CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN DE COMPUTACIÓN

Diseño de circuitos lógicos combinatorios usando optimización mediante cúmulos de partículas

Tesis que presenta

Erika Hernández Luna

Para obtener el grado de

Maestro en Ciencias

En la especialidad de

Ingeniería Eléctrica
Opción Computación

Director de Tesis: **Dr. Carlos A. Coello Coello**

México, D.F., febrero 2004

Resumen

En esta tesis se propone un algoritmo para el diseño y la posterior optimización de circuitos lógicos combinatorios a nivel de compuertas usando un algoritmo de optimización mediante cúmulos de partículas y tres enfoques de representación distintos: binaria, entera A y entera B . A pesar de que existen muchos criterios para determinar cual es el diseño de costo mínimo de un circuito, en este trabajo se utiliza una métrica basada en el número de compuertas necesarias para la implementación en hardware del circuito.

El algoritmo de optimización mediante cúmulos de partículas (*Particle Swarm Optimization* PSO) es una técnica de optimización numérica de funciones no lineales que se encuentra influenciada fuertemente por algunas otras corrientes como vida artificial, psicología social, ingeniería y ciencias de la computación, que ha conseguido su éxito gracias a que tiene un bajo costo computacional, una fácil implementación y un excelente desempeño.

Las técnicas evolutivas han sido sumamente útiles en el proceso de diseño de circuitos electrónicos debido a su poder exploratorio, pues al contar con una población de soluciones potenciales permiten evaluar diversas regiones del espacio de diseño.

También se ha visto a los largo de los años que el diseño de circuitos mediante las metodologías tradicionales es un proceso complejo que requiere de tiempo y experiencia por parte del diseñador humano, por lo que se pretende que al utilizar técnicas evolutivas puedan hallarse diseños que son radicalmente diferentes que los encontrados hasta el momento por estos diseñadores.

El PSO ha utilizado una representación binaria y frecuentemente una representación real para codificar las soluciones de los problemas, sin embargo en este trabajo además de presentarse una versión binaria, se hicieron los ajustes necesarios para presentar también una propuesta basada en dos enfoques distintos de una versión entera del algoritmo de optimización mediante cúmulos de partículas para el diseño de circuitos lógicos combinatorios a nivel de compuertas.

Los dos enfoques de la representación entera y el de la versión binaria del algoritmo propuesto se validaron usando algunos ejemplos tomados de la literatura y comparados contra otros tres enfoques: el diseño realizado por un experto humano, el diseño resultante de aplicar el algoritmo genético de cardinalidad N (NGA), y el diseño obtenido aplicando el algoritmo genético multiobjetivo (MGA).

Abstract

This thesis presents an algorithm for the design and the optimization of combinational logic circuits at gate-level using a particle swarm optimization algorithm and three different representation approaches: binary, integer A and integer B. Although there are many criteria to determine the minimal-cost circuit design, in this work we used a metric based on the number of gates necessaries for the hardware implementation of the circuit.

The Particle Swarm Optimization (PSO) algorithm is a numerical optimization technique for nonlinear functions that is strongly influenced by other areas such as artificial life, social psychology and computer science. PSO has been very successful because it has a low computacional cost, an easy implementation and an excellent performance.

Evolutionary techniques have been very useful for designing electronic circuits due to their exploratory power, since they operate using a population of potential solutions which can evaluate several regions of design space at the same time.

It has been seen along the years that designing logic circuits using traditional techniques is a complex process which requires considerable time and whose success depends on the human designer's experience. Because of that, we propose the use of evoluationary techniques to find circuits which, are often radically different from those typically found by human designers.

PSO has been used sometimes with a binary representation and more often with a real numbers encoding. However in this thesis we present two integer approaches besides the binary version of the particle swarm optimization algorithm, which are intended for designing combinational logic circuits.

The binary version and the two integer approaches of the proposed algorithm are validated using several examples taken from the specialized literature and are compared against human designers, the n-cardinality genetic algorithm (NGA) and the multiobjective genetic algorithm (MGA).

Agradecimientos

A Gelus y a Manolo por haber guiado mis pasos para llegar hasta aquí y en el camino darme a probar el sabor del triunfo y enseñarme a no querer abandonarlo nunca. A Sandy y a Isra por estar siempre conmigo y por el gran apoyo que me han brindado incluso cuando ni siquiera sabía que lo necesitaba. A mi querido Daniel porque cuando estamos juntos todo parece ser mejor.

A mis seres queridos y a todos aquellos que han influido en mi vida a lo largo de mis años de estudio y que con sus acertadas palabras me condujeron por el mejor camino.

Al Dr. Coello por sus invaluables consejos y por ser una fuente de inspiración que motiva a seguir siempre adelante.

A CONACyT por el apoyo otorgado a través de la beca proporcionada durante mi estancia en el programa de maestría y a través del proyecto 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.

Contenido

1. Introducción **1**

2. Circuitos lógicos combinatorios **5**
2.1. Algebra booleana 6
2.1.1. Tablas de verdad 7
2.1.2. Compuertas lógicas 8
2.1.3. Postulados, leyes y teoremas del álgebra booleana . . . 12
2.2. Formas Estándar 14
2.2.1. Minitérminos y maxitérminos 14
2.2.2. Suma de productos 17
2.2.3. Producto de sumas 17
2.3. Procedimiento de diseño de circuitos lógicos 18
2.4. Planteamiento formal del problema de diseño de circuitos . . . 19
2.5. Simplificación de circuitos lógicos 20
2.5.1. Simplificación algebraica 21
2.5.2. Mapas de Karnaugh 22
2.5.3. Método de Quine-McCluskey 27
2.5.4. Técnicas evolutivas 31

3. Conceptos básicos **37**
3.1. Optimización 37
3.1.1. Optimización con y sin restricciones 37
3.1.2. Optimización global y local 38
3.1.3. Optimización estocástica y determinística 38
3.1.4. Algoritmos de optimización 39
3.2. Computación evolutiva 39
3.2.1. Programación evolutiva 40
3.2.2. Estrategias evolutivas 41

3.2.3. Algoritmos genéticos 41
3.3. Algoritmo de optimización mediante cúmulos de partículas . . 41
3.3.1. Algoritmo de PSO binario 44
3.3.2. Algoritmo de PSO real 47
3.3.3. Parámetros del PSO 48

4. Descripción de la técnica **51**
4.1. Representación de los circuitos lógicos 51
4.1.1. Representación interna de las celdas 54
4.1.2. Cardinalidad 55
4.2. Evaluación 57
4.3. Obtención de expresiones booleanas 58
4.4. Algoritmo de optimización mediante cúmulos de partículas para el diseño de circuitos lógicos combinatorios 60
4.5. Mutación uniforme 63
4.6. Función de aptitud 64

5. Circuitos de una salida **67**
5.1. Experimentos 68
5.2. Ejemplo 1 72
5.3. Ejemplo 2 75
5.4. Ejemplo 3 78
5.5. Ejemplo 4 81
5.6. Ejemplo 5 84
5.7. Ejemplo 6 87
5.8. Ejemplo 7 90
5.9. Ejemplo 8 93
5.10. Ejemplo 9 96

6. Circuitos de múltiples salidas **101**
6.1. Ejemplo 1 102
6.2. Ejemplo 2 105
6.3. Ejemplo 3 108
6.4. Ejemplo 4 112

7. Conclusiones y trabajo futuro **117**

Bibliografía **121**

Lista de tablas

2.1. Tabla de verdad para dos variables de entrada. 7
2.2. Distintos símbolos y terminología para la operación AND. . . 8
2.3. Tabla de verdad de la compuerta AND. 8
2.4. Distintos símbolos y terminología para la operación OR. . . . 9
2.5. Tabla de verdad de la compuerta OR. 9
2.6. Tabla de verdad de la compuerta NOT. 10
2.7. Tabla de verdad de la compuerta XOR. 11
2.8. Minitérminos de tres variables. 15
2.9. Maxitérminos de tres variables. 15
2.10. Equivalencia entre una suma de minitérminos y un producto de maxitérminos. 16
2.11. Agrupación de términos para encontrar los implicantes primos de la función $S = \sum(0, 1, 2, 5, 6, 7, 8, 9, 10, 14)$. 28
2.12. Resultado de la comparación y combinación de términos en grupos adyacentes de la tabla 2.12. 29
2.13. Resultado de la comparación y combinación de términos en grupos adyacentes de la tabla 2.12. 29
2.14. Diagrama de implicantes primos de las tablas 2.11, 2.12 y 2.13 30
2.15. Resultado de la inclusión de los implicantes primos esenciales en la suma mínima. 31

5.1. Tabla de verdad del ejemplo 1. 72
5.2. Análisis comparativo de 20 corridas del ejemplo 1. 73
5.3. Comparación de resultados entre el PSO, un diseñador humano, el MGA y el NGA para el ejemplo 1. 74
5.4. Tabla de verdad del ejemplo 2. 75
5.5. Análisis comparativo de 20 corridas del ejemplo 2. 76
5.6. Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 2. 77

5.7. Tabla de verdad del ejemplo 3. 78
5.8. Análisis comparativo de 20 corridas del ejemplo 3. 79
5.9. Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 3. 80
5.10. Tabla de verdad del ejemplo 4. 81
5.11. Análisis comparativo de 20 corridas del ejemplo 4. 82
5.12. Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 4. 83
5.13. Tabla de verdad del ejemplo 5. 84
5.14. Análisis comparativo de 20 corridas del ejemplo 5. 85
5.15. Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 5. 86
5.16. Tabla de verdad del ejemplo 6. 87
5.17. Análisis comparativo de 20 corridas del ejemplo 6. 88
5.18. Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 6. 89
5.19. Tabla de verdad del ejemplo 7. 90
5.20. Análisis comparativo de 20 corridas del ejemplo 7. 91
5.21. Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 7. 92
5.22. Tabla de verdad del ejemplo 8. 93
5.23. Análisis comparativo de 20 corridas del ejemplo 8. 94
5.24. Comparación de resultados entre el PSO, un diseñador humano, el MGA y el NGA para el ejemplo 8. 95
5.25. Tabla de verdad del ejemplo 9. 96
5.26. Análisis comparativo de 20 corridas del ejemplo 9. 97
5.27. Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 9. 98

6.1. Tabla de verdad del ejemplo 1. 102
6.2. Análisis comparativo de 20 corridas del ejemplo 1. 103
6.3. Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 1. 104
6.4. Tabla de verdad del ejemplo 2. 105
6.5. Análisis comparativo de 20 corridas del ejemplo 2. 106
6.6. Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 2. 107
6.7. Tabla de verdad del ejemplo 3. 109

6.8. Análisis comparativo de 20 corridas del ejemplo 3. 110
6.9. Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 3. 111
6.10. Tabla de verdad del ejemplo 4. 113
6.11. Análisis comparativo de 20 corridas del ejemplo 4. 114
6.12. Comparación de resultados entre el PSO, un diseñador humano, el MGA y el NGA para el ejemplo 4. 114

Lista de figuras

2.1. Ejemplo de una expresión booleana. 6
2.2. Compuerta AND de dos entradas. 9
2.3. Compuerta OR de dos entradas. 10
2.4. Compuerta NOT. 10
2.5. Compuerta XOR de dos entradas. 11
2.6. Ejemplo de una suma de productos. 17
2.7. Diagrama de circuito de una suma de productos (SOP). 18
2.8. Ejemplo de un producto de sumas. 18
2.9. Diagrama de circuito de un producto de sumas (POS). 19
2.10. Ejemplo de funciones booleanas equivalentes. 21
2.11. Ejemplo de un diagrama de circuito y su expresión booleana. . 21
2.12. Diagrama de circuito y su expresión booleana simplificada. . . 22
2.13. Mapa de Karnaugh de dos variables. (a)Forma general. (b) Mapa de $S = A'B' + A'B = \sum(0,1) = \prod(2,4)$ 23
2.14. Mapa de Karnaugh de tres variables. (a)Forma general. (b) Mapa de $S = A'B'C' + A'B'C + A'BC + AB'C' = \sum(0,1,3,4) = \prod(2,5,6,7)$ 24
2.15. Mapa de Karnaugh de cuatro variables. (a)Forma general. (b) Mapa de $S = A'B'C' + A'B'C + A'BC + AB'C' = \sum(0,1,3,4) = \prod(2,5,6,7)$ 25
2.16. Adyacencia de celdas en los extremos de un mapa de cuatro variables. 26
2.17. Mapa de Karnaugh de cuatro variables con agrupamientos de 1, 2, 4 y 8 celdas. 26
2.18. Representación de matriz para el diseño de circuitos lógicos. . 33

3.1. Algoritmo general de la programación evolutiva. 40
3.2. Algoritmo general de las estrategias evolutivas. 41
3.3. Algoritmo general de los algoritmo genéticos. 42

3.4. Modelo Gbest del PSO. 44
3.5. Modelo Lbest del PSO. 44
3.6. Versión binaria del algoritmo de optimización mediante cúmulos de partículas. 46
3.7. Versión real del algoritmo de optimización mediante cúmulos de partículas. 48
3.8. Efecto de la eliminación del parámetro V_{max} en el algoritmo de PSO [23]. 49
3.9. Efecto de la variación del parámetro ϕ en el algoritmo de PSO [23]. 50

4.1. Representación de matriz para el diseño de circuitos lógicos. . 52
4.2. Matriz de 5×5 con un espacio de búsqueda de 5^{75}. 53
4.3. Distintos tipos de representación. 54
4.4. Una tercia de la matriz con representación entera. 54
4.5. Una tercia de la matriz con representación binaria. 55
4.6. Uso de una cardinalidad de dos para la selección de celdas . . 56
4.7. Entradas a las celdas de la matriz. 57
4.8. Matriz de ejemplo para la obtención de la expresión booleana. 58
4.9. Arbol de derivación para la salida S_0de la expresión booleana en notación prefija de la matriz mostrada en la figura 4.8. . . . 59
4.10. Derivación completa de la expresión booleana para la salida S_0 a partir de la matriz de la figura 4.8. 59
4.11. Pseudocódigo del algoritmo de optimización mediante cúmulos de partículas. 61
4.12. Ejemplo del funcionamiento de la representación entera A y entera B. 63
4.13. Ejemplo de mutación uniforme sobre una representación entera. 64
4.14. Aptitud de una solución no factible. 65
4.15. Aptitud de una solución factible. 66

5.1. Archivo con formato blif. 69
5.2. Programa en lenguaje Scheme que evalúa la expresión booleana obtenida del archivo blif de la figura 5.1. 71
5.3. Gráfica de convergencia del ejemplo 1 de la corrida ubicada en la mediana. 73
5.4. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 1. 74

5.5. Gráfica de convergencia del ejemplo 2 de la corrida ubicada en la mediana. 76
5.6. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 2. 77
5.7. Gráfica de convergencia del ejemplo 3 de la corrida ubicada en la mediana. 79
5.8. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 3. 80
5.9. Gráfica de convergencia del ejemplo 4 de la corrida ubicada en la mediana. 82
5.10. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 4. 83
5.11. Gráfica de convergencia del ejemplo 5 de la corrida ubicada en la mediana. 85
5.12. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 5. 86
5.13. Gráfica de convergencia del ejemplo 6 de la corrida ubicada en la mediana. 88
5.14. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 6. 89
5.15. Gráfica de convergencia del ejemplo 7 de la corrida ubicada en la mediana. 91
5.16. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 7. 92
5.17. Gráfica de convergencia del ejemplo 8 de la corrida ubicada en la mediana. 94
5.18. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 8. 95
5.19. Gráfica de convergencia del ejemplo 9 de la corrida ubicada en la mediana. 98
5.20. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 9. 99

6.1. Gráfica de convergencia del ejemplo 1 de la corrida ubicada en la mediana. 103
6.2. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 1. 105

6.3. Gráfica de convergencia del ejemplo 2 de la corrida ubicada en la mediana. 107
6.4. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 2. 108
6.5. Gráfica de convergencia del ejemplo 3 de la corrida ubicada en la mediana. 110
6.6. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 3. 112
6.7. Gráfica de convergencia del ejemplo 4 de la corrida ubicada en la mediana. 115
6.8. Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 4. 115

Capítulo 1

Introducción

La producción de circuitos electrónicos es un proceso que demanda cada vez diseños mejores y más complejos, debido a las tecnologías actuales de fabricación. Sin embargo, tradicionalmente en el proceso de diseño se han adoptado técnicas que utilizan un conjunto de reglas y principios que además de haber existido durante décadas dependen de la habilidad del diseñador y no aseguran llegar al diseño óptimo. El diseño de circuitos lógicos combinatorios (no secuenciales) es un problema particular del diseño de circuitos electrónicos que tiene una complejidad tal que ha permanecido como un problema de investigación abierto a lo largo del tiempo.

El objetivo principal en el proceso de diseño de circuitos lógicos combinatorios es encontrar la expresión que proporcione el comportamiento deseado y que además minimice un cierto criterio. En el caso de este trabajo de tesis el criterio a minimizarse es el costo del circuito, medido en términos del número necesario de compuertas para un diseño dado. Existen muchos criterios para determinar cuál es esta expresión de costo mínimo. Sin embargo, un circuito lógico es una implementación en hardware de una función booleana y la reducción del número de literales de cada término reduce el número de entradas de cada compuerta logrando con ello la reducción del número de compuertas necesarias y por tanto la complejidad del circuito.

Las técnicas evolutivas han sido aplicadas al diseño de circuitos [8, 31], codificando los posibles diseños en *genotipos* a partir de los cuales se construyen los circuitos o *fenotipos*. Estas técnicas han resultado muy útiles en el diseño de circuitos lógicos combinatorios debido a su poder exploratorio, pues al contar con una población de soluciones potenciales permiten evaluar diversas regiones del espacio de diseño. El área dedicada al estudio y

aplicación de técnicas evolutivas para el diseño de circuitos electrónicos es el *hardware evolutivo* [39, 19, 42] el cual está influenciado por otras áreas, principalmente ciencias de la computación, biología e ingeniería electrónica. El hardware evolutivo puede dividirse en dos grupos principales de acuerdo a la forma en que se diseñan y prueban los circuitos:

- **Evolución intrínseca**: Es el proceso evolutivo en el cual cada fenotipo es contruído y probado en hardware reconfigurable[1].
- **Evolución extrínseca**: Utiliza un modelo de hardware y evalúa los fenotipos en software (simulación).

En este trabajo se utilizó la técnica evolutiva denominada *Optimización mediante cúmulos de partículas* (Particle swarm optimization - PSO) [21] que es uno de los algoritmos de la llamada inteligencia colectiva (swarm intelligence en inglés) [3] que se refiere a aquellas técnicas de búsqueda inspiradas en el comportamiento colectivo de las colonias de insectos y de algunas otras sociedades de animales.

El PSO tiene como idea principal simular el movimiento de un grupo de aves que buscan comida. En esta simulación, el comportamiento de cada individuo está afectado por un factor individual y uno social. El factor individual se refiere a las decisiones que ha tomado el individuo y que han funcionado mejor (en términos de su aptitud) y que afectarán las nuevas decisiones que pueda tomar. El factor social se refiere a las decisiones que han tomado el resto de los individuos de la población (dentro de un cierto vecindario) que han funcionado mejor y que afectarán las nuevas decisiones tomadas por los individuos en el vecindario. Cada individuo o partícula contiene su posición actual en el espacio de búsqueda, su velocidad actual y la mejor posición encontrada por el individuo hasta este punto de la búsqueda (el factor individual) [23].

Este trabajo de tesis se encuentra dentro del ámbito de la evolución extrínseca del hardware evolutivo al diseñar circuitos lógicos combinatorios a nivel de compuertas (sólo utilizando las funciones AND, OR, NOT y XOR) mediante la técnica evolutiva de PSO.

En el capítulo 2 se da una introducción a la teoría de circuitos lógicos combinatorios y el proceso de diseño. También se describen algunas de las

[1]Sistemas cuyo hardware puede modificarse de manera que se adapte a nuevas condiciones del mismo o a distintas aplicaciones.

distintas técnicas tradicionalmente usadas para el diseño de circuitos lógicos como los mapas de Karnaugh y el método de Quine-McCluskey, además de algunos otros enfoques evolutivos como el algoritmo genético de cardinalidad N y el algoritmo genético multiobjetivo.

En el capítulo 3 se presenta un panorama de lo que son los problemas de optimización en ingeniería y su importancia. Después se da un breve repaso de lo que es la computación evolutiva, los términos más importantes y sus principales paradigmas para después dar una descripción del algoritmo de optimización mediante cúmulos de partículas.

