas

Tabla 6.12: Parámetros utilizados en el método de encapsulamiento para el ejemplo del circuito FDDI.

Media	*D.E.*	*Mediana*
80	0	80

Tabla 6.13: Estadísticas del FDDI con el Método de Encapsulamiento.

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	80	19
2	80	22
3	80	24
4	80	19
5	80	16
6	80	19
7	80	20
8	80	19
9	80	17
10	80	18
11	80	19
12	80	20
13	80	21
14	80	19
15	80	19
16	80	20
17	80	20
18	80	19
19	80	20
20	80	21

Tabla 6.14: Desempeño del FDDI con el Método de Encapsulamiento.

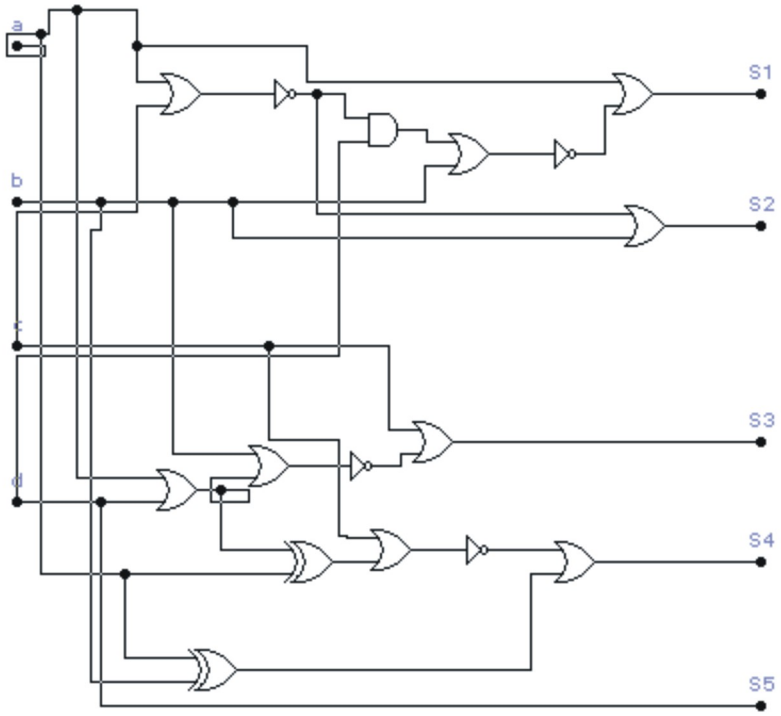

Figura 6.5: Circuito FDDI encontrado con el Método de Encapsulamiento.

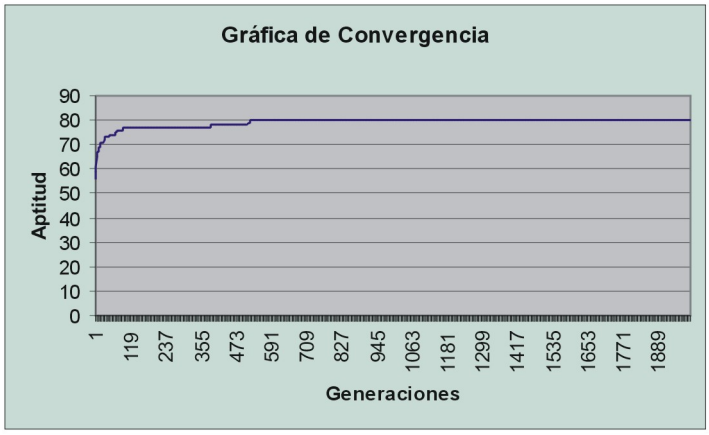

Figura 6.6: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del circuito FDDI con el Método de Encapsulamiento.

a	*b*	*c*	*d*	*S1*	*S2*	*S3*
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	1	0	0

Tabla 6.15: Tabla de verdad para el circuito de Katz.

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	880
Número máximo de generaciones	4,000
Longitud máxima permitida	320
Zona factible	30%
Mejores resultados	5% con 19 compuertas

Tabla 6.16: Parámetros utilizados en el método de encapsulamiento para el ejemplo del circuito de Katz.

Media	*D.E.*	*Mediana*
47.29	0.47016	47

Tabla 6.17: Estadísticas del circuito de Katz con el Método de Encapsulamiento.

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	47	ND
2	48	23
3	48	32
4	47	ND
5	47	ND
6	47	ND
7	47	ND
8	48	19
9	47	ND
10	48	26
11	47	ND
12	47	ND
13	47	ND
14	48	20
15	47	ND
16	47	ND
17	47	ND
18	47	ND
19	48	25
20	47	ND

Tabla 6.18: Desempeño del circuito de Katz con el Método de Encapsulamiento.
ND = Significa que no se llegó a la zona factible.