En el capítulo 4 se describe la técnica propuesta para resolver el diseño de circuitos lógicos combinatorios detallando cada uno de los elementos que la componen como la representación, la evaluación y obtención de expresiones booleanas, el algoritmo utilizado y la función de aptitud.

En el capítulo 5 se muestran los resultados obtenidos por la técnica propuesta a través de una serie de experimentos realizados con circuitos lógicos combinatorios de una sola salida. Se presentan las tablas de verdad de los circuitos de los experimentos, tablas comparativas de las tres representaciones (binaria, entera A y entera B) donde se evalúa también el desempeño contra otras técnicas evolutivas además de los diagramas lógicos y las expresiones booleanas en notación prefija de los mejores circuitos encontrados.

En el capítulo 6 se discuten los resultados obtenidos por la técnica para experimentos con circuitos de múltiples salidas. La estructura de este capítulo es similar a la del capítulo 5 ya que se presentan también las tablas de verdad de los circuitos que se están probando, las tablas comparativas y los diagramas lógicos con la expresión booleana correspondiente de los mejores circuitos encontrados.

Por último, en el capítulo 7 se dan las conclusiones acerca de los resultados obtenidos así como algunas ideas para realizar algunos trabajos posteriores y que pudieran mejorar el desempeño del algoritmo presentado.

Capítulo 2

Circuitos lógicos combinatorios

Un *sistema digital* es una combinación de dispositivos diseñada para manipular variables que solamente tomen valores discretos. Un *circuito lógico combinatorio* es un sistema digital que opera en modo binario, es decir que los valores que pueden existir sólo son el 1 y el 0, además de que en cualquier instante de tiempo la salida depende únicamente de los niveles lógicos presentes a la entrada.

En 1854, George Boole publica su libro titulado "Las leyes del pensamiento" en el que aproximó la lógica en una nueva dirección reduciéndola a una álgebra simple llamada desde entonces *algebra booleana*. Sin embargo fue hasta 1937 que Claude Shannon de los Laboratorios Bell llevó a cabo el enlace entre la lógica y la electrónica [37, 11]. Shannon demostró cómo las operaciones booleanas elementales se podían representar mediante circuitos conmutados eléctricos, y cómo la combinación de circuitos podía representar operaciones aritméticas y lógicas complejas. Posteriormente formalizó toda la estructura matemática para que los ingenieros pudieran aplicar la teoría a la práctica.

Básicamente, el álgebra booleana consiste en un método para resolver problemas de lógica y que recurre sólo a los valores binarios 0 y 1 y a tres operaciones básicas: AND, OR y NOT. Como ya se mencionó, los circuitos lógicos combinatorios tienen la característica de ser binarios por lo que es posible utilizar esta álgebra como herramienta para su análisis y diseño.

2.1. Algebra booleana

El álgebra booleana trata principalmente con constantes, variables y operaciones lógicas y su principal diferencia con respecto al álgebra tradicional es que en este caso se utilizan variables y constantes binarias, es decir que sólo pueden tomar el valor de 0 o de 1. Estos dos valores con frecuencia representan el nivel de voltaje presente en un determinado lugar del circuito, es decir que no representan números sino el estado de una variable de voltaje, mejor conocido como *nivel lógico*. Debido a que sólo existen dos valores, el álgebra booleana es más fácil de manejar que el álgebra ordinaria ya que no se tiene que lidiar con números decimales, números negativos, logaritmos, números imaginarios, etc., de hecho sólo cuenta con tres operaciones lógicas básicas:

1. Adición lógica: También conocida como operación OR. El símbolo más común para esta operación es el $(+)$. La salida será 1 si alguna de las variables es igual a 1.

2. Multiplicación lógica: Se le conoce también como operación AND. El símbolo de esta operación es el $(\cdot)$. La salida será 1 sólo cuando ambas variables sean iguales a 1.

3. Complemento: Conocida como operación de inversión o NOT. Su símbolo es el $(')$. Como su nombre lo indica, la salida siempre será la inversa del valor de la variable de entrada, es decir que si a la entrada existe un valor 1, a la salida se tendrá el valor 0.

En el álgebra booleana existen *funciones booleanas* (ver figura 2.1) que expresan la relación lógica entre las variables binarias y constan de una expresión algebraica formada por variables binarias, constantes 0 o 1 y los símbolos de operación lógica. Estas expresiones son muy útiles para la manipulación de variables lógicas y para expresar el funcionamiento de un circuito digital para determinados valores de entrada [40].

$$f = AB + BC$$

Figura 2.1: Ejemplo de una expresión booleana.

Explícitamente, el álgebra booleana $b(A)$ de un conjunto A es el conjunto de subconjuntos de A que pueden ser obtenidos mediante la aplicación de un número finito de operaciones del conjunto union (OR), intersección (AND) y complemento (NOT) y en donde cada uno de los elementos de $b(A)$ es una función booleana. El álgebra booleana también forma una lattice ya que un álgebra L es llamada una lattice cuando L es un conjunto no vacío y además existen dos operaciones binarias sobre L, $\vee$ y $\wedge$ tales que son idempotentes, conmutativas y asociativas además de satisfacer las leyes de absorción (las cuales se explicarán en la sección 2.1.3).

2.1.1. Tablas de verdad

Una tabla de verdad o de combinaciones (ver tabla 2.1) es un método gráfico para representar todos los posibles valores de las variables y mostrar los resultados de la operación deseada. Es decir, que en estas tablas se muestra la forma en que la salida del circuito se comportará con las diversas combinaciones de los niveles lógicos a la entrada [40, 11].

A	**B**	**S**
0	0	1
0	1	0
1	0	0
1	1	0

Tabla 2.1: Tabla de verdad para dos variables de entrada.

El tamaño de la tabla depende directamente del número de variables de entrada al circuito. Si por ejemplo se tienen dos valores de entrada, existen entonces cuatro posibles combinaciones como puede verse en la tabla 2.1. La relación existente entre variables y combinaciones puede expresarse mediante la ecuación 2.1.

$$\textit{Número de combinaciones} = 2^{\textit{Número de variables booleanas}} \tag{2.1}$$

Debido a sus características, las tablas de verdad son útiles para representar el diseño de un circuito lógico.

2.1.2. Compuertas lógicas

Las compuertas lógicas son circuitos electrónicos que implementan en hardware operaciones lógicas del álgebra booleana y que operan con una o más señales de entrada para generar una señal de salida [27]. Cualquier circuito lógico, por muy complejo que sea, puede ser realizado con las tres compuertas básicas AND, OR y NOT. Sin embargo, existen algunas configuraciones que se presentan muy frecuentemente y a las que, por ende, se les ha asignado un nombre y símbolo propio.

Compuerta AND

La compuerta AND implementa la multiplicación lógica, la cual puede ser representada mediante diferentes símbolos y terminologías como puede verse en la tabla 2.2.

Nombre	**Símbolo**
AND	
Multiplicación booleana	$\cdot$
Intersección	$\cap$
Conjunción	$\wedge$

Tabla 2.2: Distintos símbolos y terminología para la operación AND.

A	**B**	**S = AB**
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 2.3: Tabla de verdad de la compuerta AND.

En la tabla 2.3 se muestra la tabla de verdad de la operación AND de dos variables de entrada y como puede verse, la operación AND proporciona un valor de 1 cuando la variable A y la variable B toman un valor de 1 y es 0 en cualquier otro caso.

Se puede decir entonces que la operación AND es exactamente igual que hacer una multiplicación ordinaria, en donde el resultado será 0 siempre que alguna de las variables que se multiplica tome este valor. Es decir que la compuerta AND tendrá un valor de 1 a la salida sólo cuando todas las variables de entrada sean igual a 1. El símbolo para representar una compuerta AND en el diagrama de un circuito lógico y la expresion booleana a su salida se muestra en la figura 2.2.

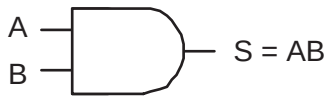


Figura 2.2: Compuerta AND de dos entradas.

Compuerta OR

La compuerta OR implementa la operación de adición lógica y al igual que la compuerta AND existen diversos términos y símbolos para representar a esta función, como muestra la tabla 2.4.

Nombre	**Símbolo**
OR	$+$
Adición booleana	$+$
Unión	$\cup$
Disyunción	$\vee$

Tabla 2.4: Distintos símbolos y terminología para la operación OR.

A	**B**	**S = A + B**
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 2.5: Tabla de verdad de la compuerta OR.

La tabla de verdad de la operación OR se muestra en la tabla 2.5. Como puede observarse, para esta función la salida será 1 siempre que alguna de las variables de entrada tome el valor de 1. En cualquier otro caso el valor de salida será 0. Dicho de otra forma, la salida de la compuerta OR será 0 sólo cuando todas las variables de entrada sean 0. La figura 2.3 muestra la representación de la compuerta OR en un diagrama de un circuito lógico y la expresión booleana a su salida.

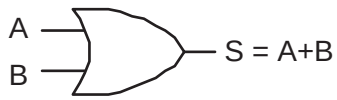


Figura 2.3: Compuerta OR de dos entradas.

Compuerta NOT

La compuerta NOT es mejor conocida como inversor e implementa la operación de complemento como muestra la tabla 2.6.

A	**S = A'**
0	1
1	0

Tabla 2.6: Tabla de verdad de la compuerta NOT.

La tabla de verdad de la compuerta NOT muestra que en esta operación sólo se tiene una variable de entrada, la cual tendrá como salida el nivel lógico contrario al nivel lógico de entrada, es decir que si a la entrada se tiene el valor de 1, a la salida se tendrá un 0 y viceversa. La figura 2.4 muestra la representación de la compuerta NOT en un diagrama de un circuito lógico y la expresión booleana a su salida.

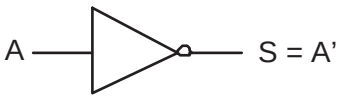


Figura 2.4: Compuerta NOT.

Compuerta XOR

La compuerta XOR también es conocida como compuerta OR-exclusiva y no implementa una operación básica del álgebra booleana. A pesar de eso se presenta frecuentemente en los sistemas digitales y por ello es útil en el diseño e implementación de circuitos lógicos. La tabla 2.7 muestra la tabla de verdad de la compuerta XOR.

A	**B**	$S = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 2.7: Tabla de verdad de la compuerta XOR.

Se puede ver en la tabla de verdad de la compuerta XOR que esta operación detecta cuando las variables de entrada tiene valores diferentes, ya que cuando los niveles lógicos de las entradas son opuestos se tiene un 1 a la salida, mientras que cuando los dos niveles lógicos son iguales, se tiene un 0. En la ecuación 2.2 se muestra la función booleana de la compuerta XOR en términos de las operaciones básicas del álgebra booleana.

$$S = A'B + AB' = A \oplus B \tag{2.2}$$

Debido a que la configuración de compuertas de la ecuación 2.2 se presenta muy a menudo se le ha asignado una representación específica para los diagramas de circuitos además de un símbolo particular, los cuales se muestra en la figura 2.5.

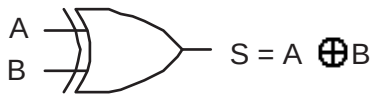


Figura 2.5: Compuerta XOR de dos entradas.

2.1.3. Postulados, leyes y teoremas del álgebra booleana

Los postulados, leyes y teoremas booleanos sirven para poder simplificar las expresiones booleanas que representan un circuito lógico y en consecuencia requerir un menor número de compuertas para su implementación.

Postulados

Los postulados describen el comportamiento de las operaciones básicas del álgebra booleana [11].

1. *a*) $0 \cdot 0 = 0$
   b) $1 + 1 = 1$
2. *a*) $1 \cdot 1 = 1$
   b) $0 + 0 = 0$
3. *a*) $1 \cdot 0 = 0 \cdot 1 = 0$
   b) $0 + 1 = 1 + 0 = 1$
4. *a*) $0' = 1$
   b) $1' = 0$

Leyes

Las tres leyes básicas del algebra booleana [11] se presentan a continuación:

1. Ley Conmutativa : Esta ley establece que no importa el orden en que se evalúen las operaciones AND y OR de dos variables ya que el resultado es el mismo.
   a) $A \cdot B = B \cdot A$
   b) $A + B = B + A$
2. Ley Asociativa : Establece que las variables de una operación AND o de una operación OR pueden ser agrupadas en cualquier forma.

a) $A \cdot (B \cdot C) = (A \cdot B) \cdot C$

b) $A + (B + C) = (A + B) + C$

3. Ley Distributiva : Se establece que una expresión puede desarrollarse multiplicando término a término como en el álgebra ordinaria así como llevar a cabo la factorización de términos.

   a) $A \cdot B + A \cdot C = A \cdot (B + C)$

   b) $(A + B) \cdot (A + C) = A + B \cdot C$

Teoremas

Existen diversos teoremas que resultan de la aplicación de los postulados y de las leyes mencionadas [11], sin embargo sólo se mostrarán los más útiles.

1. Teorema de identidad

   a) $A + 0 = A$

   b) $A \cdot 1 = A$

2. Teorema del elemento nulo

   a) $A + 1 = 1$

   b) $A \cdot 0 = 0$

3. Teorema del complemento

   a) $A + A' = 1$

   b) $A \cdot A' = 0$

4. Teorema de idempotencia

   a) $A + A = A$

   b) $A \cdot A = A$

5. Teorema de absorción

   a) $A \cdot (A + B) = A$

   b) $A + A \cdot B = A$

6. Teorema de involución

   a) $A = A''$

7. Teorema de DeMorgan

   a) $(A + B + C + \cdots)' = A' \cdot B' \cdot C' \cdot \cdots$

   b) $(A \cdot B \cdot C \cdot \cdots)' = A' + B' + C' + \cdots$

2.2. Formas Estándar

Para poder aplicar las técnicas de reducción comunes es necesario que las expresiones booleanas se encuentren en su forma estándar ya que facilitan los procedimiento de simplificación [11]. Existen diversas maneras de representar una función booleana algebraicamente pero pocas de ellas se consideran estándar. Las formas estándar tienen *términos de suma* y *términos de producto* que no implican operaciones aritméticas del álgebra booleana sino las operaciones lógicas AND y OR respectivamente [27]. Un ejemplo de término de suma puede ser $A + B + C'$ que consiste en una operación OR entre literales. Análogamente un ejemplo de término producto es $A'BC$ que consiste en una operación AND entre literales.

2.2.1. Minitérminos y maxitérminos

Un término producto en el que aparecen todas las variables sólo una vez de forma complementada o no se llama *minitérmino*. Una variable se encuentra complementada cuando su valor es 0. Cada minitérmino tiene la característica de que tiene un valor de 1 para la combinación de variables y un valor de 0 para todas las demás. El símbolo utilizado para designar un minitérmino es m_j donde j denota el equivalente decimal de la combinación binaria. En la tabla 2.8 se muestran los minitérminos para tres variables y su símbolo.

Por su parte, un *maxitérmino* es un término de suma que contiene a todas las variables de forma complementada o no, además de tener un valor de 0 para la combinación dada y un valor de 1 para todas las demás. Una variable se encuentra complementada cuando su valor es 1. Los maxitérminos se denotan con el símbolo M_j donde al igual que los minitérminos la j es el

A	**B**	**C**	**Minitérmino**	**Símbolo**
0	0	0	$A'B'C'$	m_0
0	0	1	$A'B'C$	m_1
0	1	0	$A'BC'$	m_2
0	1	1	$A'BC$	m_3
1	0	0	$AB'C'$	m_4
1	0	1	$AB'C$	m_5
1	1	0	ABC'	m_6
1	1	1	ABC	m_7

Tabla 2.8: Minitérminos de tres variables.

equivalente decimal de la combinación binaria. En la tabla 2.9 se muestran los maxitérminos para tres variables y su símbolo correspondiente.

A	**B**	**C**	**Maxitérmino**	**Símbolo**
0	0	0	$A + B + C$	M_0
0	0	1	$A + B + C'$	M_1
0	1	0	$A + B' + C$	M_2
0	1	1	$A + B' + C'$	M_3
1	0	0	$A' + B + C$	M_4
1	0	1	$A' + B + C'$	M_5
1	1	0	$A' + B' + C$	M_6
1	1	1	$A' + B' + C'$	M_7

Tabla 2.9: Maxitérminos de tres variables.

En las tablas 2.8 y 2.9 puede apreciarse que los minitérminos y los maxitérminos tienen la característica de que representan a una sola de las combinaciones de entrada por lo que se puede afirmar que existen tantos minitérminos o maxitérminos como combinaciones de los valores de las variables de entrada, además de que un minitérmino y el maxitérmino con el mismo subíndice son complementarios, es decir que $m_j = \overline{M_j}$ lo cual puede ser fácilmente verificado con las leyes de DeMorgan.

Como se mencionó anteriormente, una tabla de verdad representa el comportamiento del circuito debido a un determinado nivel lógico de las variables de entrada. A partir de una tabla de verdad se puede obtener una expresión

algebraica de la función booleana que está representando. Para poder obtener dicha expresión se tiene que hacer la suma lógica de los términos de producto para los que la función tome el valor de 1, que es lo mismo que llevar a cabo la suma de los minitérminos que producen un valor de 1 es decir una *suma de minitérminos* [27].

De manera análoga se puede obtener un *producto de maxitérminos* como expresión algebraica de la función booleana, que resulta de tomar el producto de los maxitérminos que a su salida tengan el valor de 0. La suma de minitérminos y el producto de maxitérminos describen la misma función booleana, como puede verse en la tabla 2.10.

A	**B**	**C**	**S**
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$\mathrm{S} = A'B'C + A'BC + AB'C + ABC = \sum_{j=0}^{7} m_j$$

$$\mathrm{S} = (A + B + C)(A + B' + C)(A' + B + C)(A' + B' + C) = \prod_{j=0}^{7} M_j$$

Tabla 2.10: Equivalencia entre una suma de minitérminos y un producto de maxitérminos.

En el diseño de circuitos, el producto de maxitérminos no es de uso frecuente ya que siempre puede ser sustituido por una suma de minitérminos. A continuación se resumirán algunas de las características más sobresalientes de los minitérminos.

1. Existen una cantidad de minitérminos igual al número de combinaciones de las variables de entrada, es decir que existen 2^n minitérminos donde n es el número de variables.

2. Una expresión booleana puede derivarse directamente de la tabla de

verdad como la suma lógica de minitérminos que tengan como salida un valor de 1.

3. El complemento f' de una función f es igual a la suma de los minitérminos no incluidos en f.

4. Si una función contiene a los 2^n minitérminos, su valor siempre será igual a 1.

2.2.2. Suma de productos

La suma de minitérminos es una expresión algebraica que contiene la cantidad máxima de literales en cada término lo que a menudo conlleva a que tenga más términos producto de los necesarios. En la simplificación de circuitos lógicos se tiene como objetivo reducir la cantidad de términos producto y de literales en los términos, obteniendo así una expresión reducida equivalente en forma de suma de productos (SOP - sum of products) [11]. En la figura 2.6 se muestra un ejemplo de SOP.

$$S = AC' + B'C + A'B$$

Figura 2.6: Ejemplo de una suma de productos.

El diagrama de circuito de una suma de productos consiste en un conjunto de compuertas AND seguido de una sola compuerta OR (figura 2.7), donde cada término producto necesita una compuerta AND y la suma lógica se forma con una compuerta OR que tiene como entrada las salidas de todas las compuertas AND. Este esquema de compuertas AND seguidas de una compuerta OR se conoce como *implementación en dos niveles*.

2.2.3. Producto de sumas

El producto de sumas (POS - product of sums) es una forma estándar de expresión algebraica de funciones booleanas y se obtiene formando un producto lógico de términos de suma [11]. Un ejemplo de un POS se muestra en la figura 2.8. La estructura de compuertas de un POS consiste de un grupo de compuertas OR seguidas de una compuerta AND que da como resultado una implementación en dos niveles (figura 2.9).

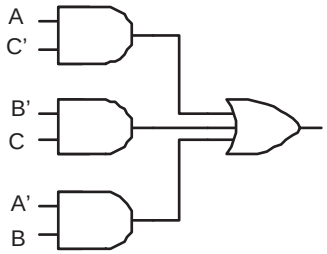


Figura 2.7: Diagrama de circuito de una suma de productos (SOP).

$$S = (A + C')(B' + C)(A' + B)$$

Figura 2.8: Ejemplo de un producto de sumas.