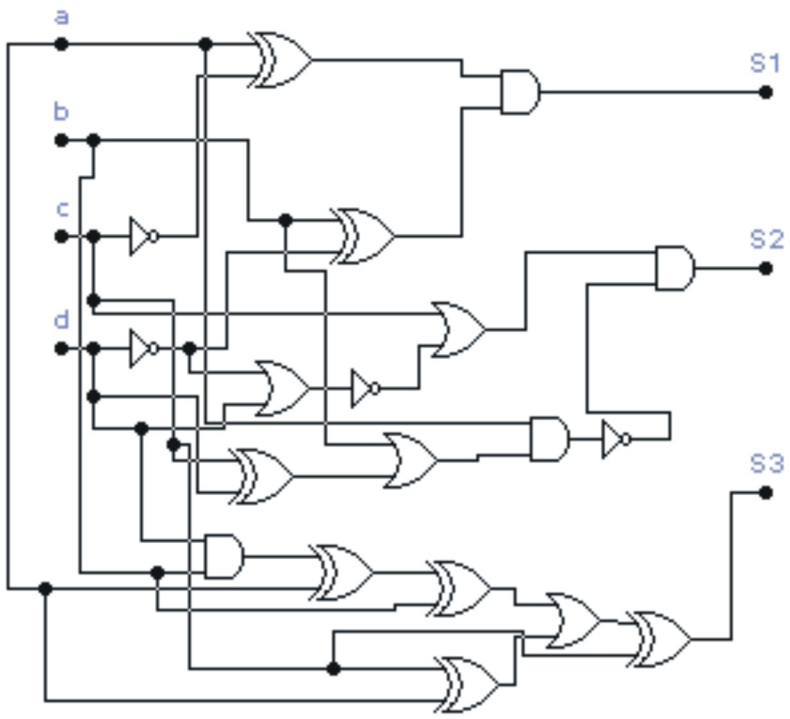

Figura 6.7: Circuito de Katz con el Método de Encapsulamiento.

PG Postfija
S1 = ((c'a⊕)(d'b⊕))
S2 = ((((d'd+)')c+)((((cd⊕)b+)a)'))
S3 = ((((db)a⊕)b⊕)(ac⊕)or)c⊕
19 Compuertas
4 AND, 4 OR, 7 XOR, 4 NOT
Diseñador Humano 1
S1 = (a ⊕ c)' (b ⊕ d)'
S2 = b'd (a' + c) + a'c
S3 = bd' (a + c') + ac'
19 Compuertas
7 AND, 4 OR, 2 XOR, 6 NOT
Diseñador Humano 2
S1 = (a ⊕ c)' (b ⊕ d)'
S2 = a'c + (a ⊕ c)' (b'd)
S3 = (S1 + S2)'
13 Compuertas
4 AND, 2 OR, 2 XOR, 5 NOT
MGA
S1 = ((b ⊕ d) + (a ⊕ c))'
S2 = S3 + ((b ⊕ d) + (a ⊕ c))
S3 = ((b ⊕ d) + (a ⊕ c)) (((a ⊕ c) + (a ⊕ b)) ⊕ c)
9 Compuertas
2 AND, 3 OR, 3 XOR, 2 NOT
AS
S1 = ((a ⊕ c) + (b ⊕ d))'
S2 = ((ca) ⊕ a) ⊕ ((s ⊕ c) + (b ⊕ d))((a ⊕ c) + d)
S3 = S1 ⊕ S2
11 Compuertas
2 AND, 2 OR, 5 XOR, 2 NOT

Tabla 6.19: Tabla comparativa de las mejores soluciones obtenidas en Programación Genética Postfija, Diseñador Humano, Algoritmo Genético Multiobjetivo y el Sistema de Colonia de Hormigas para el circuito de Katz.

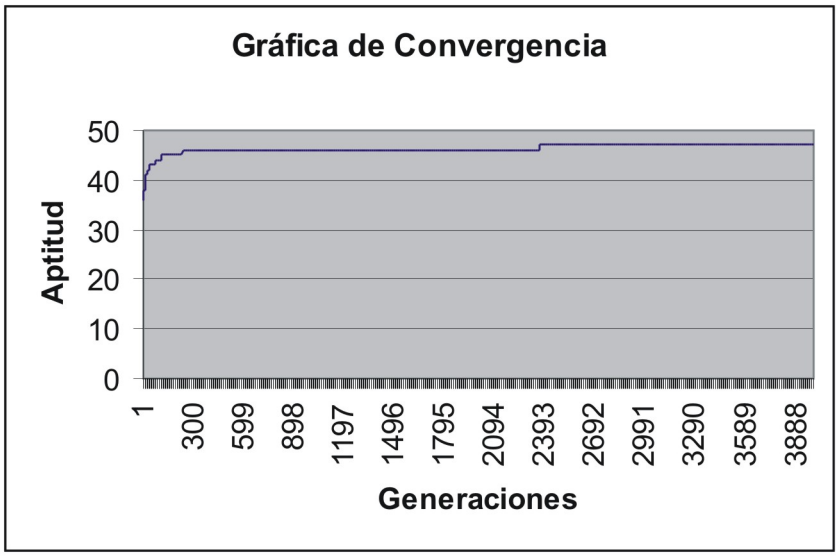

Figura 6.8: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del circuito de Katz con el Método de Encapsulamiento.

6.3.5 Ejemplo 5. Multiplicador de 3 bits.

El quinto ejemplo consiste en el multiplicador de 3 bits, que consta de 6 entradas y 5 salidas. La tabla de verdad de dicho circuito se muestra en la Tabla 6.20. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la Tabla 6.21.

Las estadísticas son mostradas en la tabla 6.22 y en la tabla 6.23 se muestra el análisis de las 5 corridas.

6.4 Método Poblacional y Autoadaptativo.

6.4.1 Ejemplo 1. Sumador de 2 bits.

El primer ejemplo consiste en el sumador de 2 bits, que consta de 4 entradas y 3 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 6.1. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la Tabla 6.24. Para el parámetro de longitud máxima permitida se empieza con una longitud de 80 alelos y, si después de 20 generaciones no aumenta la aptitud entonces dicha longitud crece 20 alelos más. Si se llega a la longitud máxima permitida entonces

a	*b*	*c*	*d*	*e*	*f*	*S1*	*S2*	*S3*	*S4*	*S5*	*S6*
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0	0	0	1
0	0	1	0	1	0	0	0	0	0	1	0
0	0	1	0	1	1	0	0	0	0	1	1
0	0	1	1	0	0	0	0	0	1	0	0
0	0	1	1	0	1	0	0	0	1	0	1
0	0	1	1	1	0	0	0	0	1	1	0
0	0	1	1	1	1	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	1	0
0	1	0	0	1	0	0	0	0	1	0	0
0	1	0	0	1	1	0	0	0	1	1	0
0	1	0	1	0	0	0	0	1	0	0	0
0	1	0	1	0	1	0	0	1	0	1	0
0	1	0	1	1	0	0	0	1	1	0	0
0	1	0	1	1	1	0	0	1	1	1	0
0	1	1	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0	1	1
0	1	1	0	1	0	0	0	0	1	1	0
0	1	1	0	1	1	0	0	1	0	0	1
0	1	1	1	0	0	0	0	1	1	0	0
0	1	1	1	0	1	0	0	1	1	1	1
0	1	1	1	1	0	0	1	0	0	1	0
0	1	1	1	1	1	0	1	0	1	0	1
1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	1	0	0
1	0	0	0	1	0	0	0	1	0	0	0
1	0	0	0	1	1	0	0	1	1	0	0
1	0	0	1	0	0	0	1	0	0	0	0
1	0	0	1	0	1	0	1	0	1	0	0
1	0	0	1	1	0	0	1	1	0	0	0
1	0	0	1	1	1	0	1	1	1	0	0
1	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	0	1	0	1
1	0	1	0	1	0	0	0	1	0	1	0
1	0	1	0	1	1	0	0	1	1	1	1
1	0	1	1	0	0	0	1	0	1	0	0
1	0	1	1	0	1	0	1	1	0	0	1
1	0	1	1	1	0	0	1	1	1	1	0
1	0	1	1	1	1	1	0	0	0	1	1
1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	1	1	0
1	1	0	0	1	0	0	0	1	1	0	0
1	1	0	0	1	1	0	1	0	0	1	0
1	1	0	1	0	0	0	1	1	0	0	0
1	1	0	1	0	1	0	1	1	1	1	0
1	1	0	1	1	0	1	0	0	1	0	0
1	1	0	1	1	1	1	0	1	0	1	0
1	1	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	1	1	1
1	1	1	0	1	0	0	0	1	1	1	0
1	1	1	0	1	1	0	1	0	1	0	1
1	1	1	1	0	0	0	1	1	1	0	0
1	1	1	1	0	1	1	0	0	0	1	1
1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	0	0	0	1

Tabla 6.20: Tabla de verdad para el Multiplicador de 3 bits.

Descripción	*Valores*
Número de corridas	5
Tamaño de la población	1,800
Número máximo de generaciones	13,000
Longitud máxima permitida	2,400
Zona factible	0%
Mejores resultados	ND

Tabla 6.21: Parámetros utilizados en el método de encapsulamiento para el ejemplo del multiplicador de 3 bits.ND=No se alcanzó la zona factible.

Media	*D.E.*	*Mediana*
382.4	0.8944	383

Tabla 6.22: Estadísticas del Multiplicador de 3 bits con el Método de encapsulamiento.

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	381	ND
2	383	ND
3	382	ND
4	383	ND
5	383	ND

Tabla 6.23: Desempeño del Multiplicador de 3 bits con el Método de Encapsulamiento.
ND = Significa que no se llegó a la zona factible.

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	560
Número máximo de generaciones	700
Longitud máxima permitida	220
Zona factible	100%
Mejores resultados	25% con 7 compuertas

Tabla 6.24: Parámetros utilizados en el método poblacional y autoadaptativo para el ejemplo del sumador de 2 bits.

Media	*D.E.*	*Mediana*
48	0	48

Tabla 6.25: Estadísticas del Sumador de 2 bits por el Método Poblacional y Autoadaptativo.

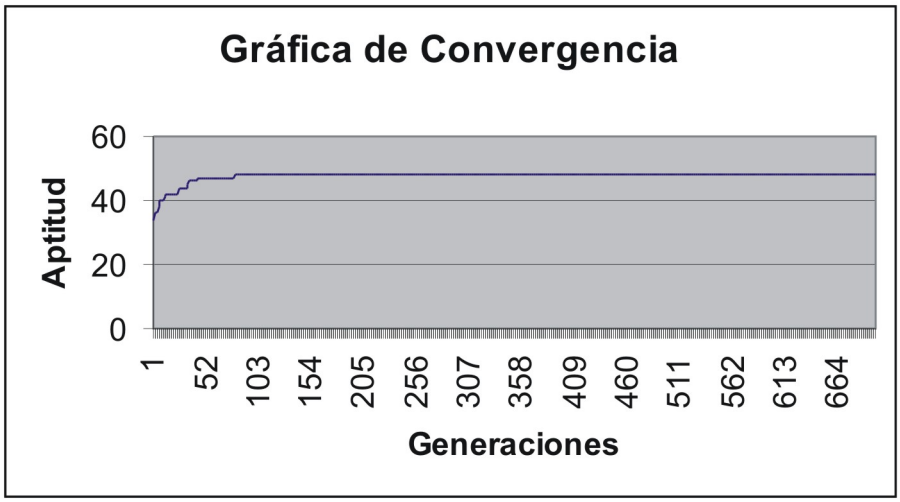

Figura 6.9: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del sumador de 2 bits con el Método Poblacional y Autoadaptativo.

aumentará el porcentaje de mutación en 5% más, si dicho porcentaje llega a 40% entonces aumentará el porcentaje de cruza en 5% más, si dicho porcentaje llega al 100% entonces se reinician dichos porcentajes en 30% y 70%, respectivamente. Las estadísticas son mostradas en la tabla 6.25 y en la tabla 6.26 se muestra el análisis de las 20 corridas. El desempeño es mostrado en la figura 6.9.

En la tabla 6.5 se compara el resultado generado con Programación Genética Postfija contra el de un Diseñador Humano, Algoritmos Genéticos y el Sistema de Colonia de Hormigas. En la figura 6.10 se muestra el circuito óptimo encontrado por el Método Poblacional y Autoadaptativo.