2.3. Procedimiento de diseño de circuitos lógicos

El proceso de diseño de un circuito lógico comienza con la especificación del problema a resolver y termina con un diagrama de circuito o una expresión booleana que lo represente [44]. De forma general se pueden enumerar los siguientes pasos en el proceso de diseño de un circuito lógico:

1. Determinar, a partir del enunciado del problema, la cantidad de entradas y de salidas necesarias.

2. Una vez determinadas las entradas y salidas se construye la tabla de verdad que ejemplifique el comportamiento deseado del circuito para todas las combinaciones posibles y establecer así la relación entre entradas y salidas.

3. A partir de la tabla de verdad se construye la expresión booleana que será simplificada posteriormente para tener así una función de salida en términos de las entradas. La simplificación es un paso muy importante en el diseño de circuitos ya que permite generar expresiones booleanas equivalentes y más sencillas.

4. Con las expresiones booleanas se traza el diagrama de circuitos lógicos que ayudará para su implementación en hardware.

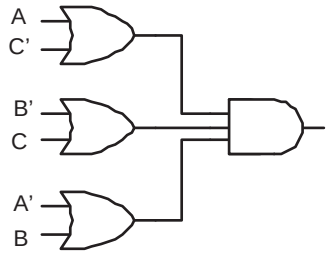


Figura 2.9: Diagrama de circuito de un producto de sumas (POS).

5. Por último, se lleva a cabo la verificación del comportamiento del circuito.

Un diseño práctico debe considerar algunas restricciones como la cantidad de compuertas, cantidad de entradas y de interconexiones, así como las limitaciones físicas de los circuitos integrados. En la mayoría de los casos, la simplificación de las expresiones booleanas busca satisfacer un solo objetivo y luego se intenta cumplir con nuevos criterios de desempeño.

2.4. Planteamiento formal del problema de diseño de circuitos

Formalmente podemos decir que el espacio booleano unidimensional $\mathcal{B}$ esta definido por el conjunto $\{0, 1\}$. Una variable booleana x puede tomar cualquier valor de este conjunto. Todos los vectores booleanos con n componentes forman el espacio booleano n-dimensional $\mathcal{B}^n$.

$$\mathcal{B}^n = \{\mathsf{X}/\mathsf{X} = (x_1, x_2, \ldots, x_n), x_i \in \mathcal{B}, \forall i = 1, \ldots, n\}$$

Sea $\mathsf{X} = (x_1, x_2, \ldots, x_n)$ un conjunto de n variables booleanas. El mapeo único de $\mathcal{B}^n$ a $\mathcal{B}$ es una función booleana $f(\mathsf{X})$.

Sea $\mathcal{C} = \{OR, AND, NOT, XOR\}$ un conjunto de compuertas lógicas booleanas con dos variables de entrada y sea $\mathcal{EB}(\mathcal{C})$ el conjunto de expresiones booleanas sobre $\mathcal{C}$, por ejemplo:

$$\begin{aligned} eb_1 &: y \text{ AND } z \\ eb_2 &: w \text{ XOR } (y \text{ OR } z) \end{aligned}$$

donde w, y y z son variables booleanas. Sea $x \in \mathcal{EB}(\mathcal{C})$ y definimos $|x|$ como el número de compuertas de x, por ejemplo:

$$\begin{aligned} x_1 &: (\text{NOT } y) \text{ AND } (w \text{ OR } z); \quad |x_1| = 3 \quad (NOT, AND, OR) \\ x_2 &: y \text{ XOR } (\text{NOT } w); \quad\quad\quad\quad |x_2| = 2 \quad (XOR, NOT) \end{aligned}$$

Sea $f(\mathsf{X})$ una función booleana y $\mathcal{F} \subset \mathcal{EB}(\mathcal{C})$ tal que $\forall x \in \mathcal{F}, x = f(\mathsf{X})$. Nuestro problema es encontrar $x^* \in \mathcal{F}$ tal que $|x^*| = \min |x|, \forall x \in \mathcal{F}$.

2.5. Simplificación de circuitos lógicos

El objetivo principal de la simplificación de circuitos lógicos es normalmente la minimización de la cantidad de hardware necesario para construir un sistema en particular, ya que a menor hardware se tendrá un menor costo final.

Por otro lado, la complejidad de un circuito lógico está relacionada directamente con la expresión booleana a partir de la cual se implementó. Esta expresión puede ser reducida para que contenga menos términos y que a su vez cada uno de éstos contenga menos literales. La nueva expresión puede entonces ser utilizada para implementar un nuevo circuito equivalente al original, es decir que genere la misma salida y por tanto la misma tabla de verdad pero mediante una función booleana distinta con un menor número de compuertas y conexiones.

La forma más simple de términos de suma de productos es una función llamada *suma mínima* que representará la suma con menor número de términos. Si existe más de una suma de productos que contenga el mínimo número de términos, la suma mínima será aquella que contenga el menor número de literales. Por ejemplo, las funciones de la figura 2.10 son equivalentes, pero la tercera de ellas es la suma mínima ya que contiene cuatro literales mientras que las otras dos tienen cinco literales cada una. Esto da como consecuencia una implementación en dos niveles con un diagrama de circuito con la

menor cantidad de compuertas y con la menor cantidad de entradas a las compuertas.

$$\boxed{\begin{array}{l} S = BC + ABC' \\ S = A'BC + AB \\ S = BC + AB \end{array}}$$

Figura 2.10: Ejemplo de funciones booleanas equivalentes.

2.5.1. Simplificación algebraica

Las expresiones booleanas pueden simplificarse mediante manipulaciones algebraicas. Sin embargo, como se verá a continuación, el procedimiento resulta complicado pues al carecer de reglas específicas para predecir los pasos sucesivos se requiere de una gran experiencia por parte del diseñador para poder aplicar los postulados y teoremas del álgebra booleana de manera que las expresiones puedan ser minimizadas.

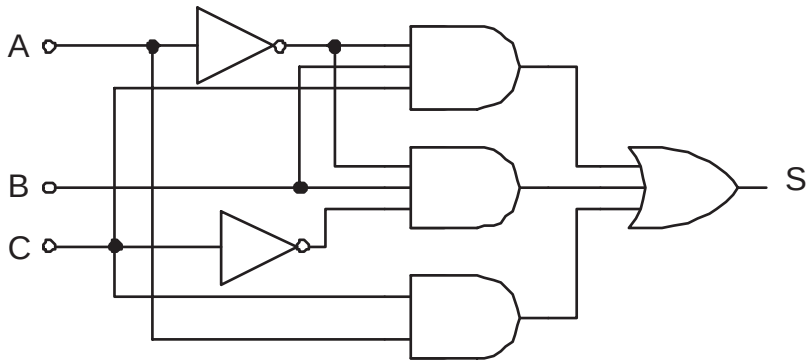


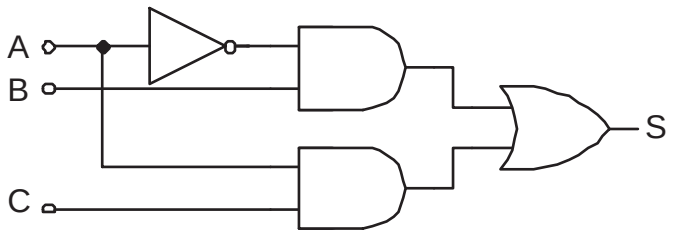
Figura 2.11: Ejemplo de un diagrama de circuito y su expresión booleana.

Otra desventaja de la simplificación algebraica es que después de llevar a cabo toda la manipulación algebraica no se puede saber cuándo se ha llegado a la expresión más sencilla. En la figura 2.11 se muestra el diagrama de circuito y la función booleana sin simplificar obtenida como resultado del

proceso de diseño para resolver algún problema. Si a la expresión booleana de la figura 2.11 se le aplican los postulados y teoremas del álgebra booleana podría obtenerse una expresión simplificada. En la figura 2.12 se muestra el resultado obtenido de la manipulación algebraica llevada a cabo para la reducción así como el diagrama de circuito donde puede apreciarse que la cantidad de compuertas necesarias para la implementación es menor que las necesitadas por el circuito original.

2.5.2. Mapas de Karnaugh

El mapa de Karnaugh es un método gráfico [18, 28] utilizado para simplificar una ecuación lógica o para convertir una tabla de verdad a su circuito lógico correspondiente mediante un proceso simple y ordenado. Como una tabla de verdad, el mapa de Karnaugh especifica el valor de la función de salida para cada combinación de valores de las variables independientes de entrada. El mapa ofrece también un diagrama visual de las distintas maneras en que puede expresarse una función booleana en forma estándar.



S = A'BC + A'BC' + AC
S = A'B(C+C') + AC
S = A'B(1) + AC
S = A'B + AC

Figura 2.12: Diagrama de circuito y su expresión booleana simplificada.

El resultado producido por el mapa siempre está en forma de suma de productos o de producto de sumas. Es decir, que se permite la simplificación de implementaciones en dos niveles pero el método no puede ser aplicado directamente a implementaciones más sencillas de los casos generales de un mayor número de niveles.

Mapas de dos, tres y cuatro variables

En la figura 2.13 se muestra un mapa de una función de dos variables que es un cuadrado de cuatro celdas (una celda para cada minitérmino).

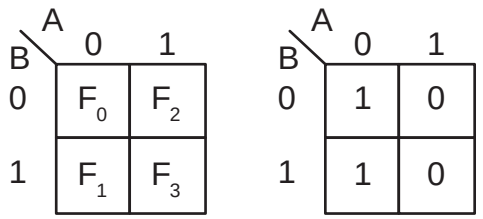
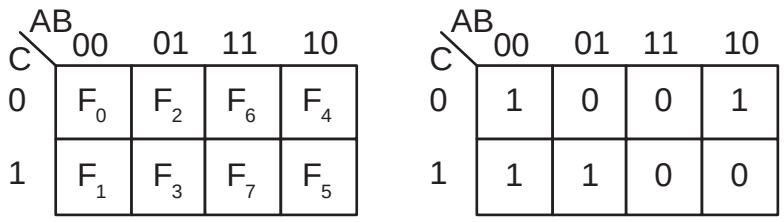


Figura 2.13: Mapa de Karnaugh de dos variables. (a)Forma general. (b) Mapa de $S = A'B' + A'B = \sum(0,1) = \prod(2,4)$

Para llenar el valor de cada celda se buscan sus coordenadas como valores de entrada en la tabla de verdad y se coloca un 1 en aquellas celdas cuyo valor en la salida sea 1 y 0 en las restantes [28]. Por ejemplo, en los mapas de la figura 2.13 se puede observar que las celdas F_0 y F_1 contienen un 1 mientras que F_2 y F_4 contienen un 0, ya que la función booleana de salida es $S = A'B' + A'B$ es decir que las coordenadas 00 y 01 del mapa de Karnaugh tendrán el valor 1 mientras que las coordenadas 10 y 11 tendrán el valor 0.

Es importante recalcar que la designación decimal de cada celda es igual al equivalente decimal del número binario formado por sus coordenadas. Los mapas para las funciones de tres y cuatro variables son extensiones directas del mapa de dos variables. El mapa de tres variables se muestra en la figura 2.14 con $2^3 = 8$ celdas y el mapa de cuatro variables se muestra en la figura 2.15 con $2^4 = 16$ celdas.

Cada 1 en el mapa corresponde a un minitérmino de la función de salida y pueden leerse directamente de aquí, tal como se hace en las tablas de verdad. Los minitérminos en cuadros adyacentes en el mapa pueden ser combinados debido a que sólo difieren en una sola variable.



(a) (b)

Figura 2.14: Mapa de Karnaugh de tres variables. (a)Forma general. (b) Mapa de $S = A'B'C' + A'B'C + A'BC + AB'C' = \sum(0, 1, 3, 4) = \prod(2, 5, 6, 7)$

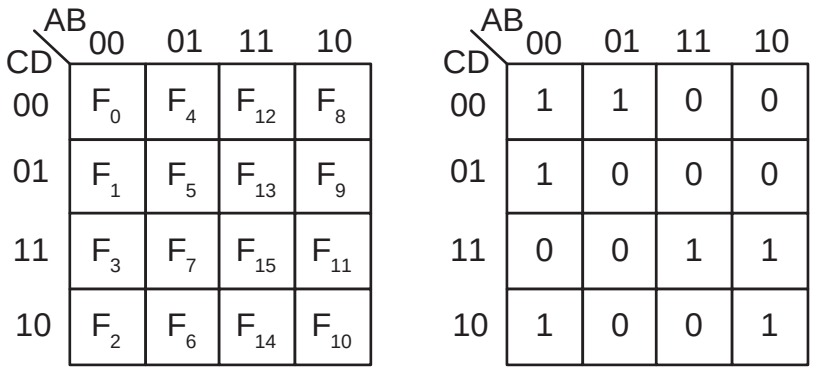
Implicantes primos

Los cuadrados adyacentes de los mapas de Karnaugh pueden combinarse debido al teorema del álgebra booleana que establece que $AB + AB' = A$. Es decir, que los términos producto puden ser combinados o simpilifcados cuando los números binarios difieren en un solo bit [28].

La expresion $ABCD + ABCD' = ABC$ tiene dos productos de cuatro variables con los números binarios 1111 y 1110 los cuales difieren solamente en el bit de menor orden por lo que pueden ser combinados. Sin embargo, los productos $ABCD$ y $A'BCD'$ no pueden ser combinados porque sus números binarios correspondientes son 1111 y 0110 que difieren en las posiciones de mayor y menor orden. Dos productos sólo pueden ser combinados si sus puntos correspondientes en un cubo de dimensión n de un mapa se encuentran a una distancia de 1. Los puntos con una distancia de 1 en un cubo de n dimensiones son celdas adyacentes en el mapa y por tanto representan productos que pueden ser combinados y que pueden ser rápidamente determinados por inspección.

En el proceso de inspección debe recordarse que las celdas a los extremos del mapa son adyacentes con las celdas que se encuentran exactamente al otro extremo en línea recta. La figura 2.16 ilustra estas adyacencias como por ejemplo las celdas F_0 y F_2 con números binarios 0000, 0010 y F_3 y F_{11} que tienen los números binarios 0011 y 1011.

En un mapa de cuatro variables, cada celda es adyacente a otras cuatro celdas correspondientes a las posiciones de los cuatro bits en los cuales el número binario puede diferir. Solamente se deben de tomar en cuenta para



(a) (b)

Figura 2.15: Mapa de Karnaugh de cuatro variables. (a)Forma general. (b) Mapa de $S = A'B'C' + A'B'C + A'BC + AB'C' = \sum(0, 1, 3, 4) = \prod(2, 5, 6, 7)$

la simplificación aquellas celdas que tengan el valor de 1 y el resto debe de ser ignorado [28].

La regla para obtener la expresión booleana simplificada a partir del mapa es escribir un producto término por cada par de celdas adyacentes con valor 1 y un producto término por cada celda que tenga valor de 1 y que no sea adyacente a ninguna otra celda. La figura 2.17 muestra el mapa de Karnaugh y la expresión booleana resultante para esta regla.

Las siguientes reglas en la obtención de la expresión booleana se aplican en los casos en que pueden ser agrupadas más de un par de celdas. Primero se considerará el caso en que se pueden combinar cuatro celdas por la aplicación sucesiva del teorema $AB + AB' = A$. En la simplificación

$$S = ABCD + ABCD' + ABC'D + ABC'D' =$$
$$(ABCD + ABCD') + (ABC'D + ABC'D') = ABC + ABC' = AB$$

puede observarse como cuatro productos de cuatro variables pueden ser simplificados para obtener un solo término.

La característica principal de la combinación de cuatro términos es que dos de las variables son iguales en los cuatro productos y los números binarios correspondientes son idénticos a excepción de dos bits. El término resultante de la combinación de estos cuatro productos es igual a la adyacencia de dos

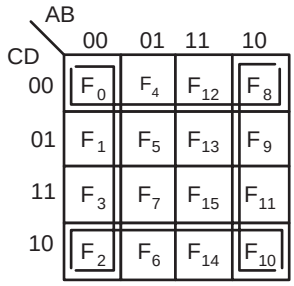
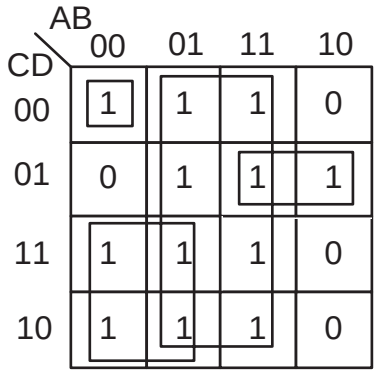


Figura 2.16: Adyacencia de celdas en los extremos de un mapa de cuatro variables.



$$S = A'B'C'D' + A'C + B + AC'D$$

Figura 2.17: Mapa de Karnaugh de cuatro variables con agrupamientos de 1, 2, 4 y 8 celdas.

celdas con la diferencia de que las dos variables en donde los bits de las celdas cambian deben de ser omitidas. En la figura 2.17 se muestra el mapa de Karnaugh con un agrupamiento de cuatro celdas y la expresión booleana correspondiente.

La última regla nos permite combinar ocho celdas, donde cada una de ellas tiene tres variables idénticas (ya sea negadas o no) en los ocho términos. En la figura 2.17 se muestra el mapa de Karnaugh con la combinación de ocho celdas y la expresión booleana resultante. La regla general establece que si se tienen 2^i términos producto con i variables idénticas entonces 2^i términos producto pueden ser combinados y las i variables que cambian pueden omitirse.

Durante el proceso de simplificación de expresiones booleanas mediante los mapas de Karnaugh, se tienen que ir encontrando y encerrando los conjuntos de celdas que puedan ser combinadas. Si uno de estos conjuntos se encuentra contenido en un conjunto más grande, entonces sólo se encerrará este último en el mapa. Los conjuntos encontrados junto con sus términos producto correspondientes son los *implicantes primos* que son exactamente los términos que se obtienen debido a la aplicación repetida del teorema $AB + AB' = A$.

2.5.3. Método de Quine-McCluskey

El método de mapas de Karnaugh puede ser muy útil cuando se tiene un problema con pocas variables de entrada, pero al ser un método gráfico la representación puede resultar sumamente complicada al tener un mayor número de variables [18, 11]. El método de Quine-McCluskey es un procedimiento sistematizado que permite la simplificación de expresiones booleanas con un mayor número de variables y que además puede ser realizado por una computadora personal. La idea básica de este método es reducir la expansión de minitérminos para obtener la suma mínima de productos llevando a cabo dos pasos principales:

1. Eliminar la mayor cantidad de literales posibles aplicando el teorema $AB + AB' = A$. Los términos resultantes obtenidos son llamados *implicantes primos*.

2. Seleccionar el conjunto mínimo de implicantes primos mediante un diagrama. Cuando estos términos son sumados lógicamente se obtiene una expresión simplificada que contiene el menor número de literales.

Determinación de los implicantes primos

En la primera parte del método de Quine-McCluskey todos los implicantes primos son obtenidos mediante la combinación de todos los minitérminos. Los minitérminos son expresados como números binarios y combinados mediante $AB + AB' = A$. Para poder obtener todos los implicantes primos, deben compararse y de ser posible combinarse todos los posibles pares de minitérminos. Para reducir el número de comparaciones, los minitérminos son ordenados en grupos de acuerdo al número de 1's existentes en su representación binaria como muestra la tabla 2.11.

Grupo 0	0	0000	$\surd$
Grupo 1	1	0001	$\surd$
	2	0010	$\surd$
	8	1000	$\surd$
Grupo 2	5	0101	$\surd$
	6	0110	$\surd$
	9	1001	$\surd$
	10	1010	$\surd$
Grupo 3	7	0111	$\surd$
	14	1110	$\surd$

Tabla 2.11: Agrupación de términos para encontrar los implicantes primos de la función $S = \sum(0, 1, 2, 5, 6, 7, 8, 9, 10, 14)$.

Tomando en cuenta el ordenamiento de cada grupo de la tabla 2.11 y considerando que dos términos pueden ser combinados si difieren en un solo bit, las comparaciones entre términos de dos grupos no adyacentes son innecesarias. Por otro lado, las comparaciones entre términos del mismo grupo también pueden omitirse debido a que dos términos con igual número de 1's deben diferir en al menos dos variables.