6.4.2 Ejemplo 2. Multiplicador de 2 bits.

El segundo ejemplo consiste en el multiplicador de 2 bits, que consta de 4 entradas y 4 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 6.6. Los parámetros utilizados para este ejemplo así como

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	48	11
2	48	7
3	48	7
4	48	9
5	48	10
6	48	10
7	48	12
8	48	11
9	48	11
10	48	11
11	48	8
12	48	8
13	48	7
14	48	10
15	48	10
16	48	7
17	48	7
18	48	9
19	48	9
20	48	9

Tabla 6.26: Desempeño del Sumador de 2 bits por el Método Poblacional y Autoadaptativo.

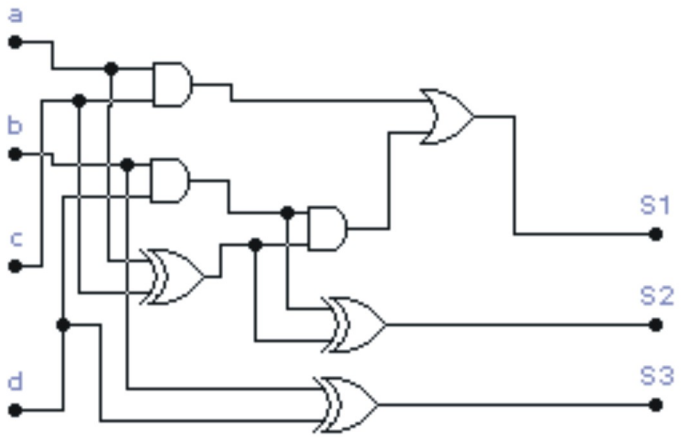

Figura 6.10: Sumador de 2 bits con el Método Poblacional y Autoadapatativo.

los resultados encontrados por este método son mostrados en la Tabla 6.27. Para el parámetro de longitud máxima permitida se empieza con una longitud de 100 alelos, si después de 20 generaciones no aumenta la aptitud entonces dicha longitud crece 20 alelos más. Si se llega a la longitud máxima permitida entonces aumentará el porcentaje de mutación en 5% más, si dicho porcentaje llega a 40% entonces aumentará el porcentaje de cruza en 5% más, si dicho porcentaje llega al 100% entonces se reinician dichos porcentajes en 30% y 70%, respectivamente. Las estadísticas son mostradas en la tabla 6.28 y en la tabla 6.29 se muestra el análisis de las 20

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	450
Número máximo de generaciones	500
Longitud máxima permitida	240
Zona factible	100%
Mejores resultados	25% con 7 compuertas

Tabla 6.27: Parámetros utilizados en el método poblacional y autoadaptativo para el ejemplo del multiplicador de 2 bits.

Media	*D.E.*	*Mediana*
64	0	64

Tabla 6.28: Estadísticas del Multiplicador de 2 bits con el Método Poblacional y Autoadaptativo.

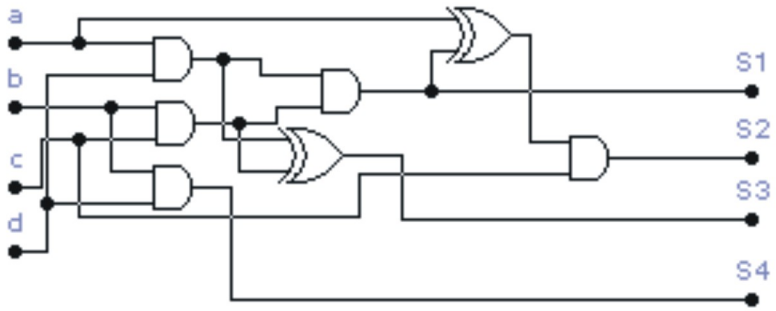

Figura 6.11: Multiplicador de 2 bits con el Método Poblacional y Autoadaptativo.

corridas. El desempeño es mostrado en la figura 6.12 y en la figura 6.11 se muestra el circuito encontrado por este método.

En la tabla 6.10 se muestra un cuadro comparativo de los mejores resultados obtenidos por otras técnicas como lo son el de un diseñador humano, uno propuesto por Miller et al., el Algoritmo Genético Multiobjetivo y el Sistema de Colonia de Hormigas.

6.4.3 Ejemplo 3. Circuito FDDI.

El tercer ejemplo consiste en el circuito FDDI, que consta de 4 entradas y 5 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 6.11. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la Tabla 6.30.

Para el parámetro de máxima longitud permitida se empieza con una longitud de 100 alelos, si después 20 generaciones no aumenta la aptitud entonces dicha longitud crece 20 alelos más. Si se llega a la longitud máxima permitida entonces aumentará el porcentaje de mutación en 5% más, si dicho porcentaje llega a 40% entonces aumentará el porcentaje de cruza en 5% más, si dicho porcentaje llega al 100% entonces se reinician dichos

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	64	7
2	64	11
3	64	10
4	64	7
5	64	11
6	64	7
7	64	9
8	64	9
9	64	10
10	64	11
11	64	8
12	64	7
13	64	10
14	64	9
15	64	9
16	64	7
17	64	10
18	64	9
19	64	11
20	64	9

Tabla 6.29: Desempeño del Multiplicador de 2 bits con el Método Poblacional y Autoadaptativo.

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	600
Número máximo de generaciones	2,000
Longitud máxima permitida	260
Zona factible	100%
Mejores resultados	55% con 15 compuertas

Tabla 6.30: Parámetros utilizados en el método poblacional y autoadaptativo para el ejemplo del circuito FDDI.