El algoritmo comienza comparando todos los términos del grupo 0 contra todos los términos del grupo 1, generando una nueva tabla en donde se escriben los términos que fueron combinados y la representación binaria del término simplificado que los representa además de que es necesario marcar los términos que pudieron ser combinados con algún otro en este paso. Después de esto se continúa con las comparaciones pero ya no del grupo 0 porque el único grupo adyacente es el grupo 1. Ahora se comparan los términos del grupo 2 contra los términos del grupo 1. Cuando todas las comparaciones necesarias han sido realizadas se obtiene la marcación de los términos que se combinaron (tabla 2.11) y una tabla como la que se muestra en la tabla 2.12 la cual se organiza también agrupando los términos con igual número de 1's en su representación binaria.

El siguiente paso consiste en generar una tercera tabla comparando y combinando los términos de grupos adyacentes pero que además tengan los guiones de su representación binaria en la misma posición, que es equivante a $A'B'C + A'B'C' = A'B'$. Los resultados de la compración y la combinación de términos para este paso son mostrados en la tabla 2.13 y la marcación de

Grupo 0	0,1	000-	$\surd$
	0,2	00-0	$\surd$
	0,8	-000	$\surd$
Grupo 1	1,5	0-01	
	1,9	-001	$\surd$
	2,6	0-10	$\surd$
	2,10	-010	$\surd$
	8,9	100-	$\surd$
	8,10	10-0	$\surd$
Grupo 2	5,7	01-1	
	6,7	011-	
	6,14	-110	$\surd$
	10,14	1-10	$\surd$

Tabla 2.12: Resultado de la comparación y combinación de términos en grupos adyacentes de la tabla 2.12.

los términos que se combinaron se ilustra en la tabla 2.12.

Grupo 0	0,1,8,9	-00-
	0,2,8,10	-0-0
	0,8,1,9	-00-
	0,8,2,10	-0-0
Grupo 1	2,6,10,14	–10
	2,10,6,14	–10

Tabla 2.13: Resultado de la comparación y combinación de términos en grupos adyacentes de la tabla 2.12.

En la tabla 2.13 puede verse que existen términos repetidos los cuales no son necesarios y pueden ser eliminados. Después de haber obtenido esta tabla el procedimiento se detiene ya que no es posible llevar a cabo la combinación de más términos. Los términos que no fueron marcados debido a que no pudieron combinarse con ningún otro término son los llamados *implicantes primos*. Ya que cada minitérmino ha sido incluido en al menos uno de los implicantes primos la función es precisamente igual a la suma de estos, es decir $S = A'C'D + A'BD + A'BC + B'C' + B'D' + CD'$.

Una expresión de suma de productos mínima consiste de la suma de algunos (pero no necesariamente todos) los implicantes primos de la función, es decir que una expresión que contenga algún término que no es un implicante primo, no puede ser mínima. Esto se puede comprobar ya que un término que no es un implicante primo no contiene el mínimo número de literales porque puede aún ser combinado con algún minitérmino para formar un implicante primo con menor número de literales. Cualquier expresión de suma de productos con términos que no son implicantes primos puede ser reemplazada por su equivalente con términos implicantes primos que reducirá el número de literales y por tanto simplificará la expresión [18].

Diagrama de implicantes primos

La segunda parte del método de Quine-McCluskey selecciona el conjunto mínimo de implicantes primos para la función booleana mediante un diagrama en donde los implicantes primos se listan hacia abajo y hacia la derecha se listan los minitérminos de la función. Un implicante primo es igual a la suma de minitérminos y por tanto se dice que los *cubre*. En el diagrama se coloca un $*$ en la intersección de implicante primo y minitérmino cuando este último es cubierto, como muestra la tabla 2.14.

		0	1	2	5	6	7	8	9	10	14
(0,1,8,9)	B'C'	*	*					*	⊛		
(0,2,8,10)	B'D'	*		*				*		*	
(2,6,10,14)	CD'			*		*				*	⊛
(1,5)	A'C'D	*			*						
(5,7)	A'BD				*		*				
(6,7)	A'BC					*	*				

Tabla 2.14: Diagrama de implicantes primos de las tablas 2.11, 2.12 y 2.13

Si un minitérmino es cubierto sólo por un implicante primo, éste es llamado *implicante primo esencial* y debe se incluído en la suma mínima de productos. Los implicantes primos esenciales son fáciles de identificar en el diagrama porque solamente contienen un $*$ en la columna correspondiente, en la tabla 2.14 las columnas de los minitérminos 9 y 14 contienen un solo $*$ por lo que $B'C'$ y CD' son esenciales.

Cada vez que un implicante primo es seleccionado para su inclusión en la suma mínima toda la fila debe de ser marcada como ya incluída. Después de hacer esto, la columna de los minitérminos ya incluídos también debe marcarse para que ya no sea agregada a la suma mínima. En la tabla 2.15 se muestra el resultado de marcar los minitérminos que son incluídos al agregar los implicantes primos esenciales a la suma mínima.

		0	1	2	5	6	7	8	9	10	14
(0,1,8,9)	B'C'	[*]	[*]					[*]	⊛		
(0,2,8,10)	B'D'	[*]		[*]				[*]		[*]	
(2,6,10,14)	CD'			[*]		[*]				[*]	⊛
(1,5)	A'C'D	[*]			*						
(5,7)	A'BD				*		*				
(6,7)	A'BC					[*]	*				

Tabla 2.15: Resultado de la inclusión de los implicantes primos esenciales en la suma mínima.

El conjunto mínimo de implicantes primos debe seleccionarse de manera que todos los minitérminos se encuentren incluídos [18]. En la tabla 2.15 aún faltan los minitérminos 5 y 7 por ser cubiertos. Sin embargo, si ahora se incluye en la suma mínima el implicante primo $A'BD$ todos los minitérminos se encontrarán cubiertos. El conjunto de implicantes primos mínimos es entonces: $S = B'C' + CD' + A'BD$.

2.5.4. Técnicas evolutivas

Las técnicas evolutivas son herramientas poderosas que han sido utilizadas para resolver una gran variedad de problemas de optimización y que han dado muy buenos resultados en ciertos dominios. El uso de técnicas evolutivas para el diseño de circuitos electrónicos es el llamado *hardware evolutivo* que se encuentra clasificado en dos grandes grupos: *evolución intrínseca* que diseña y prueba los circuitos en hardware y *evolución extrínseca* que hace una simulación del hardware en software para llevar a cabo el diseño de los circuitos.

Dentro de la evolución extrínseca se ha utilizado diversas técnicas para el diseño de circuitos lógicos combinatorios como: la programación genética

[33, 24, 4], la colonia de hormigas [29], los algoritmos genéticos [7], enfoques multiobjetivo [20, 9] e incluso el propio algoritmo de optimización mediante cúmulos de partículas [15, 10]. En esta sección se explicarán dos de estos algoritmos: el algoritmo genético de cardinalidad N (NGA) y el algoritmo genético multiobjetivo (MGA).

Algoritmo genético de cardinalidad N (NGA)

El algoritmo genético de cardinalidad N (NGA) es una propuesta basada en un algoritmo génetico con representación entera para automatizar el diseño de circuitos lógicos combinatorios y en donde se intenta minimizar el número total de compuertas utilizadas [6]. Un algoritmo genético para un problema en particular debe tener los siguientes componentes:

1. Una representación de las posibles soluciones al problema.
2. Una forma de poder generar una población inicial de posibles soluciones.
3. Una función de evaluación que pueda calificar a las soluciones de acuerdo a su "aptitud".
4. Operadores genéticos que alteren la composición de los hijos.
5. Valores para los distintos parámetros usados por el algoritmo genético (tamaño de población, probabilidad de aplicación de los operadores genéticos, etc.)

El NGA utiliza una representación de matriz (similar a la usada en este trabajo de tesis) para codificar las posibles soluciones encontradas en el proceso de diseño del circuito. Se puede decir que cualquier circuito puede ser representado como un arreglo bidimensional de compuertas $S_{i,j}$, donde j indica el nivel de la compuerta, de manera que las compuertas cercanas a las entradas tiene valores pequeños de j. En cada celda de la matriz está codificada una tercia en donde los dos primeros elementos se refieren a cada una de las entradas usadas y el tercero a la compuerta (ver figura 4.1). A pesar de que se ha probado que la representación binaria provee el máximo número de esquemas[1] [16], algunos alfabetos de mayor cardinalidad han dado mejores

[1] Un esquema es un patrón de valores que se presenta en las cadenas cromosómicas.

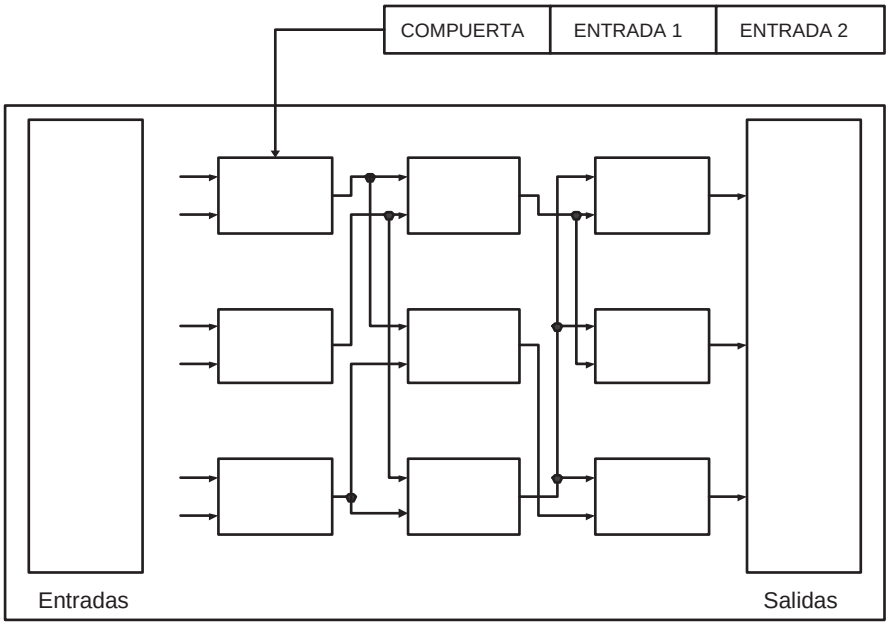


Figura 2.18: Representación de matriz para el diseño de circuitos lógicos.

resultados en un menor tiempo que su contraparte binaria. Por ello, para el NGA se decidió experimentar con un alfabeto de cardinalidad n, donde n toma normalmente un valor igual al número de renglones de la matriz que esta codificando las soluciones. El cálculo de la aptitud de un individuo x se hizo de acuerdo a la siguiente fórmula:

$$aptitud(x) = \begin{cases} \sum_{j=1}^{p} f_j(x) & \text{si x no es factible} \\ \sum_{j=1}^{p} f_j(x) + w(x) & \text{de otra forma} \end{cases} \tag{2.3}$$

donde p es el número de entradas a la tabla de verdad, $f_j(x)$ es el valor del número de salidas correctas (es decir, el número de salidas producidas por el individuo cuyo valor es igual al valor de la salida de la tabla de verdad de entrada) y $w(x)$ devuelve el número de WIREs[2] presentes en el circuito x, el cual sólo es agregado al valor de aptitud cuando el circuito es factible. Para la representación usada en el NGA con una cardinalidad n y una longitud del cromosoma de tamaño l, el tamaño del espacio de búsqueda intrínseco es n^l. La cardinalidad y la longitud de la cadena depende del tamaño de la

[2]WIRE es un buffer o reforzador de corriente que representa una ausencia de compuerta y que es utilizado para llevar a cabo la simplificación de los circuitos.

matriz usado para resolver el circuito. Los operadores genéticos usados en el NGA fueron un operador de *cruza* de dos puntos y un operador de *mutación* uniforme convencional.

Algoritmo genético multiobjetivo (MGA)

El MGA es una propuesta que utiliza una técnica de optimización evolutiva multiobjetivo para el diseño de circuitos lógicos combinatorios a nivel de compuertas [9], que reduce el número de evaluaciones necesarias de la función objetivo para encontrar el diseño óptimo. La idea principal del MGA es usar una propuesta basada en población de una técnica de optimización multiobjetivo similar a "Vector Evaluated Genetic Algorithm" (VEGA) [35] junto con una representación de matriz igual a la usada por el NGA. La optimización multiobjetivo puede definirse como:

> El problema de encontrar el vector de variables de decisión que satisface las restricciones y optimiza un vector de funciones cuyos elementos representan la función objetivo. Estas funciones forman una descripción matemática de los criterios de desempeño los cuales estan normalmente en conflicto entre sí. Por tanto, el término "optimización" significa encontrar una solución que obtenga los mejores valores posibles para todas las funciones objetivo y que sean aceptables para el diseñador.

Se puede decir que el problema de diseño de circuitos lógicos combinatorios es un problema con m restricciones iguales, donde m es el número de valores individuales de la tabla de verdad que se desean satisfacer. En cada generación, el MGA divide la población en $m+1$ subpoblaciones, donde cada una de ellas está a cargo de optimizar una restricción del problema (en este caso, las salidas del circuito). Sin embargo, el principal objetivo de cada subpoblación es tratar de igualar el valor de sus salidas con el de la tabla de verdad de entrada.

La función objetivo también es igual a la del NGA en donde se incrementa el valor de la función objetivo en uno por cada igualdad entre la salida del circuito y la salida de la tabla de verdad y si el circuito es factible (es decir, todas las salidas del circuito son iguales a las salidas de la tabla de verdad) se agrega uno por cada WIRE presente en la matriz que codifica al circuito. En el MGA una fracción de la población será seleccionada usando la

función objetivo como su aptitud (es decir, intentará maximizar la igualdad entre todas las salidas del circuito y las salidas de la tabla de verdad); otra fracción usará la igualdad entre la primera salida del circuito y de la tabla de verdad como su aptitud. La aptitud de cada subpoblación será calculada entonces mediante el siguiente esquema:

$$\begin{array}{lll} \text{si } o_j(\vec{x}) \neq t_j & \text{entonces} & \text{aptitud}(\vec{x})=0 \\ \text{si no, si } v \neq 0 & \text{entonces} & \text{aptitud}(\vec{x}) = -v \\ \text{si no,} & & \text{aptitud}(\vec{x}) = f(\vec{c}) \end{array}$$

donde o_j se refiere al valor de salida j del circuito codificado $\vec{x}$; t_j es el valor especificado por la salida j en la tabla de verdad; y v es el número de salidas del circuito $\vec{x}$ que no son iguales a las salidas de la tabla de verdad. Finalmente, $f(\vec{x})$ es la función de aptitud descrita en la ecuación 2.3. La selección en el MGA obedece distintas reglas dentro de cada subpoblación. Sin embargo, las cruza y la mutación son aplicadas a la población entera intentando llevar a cabo la recombinación del material cromosómico correspondiente a distintos circuitos parcialemente funcionales. El algoritmo general del MGA se muestra a continuación:

1. Generar una población aleatoria de tamaño P.
2. Dividir la población en m+1 subpoblaciones (m = número de salidas de la tabla de verdad).
3. Calcular los valores de aptitud de acuerdo a los objetivos de cada individuo dentro de cada subpoblación.
   - Si la salida objetivo no es igual, la aptitud es cero.
   - Si la salida objetivo es igualada, pero el circuito no es funcional, la aptitud es igual al número de salidas no igualadas multiplicado por (-1).
   - Si la salida objetivo es igual y el circuito es funcional, la aptitud está dada por la suma del número de salidas iguales y el número de WIREs en el circuito.
4. Hacer un barajeo de la población y seleccionar los padres de cada subpoblación basándose en el valor de aptitud previamente calculado de cada individuo.

5. Aplicar la cruza y la mutación a toda la población (sin importar la subpoblación) y generar la nueva población P'.

6. Si se ha alcanzado el criterio de convergencia, terminar.

7. Si no, regresar al paso 2.

En este algoritmo se puede observar que cada subpoblación coopera con las otras subpoblaciones que tienen dificultades de igualar su salida objetivo con la de la tabla de verdad. Si el circuito es factible, entonces todas las subpoblaciones tratarán de maximizar el número de WIREs en el circuito.

Capítulo 3

Conceptos básicos

3.1. Optimización

La optimización es una tarea que se presenta en diversas áreas tales como ciencias sociales, la economía y la ingeniería y básicamente implica el ajuste de una serie de parámetros o variables con la finalidad de conseguir que un sistema tenga un mejor desempeño. El primer paso en el proceso de optimización es identificar el objetivo, es decir, la métrica que determinará el desempeño del sistema y que estará sujeto a un cierto número de variables a través de las cuales se podrá llegar a un sistema óptimo. La identificación del objetivo y de las variables a la que está sujeto es el modelado del sistema. Es importante que al realizar el modelo éste se apegue a la realidad sin complicarlo demasiado, es decir llegar a un modelo equilibrado entre el sistema real y sus parámetros más relevantes. Si el modelo es muy simple entonces los resultados encontrados serán inútiles al intentar aplicarlos a la práctica mientras que por el contrario, si el modelo del sistema es demasiado complejo será muy difícil de resolver. Una vez determinado el modelo se puede aplicar algún método de optimización para obtener la solución buscada además de que se debe contar con alguna forma de evaluar la viabilidad de la solución propuesta y determinar su factibilidad.

3.1.1. Optimización con y sin restricciones

Los problemas que son resueltos generalmente por técnicas de optimización pueden ser clasificados en dos grandes grupos [32]:

1. Optimización sin restricciones: Son problemas que se presentan en aplicaciones prácticas en donde si las variables tienen algún tipo de restricción se considera que ésta no tiene efecto alguno sobre la solución encontrada.

2. Optimización con restricciones: Provienen de modelos que tienen restricciones explícitas sobre las variables. Estas restricciones pueden ser desde el establecimiento de los límites de los valores que puede tomar una variable o restricciones más complejas que involucran desigualdades no lineales.

3.1.2. Optimización global y local

La optimización local sólo trata de encontrar soluciones factibles que tengan un mejor valor en la evaluación de la función objetivo con respecto a las otras soluciones que se encuentran dentro de su mismo vecindario. Sin embargo, este tipo de optimización no siempre encuentra el mejor de todos estos "mínimos" que es precisamente la solución buscada por la optimización global [32]. A pesar de que la *solución global* es necesaria en muchas aplicaciones, frecuentemente es difícil de identificar y más aún de encontrar. Debido a la naturaleza de la búsqueda, los algoritmos que sólo tratan de encontrar óptimos locales son mucho más veloces que aquellos que buscan los óptimos globales.

3.1.3. Optimización estocástica y determinística

Los problemas de optimización estocástica se presentan cuando el modelo no puede ser descrito completamente porque depende de variables que varían o que son desconocidas al momento de la formulación del problema [32]. Para este tipo de problemas, se pueden predecir o estimar los valores desconocidos con un cierto grado de confianza de forma que se puedan producir soluciones que optimicen el desempeño esperado del modelo. Los modelos de optimización determinísticos, al contrario de los estocásticos, se encuentran completamente especificados por lo que no es necesario predecir ningún valor.

3.1.4. Algoritmos de optimización

La mayoría de los algoritmos de optimización son iterativos y siguen el mismo procedimiento, que comienza con un valor inicial estimado del valor óptimo y genera una secuencia de valores que van mejorando (con respecto a la evaluación de la función objetivo) conforme pasa el tiempo. La estrategia usada para mejorar estos valores es lo que distingue a los distintos algoritmos de optimización tales como los de búsqueda lineal, los métodos de gradiente conjugado, los de Newton, la programación lineal, etc.

Sin embargo, existen problemas que no pueden ser resueltos usando un algoritmo que tome un tiempo polinomial o incluso no se puede encontrar siquiera una solución eficiente. En este tipo de problemas en los que el mejor algoritmo conocido para resolverlos requiere de un tiempo exponencial se recurre a las *heurísticas* que pueden definirse como técnicas que buscan buenas soluciones a un costo computacional razonable sin garantizar la factibilidad u optimalidad de las mismas [34]. Algunos ejemplos de técnicas heurísticas son: *la búsqueda tabú*, *el recocido simulado* y el algoritmo de *escalando la colina*.

3.2. Computación evolutiva

La computación evolutiva se refiere a aquellas técnicas para resolver problemas que tienen una inspiración biológica. Estas técnicas se basan en el principio de que la evolución es un proceso de optimización en el cual se intenta mejorar las habilidades de los individuos y conseguir así una mejor adaptación a su entorno y por ende una mayor supervivencia. La computación evolutiva es una simulación del proceso de selección natural aplicado a un problema de búsqueda [12]. En términos generales, se puede decir que para poder simular el proceso evolutivo en una computadora se requiere:

- Codificar las estructuras que se replicarán.
- Operaciones que afecten a los individuos.
- Una función de aptitud.
- Un mecanismo de selección.

El modelo general de la computación evolutiva es una *población* de individuos llamados *cromosomas* en donde se codifican las caraterísticas de

las soluciones potenciales a un problema. Estas características son los *genes* del individuo o del cromosoma y su valor el *alelo*. En cada generación los individuos compiten por la supervivencia, pero aquellos con las mejores capacidades tienen una mayor oportunidad de reproducirse. La descendencia es generada mediante la combinación de partes de los cromosomas de los padres, es decir mediante la *cruza* aunque también puede presentarse algún tipo de *mutación* que altere el cromosoma del individuo. La medida de las habilidades y capacidades de un individuo se ve plasmada en su valor de *aptitud* que refleja cuál es el desempeño de la solución que representa dicho individuo para el problema propuesto [12].

Actualmente existen distintos tipos de algoritmos evolutivos. Sin embargo, frecuentemente se han distinguido tres paradigmas principales: la programación evolutiva, las estrategias evolutivas y los algoritmos genéticos, los cuales se originaron de manera muy independiente y con motivaciones distintas [5, 13].

3.2.1. Programación evolutiva

En esta técnica, el proceso evolutivo consiste en encontrar un conjunto de comportamientos óptimos a partir de un espacio de comportamientos observables, enfatizando los nexos de comportamiento entre padres e hijos. El operador de mutación es el único utilizado para generar la nueva población. El algoritmo general de la programación evolutiva se muestra en la figura 3.1.

```
Generar una población inicial aleatoria.
Repetir
        Evaluar la aptitud de cada individuo de la población.
        Mutar cada uno de los individuos para generar a los hijos.
        Seleccionar la nueva población.
Hasta que la condición de paro se alcance
```

Figura 3.1: Algoritmo general de la programación evolutiva.

3.2.2. Estrategias evolutivas

Están basadas en el concepto de *evolución de la evolución*. La evolución consiste en evolucionar tanto las características genéticas como los parámetros de la estrategia que modela el comportamiento del individuo en su ambiente. La mutación es aplicada sólo cuando este proceso genera individuos con una mejor aptitud. En esta técnica, los hijos pueden generarse a partir de más de dos padres. El algoritmo general de las estrategias evolutivas se muestra en la figura 3.2.

```
Generar una población inicial aleatoria.
Evaluar la aptitud de cada individuo de la población.
Repetir
        Aplicar operador de cruza.
        Aplicar operador de mutación.
        Evaluar la aptitud de cada uno de los hijos obtenidos.
        Seleccionar la nueva población.
Hasta que la condición de paro se alcance
```

Figura 3.2: Algoritmo general de las estrategias evolutivas.

3.2.3. Algoritmos genéticos

Los algoritmos genéticos enfatizan la importancia de la cruza sexual sobre la mutación; por ello el operador principal es el de cruza y el operador secundario es el de mutación (ver figura 3.3). La selección utilizada en esta técnica es probabilística.

3.3. Algoritmo de optimización mediante cúmulos de partículas

El algoritmo de optimización mediante cúmulos de partículas (*Particle swarm optimization* PSO) fue propuesto por Kennedy y Eberhart [21, 23] y tiene sus raíces en áreas como vida artificial, psicología social, ingeniería y ciencias de la computación. El PSO es un algoritmo estocástico de fácil implementación que no requiere información de gradiente derivada de la función

```
Generar una población inicial aleatoria.
Repetir
        Evaluar la aptitud de cada individuo.
        Seleccionar los padres.
        Llevar a cabo la recombinación de los padres para obtener a los hijos.
        Mutar a los hijos recién creados.
        Seleccionar la nueva población de individuos.
Hasta que la condición de paro se alcance
```

Figura 3.3: Algoritmo general de los algoritmo genéticos.

de error, lo que permite usarlo en funciones donde el gradiente no está disponible o cuando su cálculo requiere de mucho esfuerzo computacional [41].

Al igual que los tres principales paradigmas de la computación evolutiva, el algoritmo de PSO utiliza una población de soluciones potenciales que evolucionan a la solución óptima (o muy cercana a ésta) de un determinado problema. La principal diferencia del PSO y la computación evolutiva es un mecanismo que no parece tener un análogo en estas últimas, en el cual las partículas o individuos sobrevuelan a través del hiperespacio de búsqueda del problema [23]. Los algoritmos basados en una población toman en cuenta varias coordenadas del espacio de búsqueda al mismo tiempo. Un método de búsqueda poblacional puede definirse como sigue:

$$P' = m(f(P))$$

donde P es la *población*, que es un conjunto de posiciones en el espacio de búsqueda, f es la *función de aptitud*, que devuelve un vector de valores que indican la optimalidad de cada individuo de la población y m es una función de manipulación que genera una nueva población a partir de la población actual. En el PSO, la función de manipulación m está basada en el modelo de comportamiento de colonias de insectos [1].

El PSO tiene como idea principal simular el movimiento de un grupo de aves que buscan comida. En la simulación, el comportamiento de cada individuo está afectado por un factor individual y uno social. El factor individual se refiere a las decisiones que ha tomado el individuo y que han funcionado mejor (en términos de su aptitud), que afectarán las nuevas decisiones que pueda tomar. El factor social se refiere a las decisiones que han tomado el

resto de los individuos de la población (dentro de un cierto vecindario) que han funcionado mejor y que afectarán las nuevas decisiones tomadas por los individuos en el vecindario. Cada individuo o partícula contiene su posición actual en el espacio de búsqueda, su velocidad actual y la mejor posición encontrada por el individuo hasta este punto de la búsqueda (el factor individual).

El modelo de cúmulos de partículas está basado en una teoría sociocognitiva muy simple, que establece que el proceso de adaptación cultural está formado por una componente de alto nivel y otra de bajo nivel. La primera de estas componentes se manifiesta mediante la formación de patrones a través de los individuos y la habilidad de resolver problemas. La componente de bajo nivel se refiere al comportamiento de los individuos [23], que puede resumirse en tres principios: evaluar, comparar e imitar. Evaluar, es la tendencia de clasificar los estímulos como negativos o positivos, atractivos o repulsivos. Desde este punto de vista el aprendizaje puede definirse como el cambio que permite al organismo mejorar la evaluación de su ambiente. Comparar e imitar se refiere a la comparación del individuo con respecto a sus vecinos para imitar sólo a aquellos que considere mejor a él mismo.

En una población de partículas, cada uno de los individuos puede estar conectado a algún otro mediante una gran variedad de esquemas. Sin embargo, la mayoría de las implementaciones del algoritmo de PSO utilizan alguno de los dos modelos principales llamados *Gbest* y *Lbest*.

Modelo Gbest

Este modelo conecta conceptualmente a todos los individuos de la población (figura 3.4), que tiene como efecto que cada partícula estará influenciada por aquella partícula de toda la población que haya tenido el mejor desempeño. Esta partícula actuará como un atractor acercando a todas las partículas hacia ella por lo que eventualmente todas las partículas convergerán a la misma posición si este valor no es actualizado, logrando con ello una convergencia prematura.

Modelo Lbest

Este modelo crea un vecindario para cada individuo, que comprende a él mismo y a sus k vecinos (figura 3.5). El efecto que tiene es que el individuo estará influenciado por el individuo o partícula de mejor desempeño

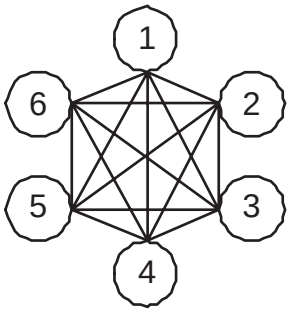


Figura 3.4: Modelo Gbest del PSO.

de una cierta porción de la población. Este esquema prevee la convergencia prematura, manteniendo múltiples atractores.

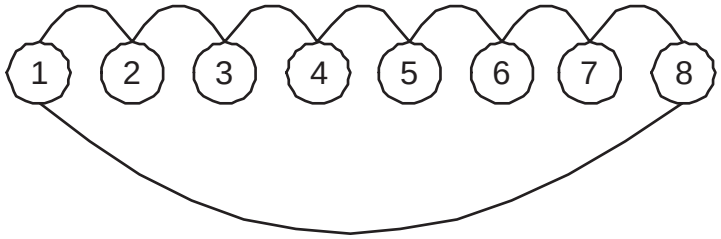


Figura 3.5: Modelo Lbest del PSO.

3.3.1. Algoritmo de PSO binario

En este modelo cada individuo de la población sólo tiene en mente la decisión binaria que tiene que tomar: si/no, falso/verdadero, etc. Cada individuo está rodeado por otros que también tienen que tomar su propia decisión. La información que tiene disponible para poder tomar la mejor decisión es su propia experiencia y la experiencia de sus vecinos. La probabilidad de que un individuo seleccione el "si" depende de qué tan exitoso haya resultado el "si" con respecto al "no" en el pasado. El individuo debe tomar una serie de decisiones de manera que todas ellas sean adecuadas en conjunto, por lo que debe de ser capaz de evaluar, comparar e imitar una cierta cantidad de decisiones binarias simultáneamente.

Matemáticamente, Kennedy y Eberhart [22] proponen un modelo donde la probabilidad de que un individuo decida si/no, falso/verdadero o alguna

otra decisión binaria es una función que depende de factores personales y sociales:

$$P[x_{id}(t) = 1] = f[x_{id}(t-1), v_{id}(t-1), p_{id}, p_{gd}]$$

donde:

- $P[x_{id}(t) = 1]$ es la probabilidad de que el individuo i seleccione 1 para el bit en la posición d de la cadena binaria.
- $x_{id}(t)$ es el estado actual del bit d de la cadena binaria.
- $v_{id}(t-1)$ es la medida de la predisposición individual o la probabilidad actual de seleccionar 1.
- p_{id} es el mejor estado encontrado para el bit d de la cadena binaria de acuerdo a la experiencia personal del individuo.
- p_{gd} es el mejor estado encontrado para el bit d de la cadena binaria encontrado por el mejor individuo del vecindario.

Si $v_{id}(t)$ es grande, el individuo tiene una mayor predisposición a seleccionar 1, mientras que valores pequeños favorecen al 0. Para ubicar el valor de este parámetro dentro del rango [0.0, 1.0], se utiliza la siguiente función:

$$s(v_{id}) = \frac{1}{1 + \exp(-v_{id})}$$

Para ajustar la disposición de cada individuo hacia el éxito personal y de la comunidad, se construye una fórmula para cada v_{id} en el momento actual, que será una función de la diferencia entre el estado actual del individuo y el mejor estado encontrado por él y sus vecinos. En cualquier situación no se puede determinar cuál será el factor de mayor influencia para tomar la decisión: el factor individual o el social, por lo cual cada uno de estos factores es multiplicado por un número aleatorio para que se vaya alternando el efecto de cada uno de ellos. La fórmula de decisión binaria es:

$$v_{id} = v_{id}(t-1) + \phi_1[p_{id} - x_{id}(t-1)] + \phi_2[p_{gd} - x_{id}(t-1)]$$

$$\text{Si } w_{id} < s[v_{id}(t)] \text{ entonces } x_{id}(t) = 1;\ \text{Si no } x_{id}(t) = 0$$

donde ϕ_1 y ϕ_2 representan números positivos con distribución uniforme con un límite superior definido y w_{id} es un vector de números aleatorios entre 0.0 y 1.0 con distribución uniforme. Cada una de estas fórmulas es aplicada repetidamente en cada una de las dimensiones de cada individuo, verificando cada vez si el valor actual de x_{id} resulta en una mejor evaluación que p_{id}, en cuyo caso se actualiza el valor. Algunas veces la decisión que tome el individuo estará basada en su experiencia personal y otras en su percepción de lo que los otros individuos creen. El algoritmo de PSO binario se muestra en la figura 3.6.

Repetir
  Para cada individuo i en la población
    Si aptitud($\vec{x_i}$) > aptitud($\vec{p_i}$)
      Para cada dimensión d del individuo hacer $p_{id} = x_{id}$

    $g = i$
    Para cada vecino j del individuo i
      Si aptitud($\vec{p_j}$) > aptitud($\vec{p_g}$) entonces $g = j$

    Para cada dimensión d del individuo
      $v_{id} = v_{id}(t-1) + \phi_1[p_{id} - x_{id}(t-1)] + \phi_2[p_{gd} - x_{id}(t-1)]$
      $v_{id} \in (-V_{max}, V_{max})$
      Si $w_{id} < s[v_{id}(t)]$ entonces $x_{id}(t) = 1$; Si no $x_{id}(t) = 0$

Hasta que la condición de paro se alcance

Figura 3.6: Versión binaria del algoritmo de optimización mediante cúmulos de partículas.

3.3.2. Algoritmo de PSO real

En esta versión del algoritmo de PSO, los individuos son vistos como puntos en el espacio donde el cambio a lo largo del tiempo es visto como un movimiento de puntos o partículas, de tal forma que el estar cerca en el espacio significa que los individuos son similares en las métricas relevantes. El aprendizaje sería un incremento o decremento en alguna dimensión, los cambios de actitud son vistos como movimientos entre el eje positivo y negativo mientras que los cambios de humor y emoción de los individuos pueden describirse conceptualmente en un sistema de coordenadas en el que existe un cierto número de partículas en movimiento. Estas partículas o puntos tenderán entonces a moverse hacia alguna otra partícula y a influenciarse entre sí mientras buscan la aprobación de sus vecinos.

El algoritmo de PSO real cambia la posición de la partícula mediante una velocidad $\vec{v_i}$ que es un vector de números que se agrega a la posición actual del individuo para moverlo desde un punto a otro en cada paso. El algoritmo de PSO recorre el espacio de búsqueda mediante la modificación del valor $\vec{v_i}$. La dirección del movimiento es una función de la posición actual de la partícula y su velocidad así como la mejor posición previa individual y la mejor posición previa en el vecindario:

$$\vec{x_i}(t) = f[\vec{x_i}(t-1), \vec{v_i}(t-1), \vec{p_i}, \vec{p_g}]$$

El cambio ahora está definido en términos de la velocidad y no de la probabilidad como en la versión binaria. Sin embargo, el cambio sigue siendo una función de la diferencia entre la mejor posición anterior y la actual y la diferencia entre la mejor posición en el vecindario anterior y la posición actual de la partícula. La fórmula para cambiar la velocidad es igual que la fórmula de la versión binaria con la única diferencia de que ahora las variables son continuas.

$$\vec{v_i}(t) = \vec{v_i}(t-1) + \phi_1[\vec{p_i}(t) - \vec{x_i}(t-1)] + \phi_2[\vec{p_g}(t) - \vec{x_i}(t-1)]$$

$$\vec{x_i}(t) = \vec{x_i}(t-1) + \vec{v_i}(t)$$

El algoritmo de PSO en el espacio de los números reales es casi idéntico a la versión binaria, excepto que $\vec{v_i}$ define incrementos en el movimiento en lugar de una probabilidad. En la figura 3.7 se muestra la versión real del algoritmo de PSO.

Repetir
  Para cada individuo i en la población
    Si aptitud($\vec{x_i}$) > aptitud($\vec{p_i}$)
      Para cada dimensión d del individuo hacer $p_{id} = x_{id}$

    $g = i$
    Para cada vecino j del individuo i
      Si aptitud($\vec{p_j}$) > aptitud($\vec{p_g}$) entonces $g = j$

    Para cada dimensión d del individuo
      $v_{id}(t) = v_{id}(t-1) + \phi_1[p_{id}(t) - x_{id}(t-1)] + \phi_2[p_{gd}(t) - x_{id}(t-1)]$
      $v_{id} \in (-V_{max}, V_{max})$
      $x_{id}(t) = x_{id}(t-1) + v_{id}(t)$

Hasta que la condición de paro se alcance

Figura 3.7: Versión real del algoritmo de optimización mediante cúmulos de partículas.

3.3.3. Parámetros del PSO

En el algoritmo de PSO existen una gran variedad de parámetros que pueden ser ajustados para modificar la manera en que lleva a cabo la búsqueda el algoritmo. Los dos parámetros más importantes son V_{max} y ϕ los cuales son establecidos al inicio del algoritmo y se utilizan a lo largo de toda la búsqueda. La manipulación de estos parámetros pueden causar cambios abruptos en el comportamiento del sistema.

V_{max}

Este parámetro es utilizado para evitar que la trayectoria de la partícula se salga de control y que se expanda en ciclos cada vez más amplios en el

espacio del problema hasta que eventualmente tienda al infinito. La figura 3.8 es una gráfica de la trayectoria de una partícula simplificada que tiene una sola dimensión. En el eje horizontal se muestran las iteraciones del algoritmo y en el eje vertical el valor de la partícula. En esta gráfica puede observarse el efecto de la eliminación del parámetro V_{max}. Para realizar esta gráfica se utilizaron los valores de los extremos, por ello en las primeras iteraciones los ciclos parecen una línea recta. Sin embargo, la trayectoria del valor de la partícula crece cada vez más conforme pasa el tiempo hasta que al llegar a las 150 iteraciones el valor se encuentra en el rango de $\pm 10^9$ [23].

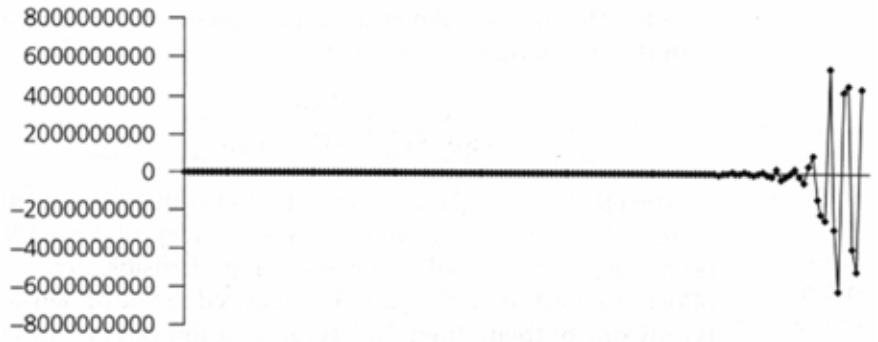


Figura 3.8: Efecto de la eliminación del parámetro V_{max} en el algoritmo de PSO [23].

ϕ

El parámetro de control ϕ también es llamado la "constante de aceleración" y determina el tipo de trayectoria que tomarán las partículas. Si ϕ tiene un valor muy pequeño, la trayectoria de la partícula crece y cae lentamente a través del tiempo, mientras que para valores grandes (alrededor de 10) pareciera que la trayectoria es aleatoria. El incremento de ϕ resulta en un incremento de la frecuencia de la trayectoria seguida por la partícula (figura 3.9) mientras que la amplitud está limitada por V_{max} [23].

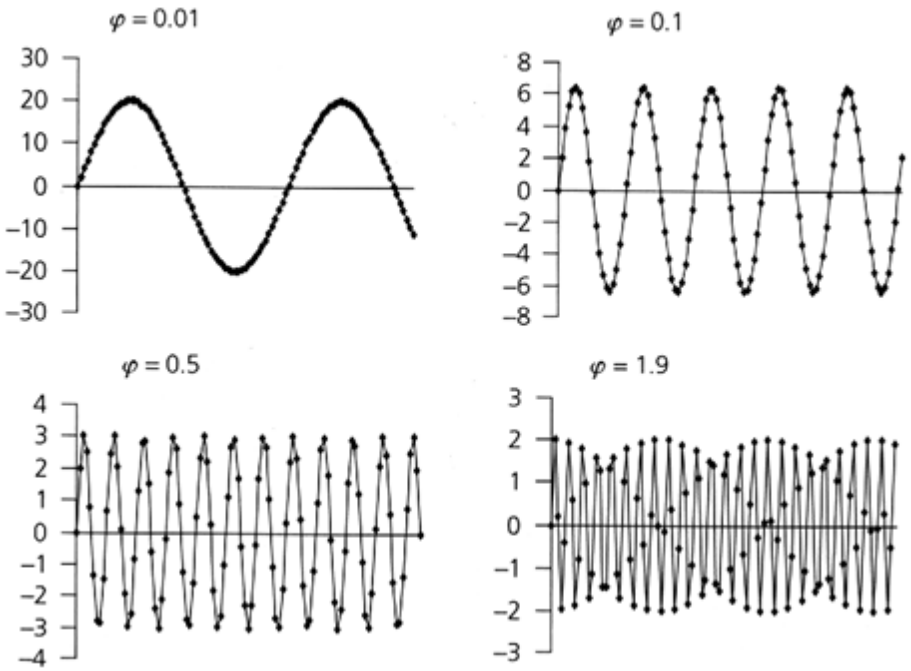


Figura 3.9: Efecto de la variación del parámetro ϕ en el algoritmo de PSO [23].