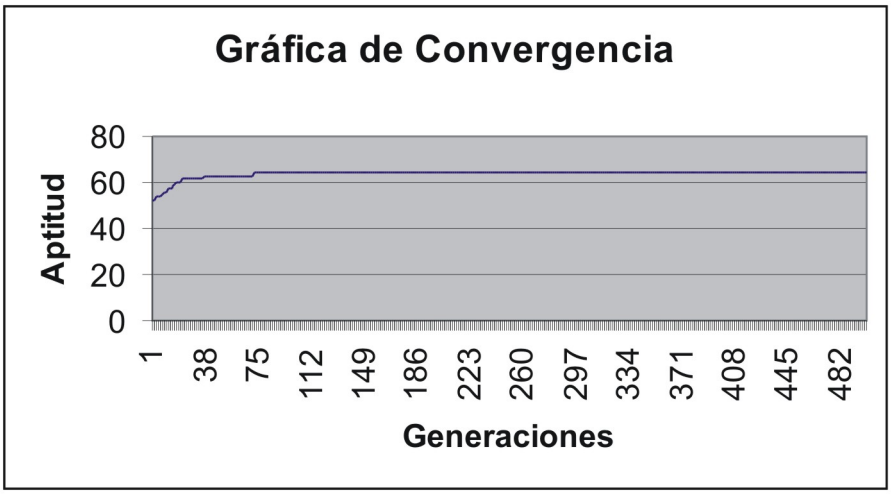

Figura 6.12: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del Multiplicador de 2 bits con el Método Poblacional y Autoadaptativo.

Media	*D.E.*	*Mediana*
80	0	80

Tabla 6.31: Estadísticas del FDDI con el Método Poblacional y Autoadaptativo.

porcentajes en 30% y 70%, respectivamente. Las estadísticas son mostradas en la tabla 6.31 y en la tabla 6.32 se muestra el análisis de las 20 corridas. El circuito con 15 compuertas es mostrado en la figura 6.13. En la figura 6.14 se muestra la gráfica de convergencia.

6.4.4 Ejemplo 4. Circuito de Katz.

El cuarto ejemplo consiste en el circuito de Katz, que consta de 4 entradas y 3 salida. La tabla de verdad de dicho circuito se muestra en la Tabla 6.15. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la Tabla 6.33. Para el parámetro de máxima longitud permitida se empieza con una longitud de 120 alelos, si después de 30 generaciones no aumenta la aptitud entonces dicha longitud crece 20 alelos más, si se llega a la longitud máxima permitida entonces aumentará el porcentaje de mutación en 5% más, si dicho porcentaje llega

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	80	19
2	80	20
3	80	19
4	80	18
5	80	17
6	80	15
7	80	21
8	80	20
9	80	21
10	80	19
11	80	17
12	80	20
13	80	21
14	80	18
15	80	22
16	80	19
17	80	19
18	80	20
19	80	21
20	80	18

Tabla 6.32: Desempeño del FDDI con el Método Poblacional y Autoadaptativo.

Descripción	*Valores*
Número de corridas	20
Tamaño de la población	880
Número máximo de generaciones	4,000
Longitud máxima permitida	320
Zona factible	75%
Mejores resultados	5% con 10 compuertas

Tabla 6.33: Parámetros utilizados en el método poblacional y autoadaptativo para el ejemplo del circuito de Katz.

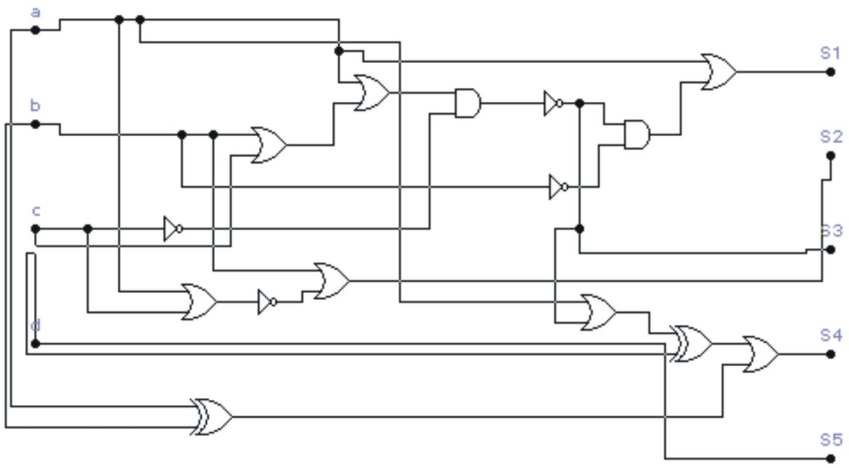

Figura 6.13: Circuito FDDI encontrado con el Método Poblacional y Autoadaptativo.

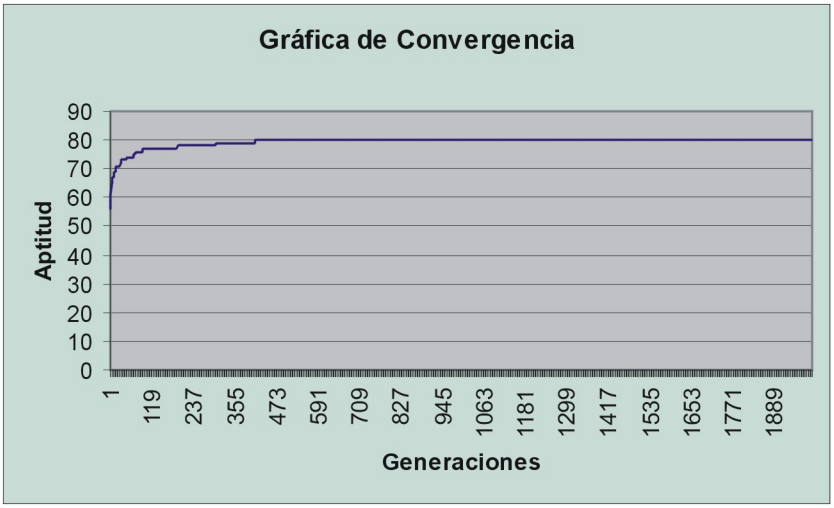

Figura 6.14: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del circuito FDDI con el Método Poblacional y Autoadaptativo.