Capítulo 4

Descripción de la técnica

En este capítulo se presenta la forma en que se atacó el problema del diseño de circuitos lógicos combinatorios. Primero se explica la representación utilizada incluyendo la forma de evaluación y de presentación de resultados. Después se describe el algoritmo de optimización mediante cúmulos de partículas y las modificaciones que se le efectuaron para que pudiera utilizarse en el diseño de circuitos lógicos. Finalmente se habla de la función de aptitud adoptada en nuestro algoritmo.

4.1. Representación de los circuitos lógicos

El principal problema que se pretende resolver en este trabajo es el diseño y la posterior optimización de circuitos lógicos combinatorios a nivel de compuertas, teniendo como entrada una tabla de verdad en la que se indican las salidas deseadas dado un conjunto de entradas posibles.

Debido a que se conocen los valores de entrada y de salida, lo que se busca es la configuración de compuertas mínima (en términos de alguna métrica) que genere las salidas indicadas en la tabla de verdad correspondiente.

Para poder diseñar y optimizar estos circuitos es necesario utilizar una representación de las soluciones que ayude a que el tiempo necesario para la decodificación no sea muy grande. Es decir, que se debe contar con una representación que permita una fácil interpretación de manera que no se tenga que invertir demasiado tiempo en realizar el mapeo de la representación a los valores usados para calcular la aptitud, ya que durante el proceso se va a realizar una gran cantidad de evaluaciones de las posibles soluciones

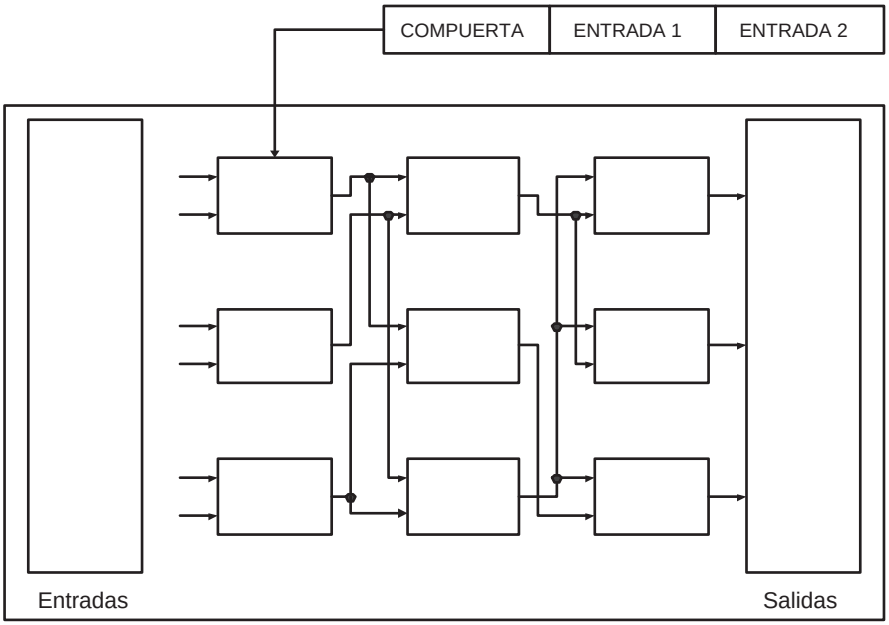


Figura 4.1: Representación de matriz para el diseño de circuitos lógicos.

que se vayan encontrando y que están contenidas dentro de cada una de las partículas de la población.

Es importante tener en cuenta que una vez decodificada la solución es necesario llevar a cabo su evaluación y obtener así una tabla de verdad como la que se da a la entrada para así comprobar la factibilidad de la solución. Una solución factible es la configuración de compuertas tal que produce la misma tabla de verdad que la tabla de entrada. Pueden existir una gran cantidad de soluciones factibles en el espacio de posibles circuitos existentes dentro de la representación adoptada, pero la configuración que se busca es la que tiene el menor número de compuertas.

Existen muchas formas para representar un circuito, sin embargo se optó por utilizar una representación de matriz como la de la figura 4.1, la cual fue propuesta por Louis [25, 26] y que ha sido usada también por otros autores como Miller [30, 31] y Coello [7, 6, 9].

En esta codificación cada celda de la matriz contiene la información referente a la compuerta y sus entradas. A pesar de que la representación permite que se utilicen compuertas de cualquier número de entradas, para fines de esta tesis se consideró sólo el caso particular de dos entradas, de manera que

cada celda de la matriz contiene una tercia, tal como se muestra en la figura 4.1.

Las compuertas que pueden estar codificadas en cada una de las tercias puede ser alguna de las siguientes: AND, OR, XOR, NOT o WIRE. Esta última en realidad no es una compuerta y no contribuye en la cuenta de compuertas presentes en la solución, sino que se trata de un buffer o reforzador de corriente que es necesario para poder llevar a cabo la optimización de las soluciones.

Las dos entradas de cada compuerta provienen únicamente de la columna previa a la columna donde se encuentra la celda. Cabe mencionar que la primera columna no tiene columna previa así que para ella las entradas de la compuerta son precisamente las variables de entrada del circuito.

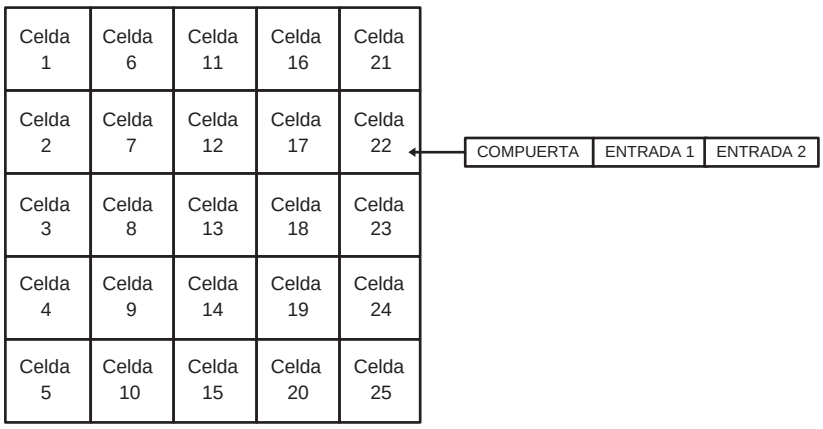


Figura 4.2: Matriz de 5×5 con un espacio de búsqueda de 5^{75}.

El espacio de búsqueda máximo debido a esta representación depende directamente del número de tipos de compuertas distintas que puedan ser utilizadas en el diseño del circuito y del tamaño de la matriz usada para la representación de las soluciones. Si tenemos una matriz de $n \times m$ entonces la longitud necesaria para representarla es de $3 \times n \times m$ ya que cada una de las celdas es una tercia de información. Si además tenemos k tipos de compuertas distintas entonces el espacio de búsqueda es de k^{3nm}. En la figura 4.2 se muestra una matriz de 5×5 con 5 tipos de compuertas distintas y con un espacio de búsqueda de tamaño $5^{5 \cdot 5 \cdot 3} = 5^{75} \approx 2.6 \text{ x } 10^{52}$.

4.1.1. Representación interna de las celdas

Cada celda de la matriz necesita codificar la información del tipo de compuerta y de sus entradas. Para esto es necesario adoptar una representación sobre la que operará la optimización mediante cúmulos de partículas. Existen varios tipos de representación que pueden utilizarse, como la representación real, binaria o entera (figura 4.3).

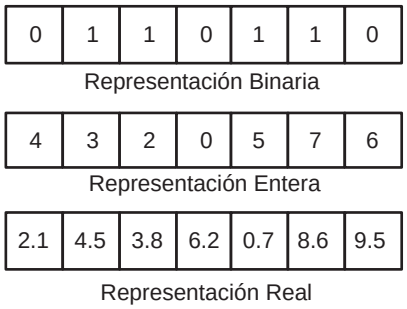


Figura 4.3: Distintos tipos de representación.

En cada una de las celdas, la primera parte de la tercia indica el tipo de compuerta y las dos partes restantes contienen sus entradas, las cuales provienen de la columna anterior a la columna donde se encuentra la celda.

La forma más sencilla de representar las compuertas y un identificador de variable o de alguna celda en la matriz es mediante números enteros. Por ejemplo, podemos asignar a las funciones booleanas AND, OR, XOR y NOT los valores de 0,1,2,3 respectivamente y el valor 4 a la compuerta de tipo WIRE. De igual forma, podemos asignar un número a cada celda de la matriz en cada columna comenzando desde arriba y también podemos asignar un identificador numérico a cada una de las variables de entrada.

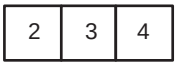


Figura 4.4: Una tercia de la matriz con representación entera.

Si tenemos la tercia que se muestra en la figura 4.4 sabemos que si se está examinando una celda que no se encuentra en la primera columna de la matriz entonces se trata de una compuerta XOR y que sus entradas son la celda 3 y la celda 4 de la columna anterior. En el caso de que la celda se

encuentre en la primera columna la compuerta XOR es la misma, sin embargo sus entradas son las variables con el identificador 3 y el identificador 4.

Para la representación binaria es necesario saber el número de bits que se requieren para representar cada uno de los elementos de un conjunto. Como tenemos 5 tipos de compuertas distintas entonces necesitamos $log_2 5 = 2.32$ bits. Como no es posible tener exactamente 2.32 bits utilizamos 3 los cuales nos proporcionarán $2^3 = 8$ identificadores distintos. Ya que tenemos más identificadores de los necesarios podemos aplicar una función de módulo para obtener siempre uno de los 5 elementos del conjunto de compuertas. En el caso de los valores de entrada a la compuerta, quizá se tenga un conjunto menor o mayor de celdas en cada columna o de identificadores de variables de entrada; sin embargo, se puede seguir el mismo procedimiento del conjunto de compuertas.

0	1	0	0	1	1	1	0	0

Figura 4.5: Una tercia de la matriz con representación binaria.

Si por ejemplo tuviéramos una celda como la de la figura 4.5 con representación binaria, su decodificación es igual a la de la figura 4.4, una compuerta XOR que tiene como entradas las celdas 3 y 4 o con las variables con identificadores 3 y 4.

En la representación real pueden existir muchas formas de asignar un identifcador único a cada uno de los elementos del conjunto (por ejemplo, un rango de valores). Sin embargo no es tan intuitiva como la representación entera o binaria; por ello se optó por utilizar estas dos últimas para el desarrollo de este trabajo.

4.1.2. Cardinalidad

Con una representación entera se puede tener exactamente el número de identificadores necesarios para representar los elementos de un conjunto, a diferencia de la representación binaria que tendrá más identificadores siempre que el número de elementos en el conjunto no sea una potencia de dos.

La cardinalidad se refiere a generar más identificadores de los necesarios en la representación entera para tener un rango de valores más amplio en cada posición de las tercias de las celdas, para luego aplicar una función módulo y obtener así algún elemento del conjunto que es un enfoque similar al usado en [6].

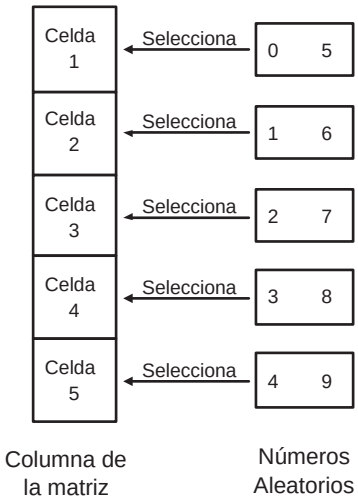


Figura 4.6: Uso de una cardinalidad de dos para la selección de celdas

Si por ejemplo tenemos 5 renglones en la matriz entonces se podrán seleccionar números aleatorios entre 0 y 4 para ser colocados en la posición correspondiente a alguna de las entradas de la compuerta dentro de las tercias. Sin embargo, como se muestra en la figura 4.6, si se usa una cardinalidad de 2, los números que podrían generarse serían entre 0 y 9 para aplicar después el modulo y poder seleccionar alguno de los renglones de la matriz en una determinada columna.

Esto ayuda a que se haga una mejor distribución de los valores seleccionados para cada una de las posiciones dentro de las tercias, pues los valores en los extremos del rango de los números no son seleccionados tan frecuentemente como los internos. Pero si el rango crece entonces los valores en los extremos también estarán representados en alguna de las posiciones internas y tendrán la misma oportunidad de ser seleccionados que el resto de los números en el rango.

4.2. Evaluación

La evaluación de un circuito codificado en la matriz se refiere a la obtención de la tabla de verdad que produce dada su configuración de compuertas. La tabla de verdad es importante debido a que es la forma en que está expresada la entrada al programa (es decir la salida el circuito del cual se está buscando el diseño óptimo y del que sólo se conocen sus salidas mas no la configuración de compuertas) y por ello la única manera en que podemos saber si la salida de la solución encontrada genera los valores deseados en algunas o en todas las combinaciones de las variables de entrada. Si la tabla de verdad del circuito es idéntica a la tabla de verdad de entrada entonces decimos que el circuito es factible.

Para poder obtener la tabla de verdad de todas las salidas del circuito, lo que se hace es evaluar cada celda de la matriz para cada una de las combinaciones de entrada. Si por ejemplo, se tienen 3 variables se tendrán $2^3 = 8$ diferentes combinaciones (ya que cada variable solamente puede tomar el valor 0 ó 1) y 8 evaluaciones de cada una de las celdas.

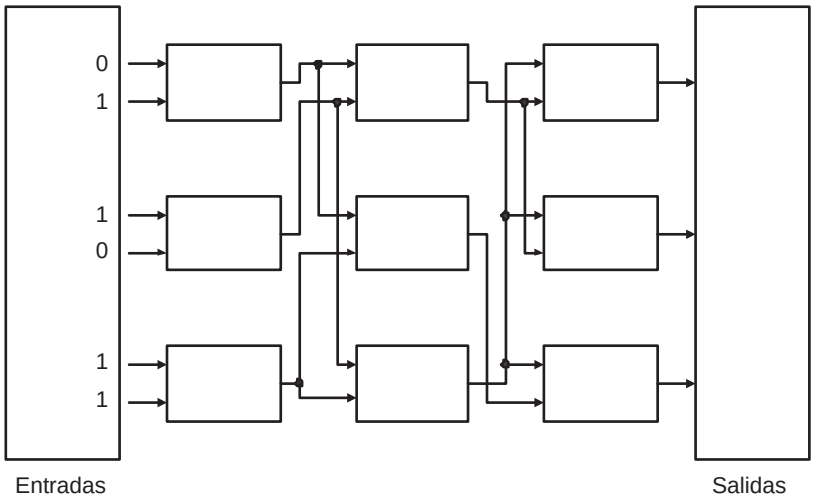


Figura 4.7: Entradas a las celdas de la matriz.

El proceso de evaluación comienza por la columna que está más a la izquierda de la matriz. Como se ve en la figura 4.7, las entradas de las compuertas de cada celda de esta columna son los valores de las variables de entrada de cada una de las combinaciones.

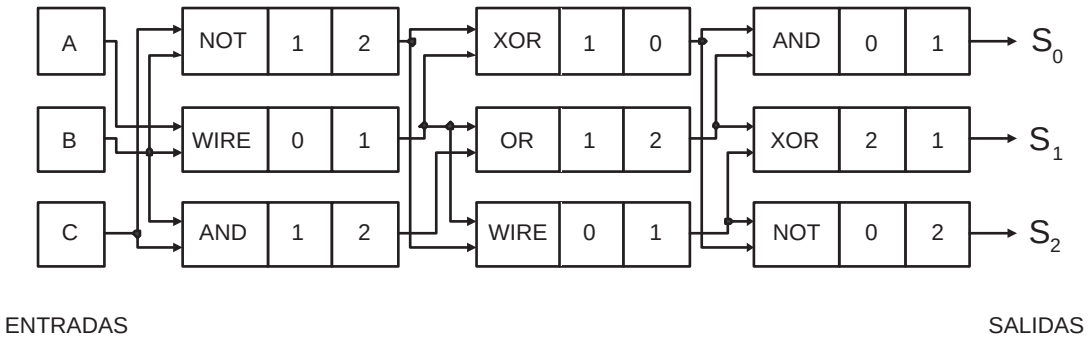


Figura 4.8: Matriz de ejemplo para la obtención de la expresión booleana.

Sin embargo, como también muestra la figura 4.7 para la siguiente columna, los valores de entrada de las compuertas son los valores de salida de las celdas de la columna previa. La evaluación continúa hasta que se llega a la última columna, la cual contiene en cada celda la salida final del circuito codificado. La celda superior derecha es la primera salida y la que está debajo de ella en esa misma columna es la segunda salida y así sucesivamente, por lo cual en esta columna se obtiene el valor a la salida del circuito de la matriz para la combinación de valores de entrada.

Durante el proceso de evaluación se verifica la cantidad de salidas de la tabla de verdad de la solución que son iguales a las salidas de la tabla de verdad de entrada para poder así asignar un valor de aptitud a la solución, el cual se explica más adelante.

4.3. Obtención de expresiones booleanas

La salida del programa es el diseño de un circuito lógico, el cual está dado por una expresión booleana. En el caso de las expresiones booleanas en notación prefija, es necesario comenzar el proceso de obtención en la columna que está más a la derecha de la matriz, que es la que contiene las salidas del circuito.

Como se mencionó antes, las dos entradas de cada compuerta son celdas de la columna previa. Para obtener las entradas de las compuertas se necesita siempre consultar la columna anterior a la celda que se esté verificando. Nótese que la columna de la extrema izquierda no tendrá una columna previa. Por ello, las entradas de las compuertas de esta primera columna son

las variables de entrada del circuito que estarán expresadas mediante letras mayúsculas como se muestra en la figura 4.8.

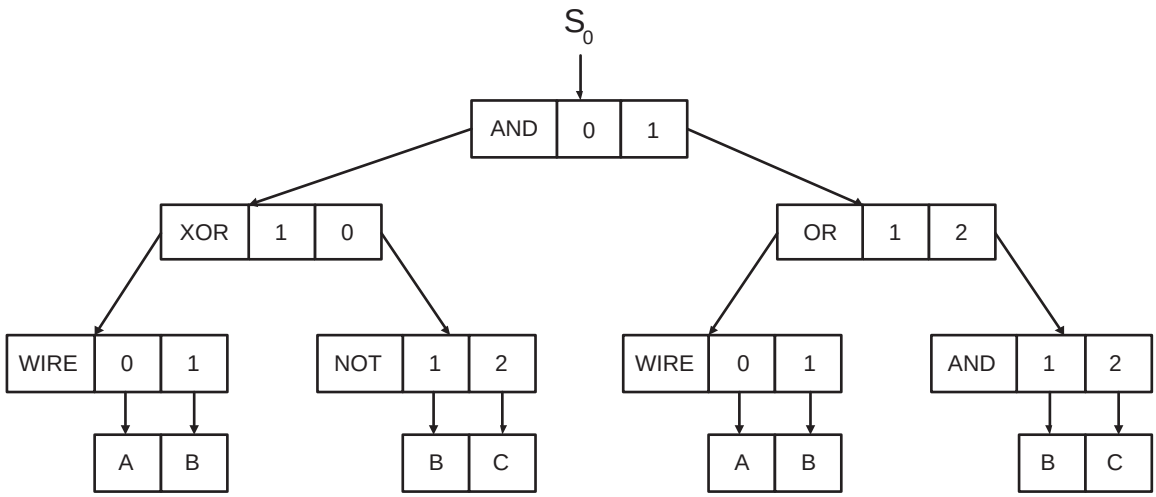


Figura 4.9: Arbol de derivación para la salida S_0de la expresión booleana en notación prefija de la matriz mostrada en la figura 4.8.

Para obtener la expresión booleana de alguna de las salidas del circuito codificado, es necesario identificar la salida deseada en la columna de la extrema derecha de la matriz y a partir de ahí se comienza la construcción (figura 4.9).

```
1. (AND 0 1)
2. (AND (XOR 1 0) (OR 1 2))
3. (AND (XOR (WIRE 0 1) (NOT 1 2)) (OR (WIRE 0 1) (AND 1 2)))
4. (AND (XOR (WIRE A B) (NOT B C)) (OR (WIRE A B) (AND B C)))
```

Figura 4.10: Derivación completa de la expresión booleana para la salida S_0 a partir de la matriz de la figura 4.8.