Media	*D.E.*	*Mediana*
47.748	0.44426	48

Tabla 6.34: Estadísticas del circuito de Katz con el Método Poblacional y de Encapsulamiento.

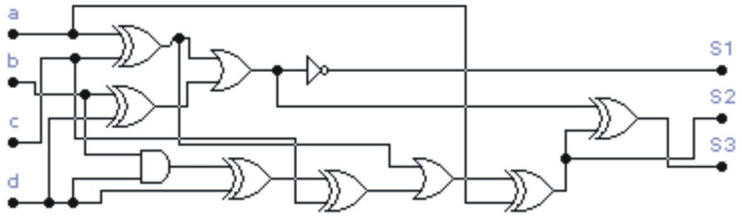

Figura 6.15: Circuito de Katz con el Método Poblacional y Autoadaptativo.

a 40% entonces aumentará el porcentaje de cruza en 5% más, si dicho porcentaje llega al 100% entonces se reinician dichos porcentajes en 30% y 70%, respectivamente. Las estadísticas son mostradas en la tabla 6.34 y en la tabla 6.35 se muestra el análisis de las 20 corridas. El circuito con 10 compuertas es mostrado en la figura 6.15. En la figura 6.16 se muestra la gráfica de convergencia.

En la tabla 6.36 se compara el resultado obtenido con la Programación Genética Postfija contra el de un Diseñador Humano 1 que utiliza mapas de Karnaugh y álgrebra booleana, el Diseñador Humano 2 utiliza el método de Quine-McCluskey, el Algoritmo Genético Multiobjetvio y el Sistema de Colonia de Hormigas.

6.4.5 Ejemplo 5. Multiplicador de 3 bits.

El quinto ejemplo consiste en el multiplicador de 3 bits, que consta de 6 entradas y 6 salidas. La tabla de verdad de dicho circuito se muestra en la Tabla 6.20. Los parámetros utilizados para este ejemplo así como los resultados encontrados por este método son mostrados en la Tabla 6.37. Para el parámetro de máxima longitud permitida se empieza con una longitud de 280 alelos, si después de 80 generaciones no aumenta la aptitud entonces dicha longitud crece 20 alelos más, si se llega a la longitud máxima permitida entonces aumentará el porcentaje de mutación en 5% más, si dicho porcentaje llega a 40% entonces aumentará el porcentaje de cruza en 5% más, si dicho porcentaje llega al 100% entonces se reinician

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	47	ND
2	48	21
3	48	14
4	48	18
5	48	12
6	47	ND
7	48	15
8	48	14
9	48	15
10	48	17
11	48	20
12	47	ND
13	48	13
14	47	ND
15	48	12
16	48	16
17	48	10
18	48	15
19	48	19
20	47	ND

Tabla 6.35: Desempeño del circuito de Katz con el Método Poblacional y Autoadaptativo. No se llegó a la Zona Factible.

PG Postfija
S1 = (P1)'
S2 = bdd ⊕ c ⊕ P2 + a ⊕
S3 = S2 P1 ⊕
Donde P1 = ac ⊕ bd ⊕ +
Donde P2 = a c ⊕
10 compuertas
1 AND, 2 OR, 6 XOR, 1 NOT
Diseñador Humano 1
S1 = (a ⊕ c)' (b ⊕ d)'
S2 = b'd (a' + c) + a'c
S3 = bd' (a + c') + ac'
19 compuertas
7 AND, 4 OR, 2 XOR, 6 NOT
Diseñador Humano 2
S1 = (a ⊕ c)' (b ⊕ d)'
S2 = a'c + (a ⊕ c)' (b'd)
S3 = (S1 + S2)'
13 compuertas
4 AND, 2 OR, 2 XOR, 5 NOT
MGA
S1 = ((b ⊕ d) + (a ⊕ c))'
S2 = S3 + ((b ⊕ d) + (a ⊕ c))
S3 = ((b ⊕ d) + (a ⊕ c)) (((a ⊕ c) + (a ⊕ b)) ⊕ c)
9 compuertas
2 AND, 3 OR, 3 XOR, 2 NOT
AS
S1 = ((a ⊕ c) + (b ⊕ d))'
S2 = ((ca) ⊕ a) ⊕ ((s ⊕ c) + (b ⊕ d))((a ⊕ c) + d)
S3 = S1 ⊕ S2
11 Compuertas
2 AND, 2 OR, 5 XOR, 2 NOT

Tabla 6.36: Tabla comparativa de las mejores soluciones obtenidas en PG Postfija Poblacional, Diseñador Humano 1, Diseñador Humano 2, Algoritmo Genético Multiobjetivo y el Sistema de Colonia de Hormigas para el circuito de Katz.

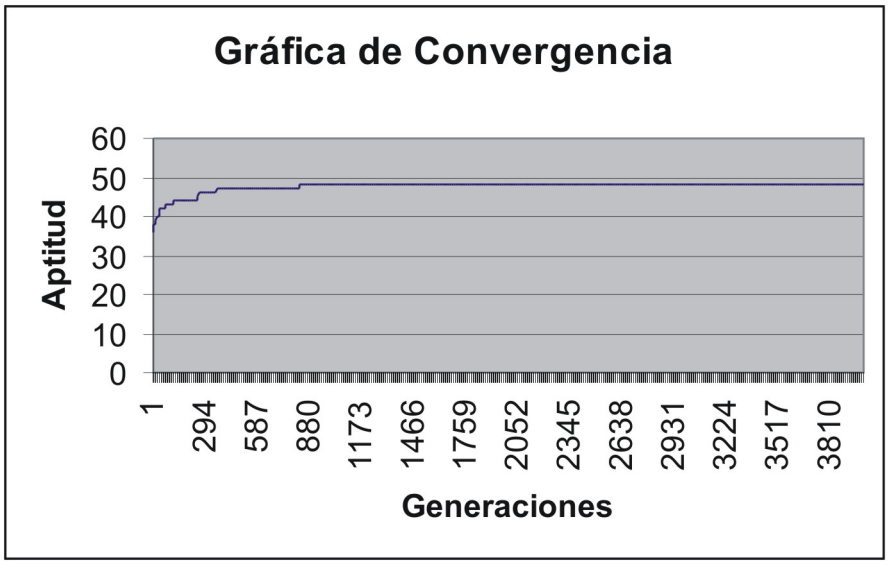

Figura 6.16: Gráfica de convergencia de la corrida ubicada en la mediana del ejemplo del circuito de Katz con el Método Poblacional y Autoadaptativo.