Primero se tiene que colocar la función booleana (o compuerta) de la celda y a continuación el valor de sus entradas. Después se necesitan decodificar las entradas de esta primera función booleana, para lo cual se toma la columna anterior y se colocan las siguientes funciones en su posición correspondiente en la expresión. Para decodificar las entradas de estas funciones booleanas

de segundo nivel se realiza el mismo procedimiento hasta que se llega a la primera columna. Por último, en la figura 4.10 se muestra todo el proceso hasta la última decodificación, que es la traducción de las entradas de las celdas de la primera columna a variables de entrada del circuito, completando así la expresión booleana en notación prefija.

4.4. Algoritmo de optimización mediante cúmulos de partículas para el diseño de circuitos lógicos combinatorios

El algoritmo de optimización mediante cúmulos de partículas introducido en la sección 3.3 es una técnica de optimización numérica de funciones no lineales que ha tenido bastante éxito gracias a que tiene un bajo costo computacional, una fácil implementación y un excelente desempeño [23], siendo éstas las principales motivaciones de usar este algoritmo para el diseño de circuitos lógicos combinatorios.

Tradicionalmente el algoritmo de PSO ha sido utilizado para representación binaria [22] y real [23], teniendo un gran éxito con esta última. A pesar de ello, en este trabajo no se utilizó la representación real sino que se hicieron algunas modificaciones al algoritmo para que utilizara representación entera.

Existen varias maneras de modelar el diseño de circuitos con esta representación, sin embargo sólo se implementaron dos de ellas. La primera fue propuesta por Eberhart y Shi [17] para optimización combinatoria y la denominaremos *representación entera A*. La segunda representación es una de las contribuciones de este trabajo de tesis y será denominada *representación entera B*.

En la figura 4.11 puede verse el algoritmo general implementado. El PSO tiene como idea principal simular el movimiento de un grupo de aves que buscan comida. En la simulación, el comportamiento de cada individuo está afectado por un factor individual y uno social. El factor individual *Pbest* se refiere a las decisiones que ha tomado el individuo y que han funcionado mejor (en términos de su aptitud) y las afectarán las nuevas decisiones que pueda tomar. El factor social *Nbest* se refiere a las decisiones que han tomado el resto

Inicializar la población P de partículas aleatoriamente.
Repetir {
    Para cada partícula i en la población P {
        Calcular el valor de aptitud de la partícula $P[i]$
        Si el valor de aptitud de $P[i]$ es mejor que el valor de aptitud de la mejor partícula encontrada en su historia (*Pbest[i]*)
            Actualizar el valor de *Pbest[i]* al valor de $P[i]$
    }

    Para cada partícula i en la población P {
        Seleccionar la partícula con el mejor valor de aptitud dentro del vecindario topológico de $P[i]$ y actualizar el valor de *Nbest[i]*
    }

    Para cada partícula i en la población P {
        Calcular la nueva velocidad para cada dimensión de la partícula
            $\vec{V}[i]_{nueva} = \vec{V}[i]_{anterior} + \phi_1(\vec{P}best[i] - \vec{P}[i]) + \phi_2(\vec{N}best[i] - \vec{P}[i])$
        Actualizar la posición de la partícula $P[i]$
    }

    Aplicar la mutación uniforme de acuerdo al porcentaje proporcionado.
} Hasta que la condición de paro se alcance

Figura 4.11: Pseudocódigo del algoritmo de optimización mediante cúmulos de partículas.

de los individuos de la población (dentro de un cierto vecindario) que han funcionado mejor y que afectarán las nuevas decisiones tomadas por los individuos en el vecindario.

En el algoritmo de la figura 4.11 el paso que diferencía a la representación binaria, entera A y entera B es en el que se actualiza la posición. Este proceso realiza la prueba de cada una de las dimensiones de la partícula y lleva a cabo la actualización de su posición en caso de ser necesario. Para cada tipo de representación el procedimiento que se lleva a cabo es diferente y se explica a continuación:

- Representación binaria

  Si flip[s(V_d)] = 1
    Establecer el valor de la posición d al valor 1
  Si no
    Establecer el valor de la posición d al valor 0

- Representación entera A

  Si flip[sig(V_d)] = 1
    Establecer el valor de la posición d al valor de $Nbest$ en la misma posición

- Representación entera B

  Si flip[sig(V_d)] = 1
    Establecer el valor de la posición d al valor de $Nbest$ en la misma posición
  Si no
    Si flip[1 - sig(V_d)] = 1
      Establecer el valor de la posición d al valor de $Pbest$ en la misma posición

En los tres casos *flip*[] devuelve el valor 1 con una probabilidad igual a la que recibe como parámetro. La variable V_d se refiere a la velocidad de la partícula, que es la predisposición individual del alelo d para seleccionar una u otra de las opciones existentes (en este caso el valor 0 o 1) y que estará determinada por un valor de probabilidad en el rango [0.0,1.0]. Si V_d tiene un valor grande, el individuo tenderá a escoger el valor 1 mientras que cuando V_d tenga un valor pequeño se inclinará por el valor 0. La función de la ecuación 4.1 nos permite llevar a cabo la normalización de la variable V_d además de que a valores grandes de ω la función *sig* será cercana a 1, mientras que para valores pequeños estará cerca del 0 que es justamente el comportamiento deseado, es decir que para valores grandes de V_d se espera que haya una alta probabilidad de que sea seleccionado el valor 1 y que para valores pequeños de V_d la probabilidad de seleccionar el 1 sea mucho menor. La definición de *sig* es la siguiente:

$$\text{sig}(\omega) = \frac{1}{1 + \exp(-\omega)} \tag{4.1}$$

En la figura 4.12 se muestra de forma gráfica el resultado de la aplicación de la representación entera A y entera B a los valores de una partícula.

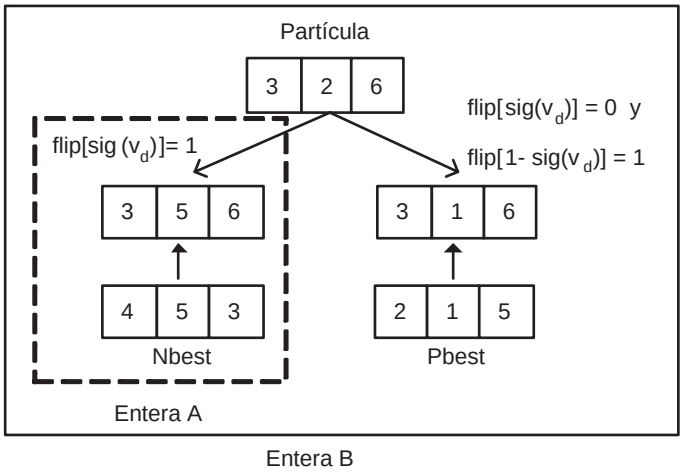


Figura 4.12: Ejemplo del funcionamiento de la representación entera A y entera B.

4.5. Mutación uniforme

La expresión principal de ajuste de la optimización mediante cúmulos de partículas puede ser vista como un tipo de mutación. Sin embargo, para el problema de diseño de circuitos se encontró que su poder exploratorio no era suficiente, por lo que se agregó un operador de mutación uniforme [14] con un cierto porcentaje.

El operador recorre todas las posiciones del alelo (en nuestro caso, cada uno de los valores de las tercias en cada una de las celdas de la matriz) y para cada una de ellas aplica una función que devuelva el valor "verdadero" con una probabilidad igual al porcentaje de mutación dado por el usuario. Si la función devuelve un valor "verdadero" entonces el valor de la posición es cambiado por algún otro dentro de su rango, como se ve en la figura 4.13.

En el caso de la representación entera, el operador de mutación es importante debido a que todas las soluciones tienen una mayor tendencia a igualar los valores de *pbest* y *nbest* y que de hecho no moverán sus valores cuando ya sean idénticas a alguna de ellas.

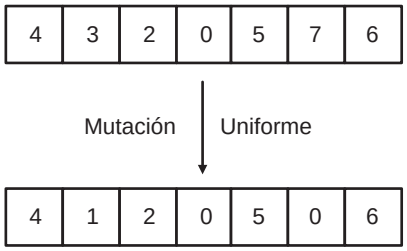


Figura 4.13: Ejemplo de mutación uniforme sobre una representación entera.

4.6. Función de aptitud

Para determinar que un circuito es de costo mínimo se pueden tomar en cuenta varios criterios como el número de literales, el número de símbolos presentes en la función booleana o el número de niveles del circuito. También, existen algunas restricciones físicas de los circuitos lógicos como el *fan-in* que especifica el número máximo de entradas a una compuerta, el *fan-out* que especifica la carga máxima a la salida y el *retardo de la propagación* que es el tiempo necesario para que el cambio de valor de una señal se propague de la entrada a la salida y que influye directamente en la velocidad de operación. Sin embargo, la complejdad de un circuito depende de su número de compuertas y la complejidad de una compuerta depende de su número de entradas de manera que si el diseño del circuito está expresado mediante una función booleana, al reducir el número de literales en dicha expresión se reducirá el número de entradas de cada compuerta y por ende el número total de compuertas necesarias en el circuito reduciendo con esto la complejidad.

Dadas las consideraciones anteriores se optó por intentar reducir el número de compuertas de un circuito válido para obtener el diseño óptimo. En [25] y [30] la función de aptitud sólo trata de maximizar el número de correspondencias entre la salida del circuito y la tabla de verdad de entrada, es decir que funciona en una sola etapa. Sin embargo, en esta propuesta se manejó el problema de diseño de circuitos como un problema de maximización en dos etapas, similar al adoptado por Coello en [6, 9].

En la primera etapa se trata de conseguir una solución factible mediante la maximización de la correspondencia entre la tabla de verdad de la solución y la tabla de verdad de entrada. Es decir que para cada elemento de la tabla de verdad de la solución que sea igual al mismo elemento en la tabla de verdad de entrada se aumentará en uno la aptitud de la partícula de manera

A	B	C	Entrada	Solución
0	0	0	1	1
0	0	1	1	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	0	1
1	1	1	0	0

Aptitud de la solución : 6

Figura 4.14: Aptitud de una solución no factible.

que mientras la solución no sea factible la aptitud será menor que el número total de combinaciones posibles de los valores de las variables de entrada (figura 4.14).

En la figura 4.15 se muestra la segunda etapa, que es la que intenta optimizar una solución factible. Debido a que es un problema de maximización, las soluciones factibles deben tener una mayor aptitud que las soluciones no factibles por lo que su aptitud estará dada por el número total de correspondencias entre la tabla de verdad de la solución y la de entrada (como son tablas del mismo tamaño este valor siempre será mayor que el valor de aptitud de una solución no factible) más una bonificación por cada WIRE presente en la matriz y por cada compuerta que esté en la matriz pero que no participe en la solución.

La maximización de WIREs en la matriz da como resultado una disminución en el número de compuertas necesarias para la solución ya que éstas no implementan una función booleana sino que simplemente son buffers o reforzadores de corriente.

Si por ejemplo tenemos una matriz de tamaño 4×4 la cual sabemos que contiene una solución factible para una determinada tabla de verdad de 3 entradas y 1 salida, además de que tiene 6 WIREs y 6 compuertas que no participan en la solución entonces podríamos decir que su aptitud sería 2^3 (combinaciones distintas de entrada) $\times$ 1 (salida) + 6 (WIREs) + 6 (compuertas que no participan en la solución)= 20. De igual forma si se quiere saber el número de compuertas necesarias para implementar una solución

A	B	C	Entrada	Solución
0	0	0	1	1
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

Aptitud de la solución = 20

8(correspondencias) + 6(WIREs) + 6(compuertas que no participan en la solución)

Figura 4.15: Aptitud de una solución factible.

dada su aptitud tendríamos que calcular primero el número de WIREs y de compuertas que no se usan en la solución que es 20 (aptitud) - 2^3 (combinaciones distintas de entrada) $\times$ 1 (salida) = 12 (WIREs + compuertas no usadas) y luego obtener el número de compuertas, restando este último número al total de celdas en la matriz es decir 16 (celdas) - 12 (WIREs + compuertas no usadas) = 4 (compuertas).

Es importante contabilizar sólo las celdas que participen en la solución y no todas las presentes en la matriz ya que para una determinada salida puede ocurrir que no sean necesarias todas las celdas para implementar la función booleana.

Capítulo 5

Circuitos de una salida

El desempeño del algoritmo de optimización mediante cúmulos de partículas para el diseño de circuitos lógicos combinatorios se evaluó mediante una serie de experimentos en donde el objetivo principal era evaluar la consistencia del algoritmo para producir circuitos factibles que minimizaran el número total de compuertas utilizadas.

Cada experimento consta de una tabla de verdad de algunas variables de entrada y una sola salida que describe el comportamiento de un circuito en su totalidad. La tabla de verdad es introducida al programa y se lleva a cabo un conjunto de corridas para analizar el comportamiento estadístico del algoritmo. Se mostrarán nueve circuitos con distintos grados de dificultad y por ende con una cantidad distinta de evaluaciones de la función objetivo. Los resultados obtenidos serán presentados mediante tablas comparativas de las tres representaciones. Por último, se presentarán gráficas de convergencia y los diagramas de los circuitos lógicos encontrados con menor número de compuertas.

Los resultados obtenidos por las tres versiones del PSO son comparados con respecto a los resultados de otras dos técnicas heurísticas: el algoritmo genético de cardinalidad n (NGA) y el algoritmo genético multiobjetivo (MGA) así como por diseñadores humanos que utilizaron mapas de Karnaugh y el método de Quine-McCluskey junto con la simplificación algebraica.

5.1. Experimentos

Las tablas de verdad para realizar los experimentos fueron tomadas de la literatura, principalmente de experimentos realizados con el algoritmo genético de cardinalidad N (llamado NGA) [6] y el algoritmo genético multiobjetivo (llamado MGA) [9] así como algunos ejemplos de archivos en el formato denominado *Berkeley Logic Interchange Format* (blif) que han sido utilizados para optimización de redes de nodos lógicas multinivel [43]. Los ejemplos 1,2,3 y 8 fueron tomados de experimentos llevados a cabo con los algoritmos NGA y MGA y los ejemplo 4,5,6,7 y 9 fueron tomados de arhivos con formato blif y convertidos a tablas de verdad para que pudieran ser introducidos al programa.

Conversión de archivos con formato blif a tablas de verdad

La conversión de archivos en formato blif a tablas de verdad se llevó a cabo mediante un procedimiento en dos etapas. En la primera de ellas a partir del archivo blif se obtiene una expresión booleana en notación prefija. En la segunda etapa, la expresión booleana es introducida a un programa escrito en el lenguaje funcional Scheme el cual realiza la evaluación de la expresión para finalmente obtener la tabla de verdad. En la figura 5.1 se muestra el ejemplo de un archivo con formato blif.

La estructura de los archivos blif puede definir una gran variedad de tipos de circuitos [36]. Sin embargo, para los fines de este trabajo sólo se tomaron los archivos que describían circuitos lógicos combinatorios y por ello sólo se resaltarán los puntos importantes de la estructura con relación a este tipo de circuitos. Así mismo sólo se explicará el procedimiento de conversión realizado para los archivos que describen circuitos lógicos combinatorios y no para todo el conjunto de archivos con formato blif.

El primer paso para la obtención de la expresión booleana consiste en determinar las entradas y salidas del circuito así como sus identificadores. En las líneas **b** y **c** de la figura 5.1 se puede ver el identificador de entradas **.inputs** y los símbolos **A,B,C y D** asociados a cada una de las variables de entrada. De forma similar las salidas están identificadas mediante **.outputs** y en este caso sólo tenemos una salida del circuito con el símbolo **z**.

El resto del archivo contiene líneas con el identificador **.names** seguidas de tres símbolos y de algunas líneas con los símbolos **1,0 y -** hasta que se llega al final del archivo indicado mediante el identificador **.end**. Cada una de

las líneas **.names** se procesa de la misma forma por lo que sólo se explicarán algunas de ellas.

a. .model hard_AZ1
b. .inputs A B C D
c. .outputs z
d. .names A C n1
e. 11 1
f. .names B D n2
g. 0- 1
h. -1 1
i. .names B C n3
j. 10 1
k. .names C D n4
l. 10 1
m. .names n1 n2 n5
n. 11 1
p. .names n2 C n6
q. 00 1
r. .names n3 n4 n7
s. 1- 1
t. -1 1
u. .names A n7 n8
v. 01 1
w. .names n8 n5 n9
x. 1- 1
y. -1 1
z. .names n9 n6 z
A. 1- 1
B. -1 1
C. .end

Figura 5.1: Archivo con formato blif.

Los dos primeros símbolos a continuación del identificador **.name** representan las dos entradas al nodo y el tercero representa la salida que es el

nombre del nodo. En la línea **z** de la figura 5.1 tenemos los símbolos de entrada **n9,n6** y de salida **z**. Evidentemente la salida **z** es la salida del nodo, además de ser la salida de todo el circuito tal como lo indica **outputs** por lo que se debe comenzar en esta línea para obtener la expresión booleana en notación prefija del circuito.

La salida **z** depende de las dos variables de entrada **n9** y **n6**. Para poder relacionar las entradas con la salida se tienen que tomar todas las líneas a continuación de un identificador **.name** hasta que se encuentre otro de ellos o un identificador **.end**. Cada renglón a continuación de **.name** está relacionado mediante una compuerta **OR** mientras que los elementos en el mismo renglón están relacionados mediante una compuerta **AND**. Si en el renglón se presenta un símbolo significa que la variable se tiene que colocar tal como está expresada en la línea de **name**; si es un 0 entonces la variable se encuentra negada y por último si se trata de un símbolo **-** significa que la variable no se encuentra presente en el término. Es importante resaltar que las variables tienen que colocarse en el mismo orden en que aparecen en la línea de **.name**.

Para la línea **z** de la figura 5.1 la expresión obtenida es $z = n6 + n9$ o en notación prefija z = (OR n6 n9). Sin embargo, los símbolos **n9,n6** no pertenecen a los símbolos de las variables de entrada por lo que el siguiente paso consiste en obtener las expresiones que representan a estos nodos hasta que **z** pueda ser expresada sólo en términos de las variables de entrada. El símbolo **n6** se encuentra en la línea **p** de la figura 5.1 en la tercera posición despues del identificador **.name** que indica que **n6** es la salida que se está buscando. La expresión resultante para **n6** es $n6 = n2 \cdot C$ que aún no está definida completamente en términos de las variables de entrada por lo que se tiene que buscar la expresión para **n2**. La salida **z** sería hasta este momento $z = (n2 \cdot C) + n9$ o en notación prefija z = (OR (AND (NOT n2) (NOT C)) n9). La expresión resultante después de todo el proceso se muestra a continuación:

z=(OR (AND (NOT (OR D (NOT B))) (NOT C)) (OR (AND (AND A C) (OR D (NOT B))) (AND (NOT A) (OR (AND C (NOT D)) (AND B (NOT C))))))

El último paso para obtener la tabla de verdad a partir de la expresión booleana en notación prefija se realiza mediante un programa en el lenguaje funcional Scheme que evalúa la expresión para cada una de las combinaciones de las variables de entrada. Para ello se define una variable que contiene

todas las combinaciones posibles dependiendo del número de variables de entrada, una función que tiene la expresión booleana a evaluar y un par de funciones más para tomar cada una de las combinaciones y evaluarla en la expresión booleana. El programa completo para evaluar una expresión de cuatro variables de entrada y una salida se muestra en la figura 5.2.

```
(define lista '(
(0 0 0 0) (0 0 0 1) (0 0 1 0) (0 0 1 1)
(0 1 0 0) (0 1 0 1) (0 1 1 0) (0 1 1 1)
(1 0 0 0) (1 0 0 1) (1 0 1 0) (1 0 1 1)
(1 1 0 0) (1 1 0 1) (1 1 1 0) (1 1 1 1)))

(define evaluaTodo
(lambda (ls)
(if (null? ls) '()
(cons (funcbool (caar ls) (cadar ls) (caddar ls) (car (cdddar ls)))
(evaluaTodo (cdr ls))))))

(define evalua
(lambda ()
(evaluaTodo lista)))

(define funcbool
(lambda (D C B A)
(OR (AND (NOT (OR D (NOT B))) (NOT C))
    (OR (AND (AND A C) (OR D (NOT B)))
        (AND (NOT A) (OR (AND C (NOT D)) (AND B (NOT C))))))))
```

Figura 5.2: Programa en lenguaje Scheme que evalúa la expresión booleana obtenida del archivo blif de la figura 5.1.