Descripción	*Valores*
Número de corridas	5
Tamaño de la población	1,800
Número máximo de generaciones	13,000
Longitud máxima permitida	2,400
Zona factible	20%
Mejores resultados	5% con 148 compuertas

Tabla 6.37: Parámetros utilizados en el método poblacional y autoadaptativo para el ejemplo del multiplicador de 3 bits.

Media	*D.E.*	*Mediana*
383.2	0.4472	383

Tabla 6.38: Estadísticas del Multiplicador de 3 bits con el Método Poblacional y autoadaptativo.

No. de Corrida	*Aptitud*	*No. de Compuertas*
1	383	ND
2	383	ND
3	383	ND
4	384	148
5	383	ND

Tabla 6.39: Desempeño del multiplicador de 3 bits con el Método Poblacional y Autoadaptativo.
ND = Significa que no se llegó a la zona factible.

dichos porcentajes en 30% y 70%, respectivamente. Las estadísticas son mostradas en la tabla 6.38 y en la tabla 6.39 se muestra el análisis de las 5 corridas.

En la tabla 6.40 se compara el resultado obtenido con la Programación Genética contra el de un Diseñador Humano, que utiliza mapas de Karnaugh.

6.5 Análisis de Resultados.

En la tabla 6.41 se hace una comparación de los resultados obtenidos por cada uno de los métodos propuestos. Como se observa, el método poblacional y autoadaptativo tienen mejor desempeño que el método de encapsulamiento. En todos los circuitos, se obtienen mejores resultados con el método poblacional y autoadaptativo; esta mejora se observa sobre todo en circuitos "complejos" como el *circuito de Katz* de 4 entradas y 3 salidas así como en el *circuito del multiplicador de 3 bits* de 6 entradas y 6 salidas.

No. de Compuertas con PG	*No. de Compuertas con el DH*
148	249

Tabla 6.40: Resultados encontrados por la Programación Genética Postfija y el Diseñador Humano utilizando Mapas de Karnaugh.

En lo que respecta al circuito de Katz, con el método de encapsulamiento se llega 30% de las veces a la zona factible, mientras que con el método poblacional y autoadaptativo se llega el 75% de las veces a la zona factible. En lo que respecta al circuito del multiplicador de 3 bits se observa cómo el Método de Encapsulamiento no logra alcanzar la zona factible en ninguna de las 5 corridas, mientras que el Método Poblacional y Autoadaptativo logra llegar a la zona factible en 1 de las 5 corridas.

Sumador de 2 bits.	
20 corridas Tamaño de Población: 560 individuos. Número máximo de generaciones: 700.	
M. Encapsulamiento	*M. Poblacional*
100% Zona Factible 10% Sol. óptimas	100% Zona Factible 25% Sol. óptimas
Multiplicador de 2 bits.	
20 corridas Tamaño de Población: 450 individuos. Número máximo de generaciones: 500.	
M. Encapsulamiento	*M. Poblacional*
100% Zona Factible 15% Sol. óptimas	100% Zona Factible 25% Sol. óptimas
Circuito FDDI.	
20 corridas Tamaño de la Población: 600 individuos. Número máximo de generaciones: 2,000.	
M. Encapsulamiento	*M. Poblacional*
100% Zona Factible Mejor con 16 compuertas	100% Zona Factible Mejor con 15 compuertas
Circuito de Katz.	
20 corridas Tamaño de Población: 880 individuos. Número máximo de generaciones: 4,000.	
M. Encapsulamiento	*M. Poblacional*
30% Zona Factible Mejor con 19 comp.	75% Zona Factible Mejor con 10 comp.
Multiplicador de 3 bits.	
5 corridas Tamaño de Población: 1,800 individuos. Número máximo de generaciones: 13,000.	
M. Encapsulamiento	*M. Poblacional*
0% Zona Factible ND	20% Zona Factible Mejor con 148 comp.

Tabla 6.41: Tabla comparativa del desempeño de los enfoques de encapsulamiento y poblacional/autoadaptativo. ND = No se alcanzó la zona factible.

Capítulo 7

Conclusiones.

- La representación que se utilizó, Representación Postfija, descrita en el capítulo 4, resulta más fácil de evaluar ya que sólo hace uso de una pila estática.
- El uso del encapsulamiento dio como resultado, además de la reutilización de código genético, el poder utilizar el operador *EGL*, mediante el cual un mismo individuo puede representar más soluciones al asignarse diferentes valores a las p's y posteriormente seleccionar el de mejor aptitud.
- El Método Poblacional y Autoadaptativo permitió explorar paulatinamente el espacio de búsqueda lo que se traduce en un esfuerzo computacional menor de lo que se requiere en el Método de Encapsulamiento. Además de ésto, en todos los ejemplos el primer método tuvo mejor comportamiento que el segundo.
- En cualquiera de los dos métodos propuestos en esta tesis, ya sea el de Encapsulamiento o el Poblacional/Autoadaptativo, el parámetro de longitud máxima permitida juega un papel determinante en el comportamiento del algoritmo.
- En general, podemos decir que el Método Poblacional/Autoadaptativo tiene un buen desempeño, ya que al estar autoadaptando los parámetros de longitud máxima permitida, porcentaje de cruza y mutación, se introduce un mecanismo que evita la pérdida de diversidad de los individuos de la población, evitando con ello una convergencia prematura.

7.1 Trabajos Futuros.

En un futuro las líneas a seguir serían las siguientes:

- Probar con diferentes funciones de aptitud, hasta encontrar una cuyo comportamiento sea más robusto para cualquier tipo de circuito que se desee utilizar.
- Aplicar el operador de Edición una vez que se llegue a la zona factible. Esto con la finalidad de reducir la expresión y producir circuitos con el menor número de compuertas posibles.
- Utilizar árboles mitad y mitad con la finalidad de tener una población con mayor diversidad entre sus individuos. Esto evitará la convergencia prematura del algoritmo.
- En el Método Autoadaptativo, incluir un mecanismo que permita que el parámetro de longitud máxima adapte su valor de acuerdo al circuito que se está optimizando, ya que se observó que este parámetro determina en gran medida el comportamiento que tendrá el programa.
- Adoptar un enfoque multiobjetivo, en donde se tuvieran presente la funcionalidad, la reutilización de código genético y el número de compuertas del circuito.

Bibliografía

[1] Carlos A. Coello Coello , Alan D. Christiansen, and Arturo Hernández Aguirre. Diseño Óptimo de Circuitos Lógicos usando Algoritmos Genéticos. In *Primer Encuentro de Computación,* Taller de Aprendizaje, pages 1-10, Querétaro, Querétaro, Septiembre 1997.

[2] Carlos A. Coello 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] M. Morris Mano and Charles R. Kime *Fundamentos de Diseño Lógico y Computadoras.* Ed. Prentice Hall.

[4] Carlos A. Coello Coello, Alan D. Christiansen and Hernández Aguirre. Towards Automated Evolutionary Design of Combinantional Circuits. *Computers and Electrical Engineering,* 27(1):1-28, January 2001.

[5] Carlos A. Coello Coello, Alan D. Christiansen and Arturo Hernández Aguirre. Use of Evolutionary Techniques to Automated the Design of Combinantional Circuits. *International Journal of Smart Engineering System Design,* 2(4):299-314, June 2000.

[6] Carlos A. Coello Coello and Arturo Hernández Aguirre. Design of Combinantional Logic Circuits through an Evolutionary Multiobjetive Optimization Approach. Technical Re-

port Lania-RI-2000-05, Laboratorio Nacional de Informática Avanzada, 2000.

[7] Carlos A. Coello Coello, Arturo Hernández Aguirre, and Bill P. Buckles. Evolutionary Multiobjetive Design of Combinantional 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.

[8] Carlos A. Coello, Zavala G. Rosa Laura, Benito Mendoza G., and Arturo Hernández Aguirre. Ant Colony System for the Design of Combinantional Logic Circuits. In Julian Miller, Adrian Thopmson, Peter Thomson, and Terence C. Fogarty, editors, *Evolvable Systems: From Biology to Hardware,* pages 21-30, Edinburgh, Scotland, April 2000.

[9] Carlos A. Coello Coello, Zavala G. Rosa Laura, Benito Mendoza G., and Arturo Hernández Aguirre. Automated Design Combinantional Logic Circuits using the Ant System. Technical Report Lania-RI-2000-07, Laboratorio Nacional de Informática Avanzada, 2000.

[10] Charles Darwin. *The Origin of Species by Means of Natural Selection or the preservation of Favored RAces in the Struggle for life.* Random House, New York, 1993. (Publicado originalmente en 1929).

[11] George J. Friedman. Selective feedback computers for engineering synthesis and nervous system analogy. Master's thesis, Universoty of California at Los Angeles, February 1959.

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

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

[14] John R. Koza. Herarchical genetic algorithms operating on populations of computer programs. In N. S. Sridharan, editor, *Proceedings of 11th International Joint Conference on Artificial Intelligence,* pages 768-774, San Mateo, California, 1989. Morgan Kaufmann.

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

[16] John R. Koza Scalable learning in genetic programming using automatic function definition. In Jr. Kenneth E. Kinnear, editor, *Advances in Genetic Programming,* chapter 5, pages 99-117. The MIT Press, Cambridge, Massachusetts, 1994.

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

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

[19] Julian F. Miller, P. Thomson, and T. Fogarty. Desingning electronic circuits using evolutionary algorithms. Arithmetic circuits: A case study. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science,* pages 105-131. Morgan Kraufmann, Chichester, England, 1998.

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

[21] Carlos A. Coello, Alan D. Christiansen, and Arturo Hernández Aguirre. Using Genetic Algorithms to Design Combinantional Logic Circuits. In Cihan H. Dagli, Metin Akay, C.L. Philip Chen, Benito R. Fernández, and Joydeep Ghosh, editors. *ANNIE'96. Intelligent Engineering trough Artificial Neural Networks,* volume 6 of *Smart Engineering Systems: Neural Networks, Fuzzy Logic and Evolutionary Programming,* pages 391-396, November 1996.

[22] George Boole. *An Investigation of the Laws of Thougt.* Dover Pub., New York, 1954.

[23] M. Morris Mano. *Diseño Lógico.* Ed. Prentice Hall.

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

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

[26] Shannon, C.E. (1938) A symbolic analysis of relay and switching circuits, *Transactions of the AIEE*, 57, pp. 713-723.

[27] Edwin D. de Jong, Richard A. Watson and Jordan B. Pollack. Reducing Bloat and Promoting Diversity using Multi-Objective Methods. *GECCO-2001 Proceedings of the Genetic and Evolutionary Computation Conference.* July 7-11, 2001.

[28] Diseño de Circuitos Lógicos Combinatorios usando Programación Genética, by Eduardo Serna Pérez (MSc in Artificial Intelligence). Defended on Febraury 16th, 2001. This thesis was advised by Dra. Katya Rodríguez Vázuez.