5.2. Ejemplo 1

El ejemplo 1 es un circuito sencillo de tres entradas y una salida cuya tabla de verdad se muestra en la tabla 5.1. Los resultados obtenidos por las tres representaciones en 20 corridas independientes se muestran en la tabla 5.2. Puede observarse que las tres representaciones encontraron circuitos factibles en todas las corridas. Sin embargo la representación binaria encontró circuitos hasta con 6 y 7 compuertas mientras que la representación entera A y B produjeron circuitos con sólo 4 y 5 compuertas además de que esta última encontró circuitos de 4 compuertas un mayor número de veces.

A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Tabla 5.1: Tabla de verdad del ejemplo 1.

Los parámetros utilizados para la obtención de estos resultados se encuentran en la tabla 5.3 además de una comparación más detallada entre la representación binaria, entera A y entera B del PSO, el NGA, el MGA y un diseñador humano. En la figura 5.3 se muestra una gráfica de convergencia comparativa (de la corrida en la mediana) de las tres representaciones donde puede verse que la representación entera A fue la que convergió más rápido para este ejemplo. En la figura 5.4 se muestra el diagrama lógico del mejor circuito el cual requiere de 4 compuertas para su implementación.

Para este ejemplo el MGA tuvo un desempeño muy similar al de las tres versiones del PSO, aunque encontró un mayor número de veces el circuito con 4 compuertas y tuvo una aptitud promedio ligeramente mayor que la representación entera B. El NGA no tuvo un muy buen desempeño en comparación con el PSO y el MGA ya que encontró la solución de 4 compuertas tan sólo el 10 % de las veces y circuitos factibles sólo el 80 % mientras que los demás encontraron circuitos factibles en todas las corridas. La expresión booleana encontrada por el diseñador humano fue $S = A(B \oplus C) + C(A \oplus B)$ que tiene 5 compuertas y se obtuvo mediante mapas de Karnaugh e identidades del algebra booleana. Esta expresión fue reportada por Coello [9].

Corrida	Binaria			Entera A			Entera B		
	AP[1]	**MA**[2]	**NC**[3]	**AP**	**MA**	**NC**	**AP**	**MA**	**NC**
0	19.13	28	5	24.91	28	5	24.40	29	4
1	16.98	29	4	26.43	29	4	25.80	29	4
2	16.75	28	5	25.79	29	4	24.63	29	4
3	18.37	28	5	25.77	28	5	25.19	29	4
4	18.11	28	5	26.46	28	5	23.96	28	5
5	15.29	27	6	26.22	28	5	24.90	28	5
6	18.24	28	5	25.89	29	4	25.13	29	4
7	15.71	26	7	26.10	29	4	21.61	28	5
8	14.67	27	6	26.04	28	5	22.48	29	4
9	15.73	27	6	26.91	29	4	23.89	29	4
10	17.14	29	4	26.55	29	4	22.50	28	5
11	17.70	29	4	26.52	28	5	25.72	28	5
12	16.23	27	6	24.78	28	5	24.49	28	5
13	17.44	29	4	25.05	28	5	23.31	28	5
14	15.71	28	5	27.37	29	4	23.28	29	4
15	18.07	28	5	27.49	29	4	24.90	29	4
16	16.27	28	5	26.51	28	5	22.56	29	4
17	18.14	28	5	27.40	29	4	25.95	29	4
18	15.95	27	6	26.00	28	5	20.63	29	4
19	16.11	28	5	25.92	28	5	26.62	29	4

[1]Aptitud promedio [2]Mejor aptitud [3]Número de compuertas

Tabla 5.2: Análisis comparativo de 20 corridas del ejemplo 1.

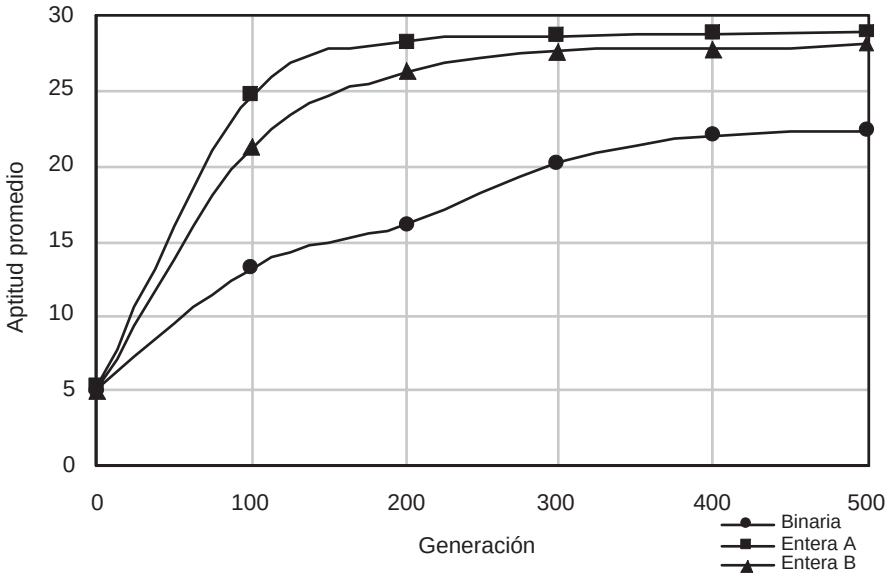
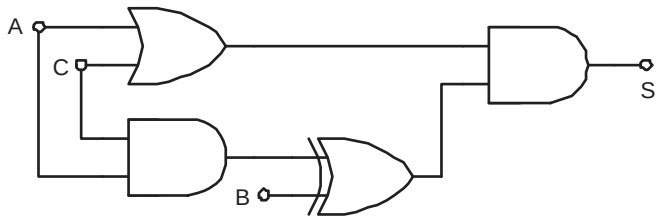


Figura 5.3: Gráfica de convergencia del ejemplo 1 de la corrida ubicada en la mediana.

Parámetros	PSO B	PSO EA	PSO EB	MGA	NGA	DH
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE					
Matriz	5×5					-
Tamaño de la población	50			90		-
Número de generaciones	500			300		-
Evaluaciones	25,000			27,000		-
Número de corridas	20					-
$\phi_1 = \phi_2$	0.8	0.2		-		-
Vmax	3.0	0.4		-		-
Pm	10 %			0.6667 %		-
Pc	-			50 %		-
Tamaño del vecindario	3			-		-
Cardinalidad	3			5		-
Número de compuertas de la mejor solución	4	4	4	4	4	5
Mejor solución	20 %	45 %	65 %	75 %	10 %	-
Circuitos factibles	100 %	100 %	100 %	100 %	80 %	-
Número de compuertas promedio	5.15	4.55	4.35	4.25	11.60	-
Aptitud promedio	27.85	28.45	28.65	28.75	21.40	-
Desviación estándar	0.8127	0.5104	0.4893	0.4330	8.4380	-

Tabla 5.3: Comparación de resultados entre el PSO, un diseñador humano, el MGA y el NGA para el ejemplo 1.



$$S = (A + C)(B \oplus AC)$$

Figura 5.4: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 1.

5.3. Ejemplo 2

La tabla de verdad de este ejemplo consta de cuatro entradas y una salida y se muestra en la tabla 5.4. Los resultados obtenidos por las tres representaciones son mostrados en la tabla 5.5. En este caso (a diferencia del ejemplo 1) no se encontraron circuitos factibles en todas las corridas además de que el mejor circuito encontrado mostrado en la figura 5.6 tiene 6 compuertas y sólo se encontró una sola vez. A pesar de que en la gráfica de convergencia de la figura 5.5 se puede apreciar que la representación entera A converge más rápidamente, la representación entera B obtuvo mejores resultados globales como se muestra en la tabla 5.6.

D	C	B	A	S
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Tabla 5.4: Tabla de verdad del ejemplo 2.

Para este ejemplo el MGA no tuvo tan buen desempeño como las dos versiones enteras del PSO, ya que el mejor circuito que encontró tiene 7 compuertas contra las 6 compuertas del mejor circuito encontrado por la representación entera A. El MGA encontró el circuito de 7 compuertas el 15 % de las veces, mientras que la representación entera B lo hizo el 30 % además de que esta última encontró circuitos factibles el 90 % de las veces contra el 35 % del MGA. En comparación con el PSO binario el MGA tuvo un desempeño similar a pesar de que el mejor circuito encontrado por el PSO binario fue de 9 compuertas. La expresión booleana encontrada por el diseñador humano fue $S = ((A \oplus B) \oplus ((AD)(B + C))) + ((A + C) + D)'$ que tiene 9 compuertas y se obtuvo mediante mapas de Karnaugh. Esta expresión fue reportada por Coello [9].

Corrida	Binaria			Entera A			Entera B		
	AP	MA	NC	AP	MA	NC	AP	MA	NC
0	14.12	15	*[1]	22.31	31	10	19.72	34	7
1	14.45	30	11	30.92	33	8	21.36	33	8
2	14.17	15	*	29.51	34	7	17.16	31	10
3	14.80	30	11	30.30	34	7	17.88	32	9
4	14.11	15	*	31.88	34	7	17.81	34	7
5	13.94	15	*	14.94	15	*	20.98	33	8
6	14.09	15	*	29.60	33	8	20.37	33	8
7	15.34	32	9	20.19	32	8	14.64	15	*
8	15.45	32	9	29.12	34	7	22.41	33	8
9	14.09	29	12	14.90	15	*	17.30	33	8
10	15.58	32	9	14.93	15	*	20.14	34	7
11	14.27	15	*	18.77	31	10	18.85	32	9
12	14.77	31	10	32.42	35	6	14.32	15	*
13	14.47	29	12	14.93	15	*	19.22	34	7
14	14.08	15	*	29.39	34	7	20.07	34	7
15	14.31	15	*	14.90	15	*	15.81	33	8
16	14.12	28	13	14.77	15	*	21.99	34	7
17	13.93	15	*	29.62	32	9	15.00	31	10
18	14.32	15	*	14.91	15	*	16.79	33	8
19	14.10	15	*	30.44	33	8	28.37	33	8

[1] Circuito no factible.

Tabla 5.5: Análisis comparativo de 20 corridas del ejemplo 2.

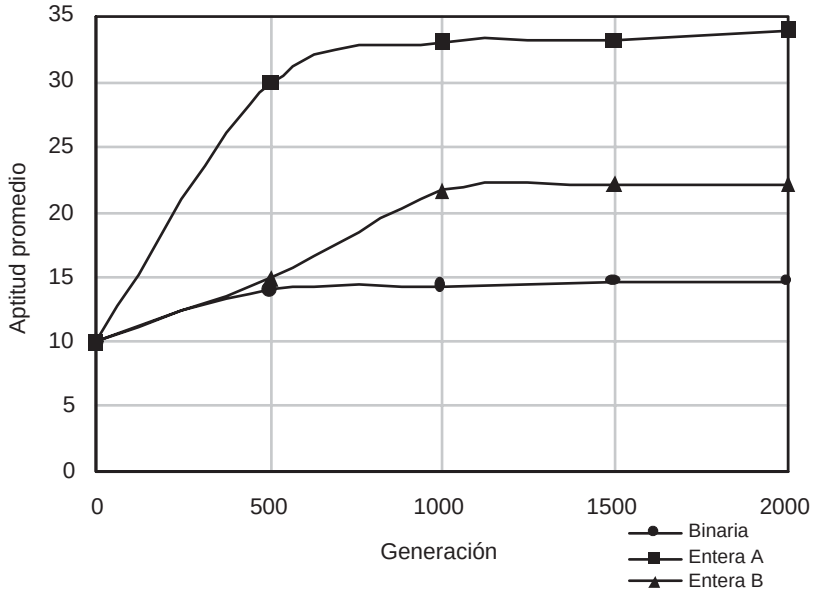
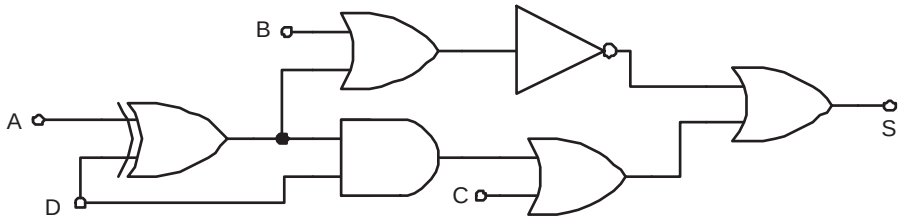


Figura 5.5: Gráfica de convergencia del ejemplo 2 de la corrida ubicada en la mediana.

Parámetros	PSO B	PSO EA	PSO EB	MGA	DH
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE				
Matriz	5×5				-
Tamaño de la población	50			170	-
Número de generaciones	2000			600	-
Evaluaciones	100,000			102,000	-
Número de corridas	20				-
$\phi_1 = \phi_2$	0.8	0.2		-	-
Vmax	3.0	0.4		-	-
Pm	10 %			0.6667 %	-
Pc	-			50 %	-
Tamaño del vecindario	3			-	-
Cardinalidad	3			5	-
Número de compuertas de la mejor solución	9	6	7	7	9
Mejor solución	15 %	5 %	30 %	15 %	-
Circuitos factibles	45 %	65 %	90 %	35 %	-
Número de compuertas promedio	19.1	14.25	9.80	19.95	-
Aptitud promedio	21.9	26.75	31.20	21.05	-
Desviación estándar	7.8867	8.9022	5.6157	8.9294	-

Tabla 5.6: Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 2.



$$S = (B + (D \oplus A))' + (C \oplus D(D \oplus A))$$

Figura 5.6: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 2.

5.4. Ejemplo 3

La tabla de verdad de este ejemplo se muestra en la tabla 5.7. Los resultados obtenidos por las tres representaciones son mostrados en la tabla 5.8. Para este ejemplo la representación binaria no encontró muchos circuitos factibles además de que la mejor solución que encontró fue un circuito de 9 compuertas mientras que las otras dos encontraron un circuito con 7 compuertas que es mostrado en la figura 5.8. La gráfica de convergencia de la figura 5.7 muestra que la representación entera A también converge más rápido pero la entera B tiene un mejor comportamiento global como lo muestra la tabla 5.9.

D	C	B	A	S
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Tabla 5.7: Tabla de verdad del ejemplo 3.

En este ejemplo, el mejor circuito encontrado por las dos versiones enteras del PSO y por el MGA tiene 7 compuertas, el cual se obtuvo en más o menos el mismo número de corridas en las tres heurísticas. Sin embargo, las dos representaciones enteras del PSO encontraron circuitos factibles en al menos 75 % de las veces mientras que el MGA sólo lo consiguió el 35 %. La aptitud promedio del MGA fue mayor que la aptitud promedio de la versión binaria del PSO pero fue menor que las aptitudes promedio de las dos versiones enteras del PSO. La expresión booleana encontrada por el diseñador humano fue $S = BC + A'(C' + D') + B'(AD)$ que tiene 11 compuertas y se obtuvo mediante mapas de Karnaugh por la autora de esta tesis y el resultado se utilizó como referencia para evaluar los resultados obtenidos por el PSO.

Corrida	Binaria			Entera A			Entera B		
	AP	**MA**	**NC**	**AP**	**MA**	**NC**	**AP**	**MA**	**NC**
0	14.30	15	*	31.35	34	7	17.45	32	9
1	14.28	15	*	24.18	32	9	15.86	31	10
2	14.16	15	*	14.92	15	*	17.03	30	11
3	14.44	32	9	18.06	32	9	19.50	33	8
4	13.89	15	*	29.62	33	8	18.85	34	7
5	14.26	15	*	25.32	33	8	23.10	33	8
6	14.11	15	*	28.30	32	9	19.58	33	8
7	13.95	15	*	26.40	32	9	14.74	15	*
8	14.01	15	*	30.27	33	8	17.88	33	8
9	14.05	15	*	15.12	30	11	19.84	34	7
10	14.05	15	*	14.93	15	*	27.28	34	7
11	13.97	15	*	28.40	33	8	18.18	33	8
12	13.85	15	*	14.91	15	*	14.71	15	*
13	14.44	26	15	26.19	31	10	21.38	33	8
14	14.24	15	*	32.09	34	7	25.28	33	8
15	14.16	28	13	14.88	15	*	16.77	33	8
16	14.33	15	*	31.06	33	8	19.02	32	9
17	15.15	31	10	33.00	34	7	27.83	34	7
18	14.67	31	10	31.17	33	8	18.14	32	9
19	14.22	15	*	14.94	15	*	16.73	31	10

Tabla 5.8: Análisis comparativo de 20 corridas del ejemplo 3.

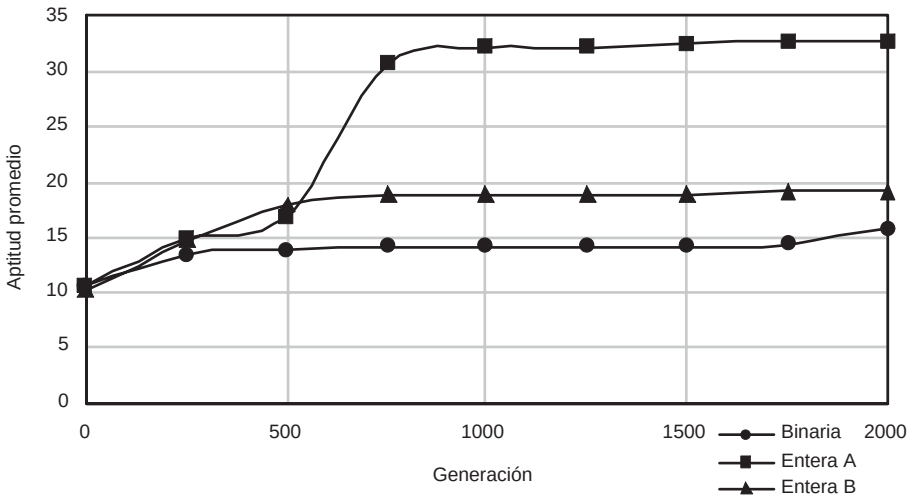
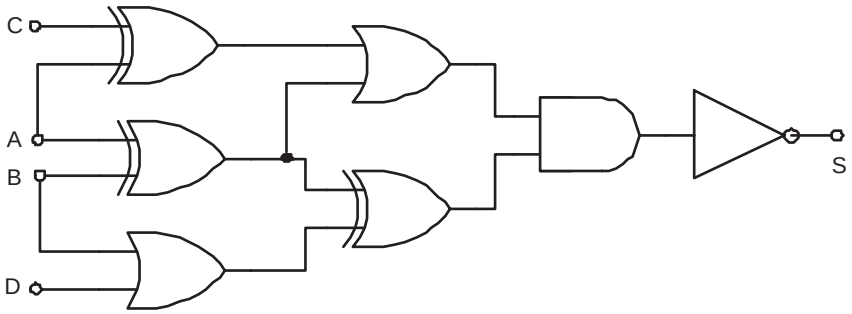


Figura 5.7: Gráfica de convergencia del ejemplo 3 de la corrida ubicada en la mediana.

Parámetros	PSO B	PSO EA	PSO EB	MGA	DH
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE				
Matriz	5×5				-
Tamaño de la población	50			170	-
Número de generaciones	2000			600	-
Evaluaciones	100,000			102,000	-
Número de corridas	20				-
$\phi_1 = \phi_2$	0.8	0.2		-	-
Vmax	3.0	0.4		-	-
Pm	10 %			0.6667 %	-
Pc	-			50 %	-
Tamaño del vecindario	3			-	-
Cardinalidad	3			5	-
Número de compuertas de la mejor solución	9	7	7	7	11
Mejor solución	5 %	15 %	25 %	15 %	-
Circuitos factibles	25 %	75 %	90 %	35 %	-
Número de compuertas promedio	22.35	14.25	10.10	20.55	-
Aptitud promedio	18.65	28.2	30.90	20.45	-
Desviación estándar	6.5877	7.8780	5.5431	8.3759	-

Tabla 5.9: Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 3.



$$S = (((C \oplus A) + (B \oplus A))((B + D) \oplus (B \oplus A)))'$$

Figura 5.8: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 3.

5.5. Ejemplo 4

La tabla de verdad de este ejemplo se muestra en la tabla 5.10 y fue tomada del archivo blif nombrado *miller*. Los resultados obtenidos por las tres representaciones son mostrados en la tabla 5.11. Para este ejemplo la representación binaria y la entera B encontraron circuitos factibles en todas las corridas mientras que la entera A falló en una sola (tabla 5.12). Por otro lado las tres representaciones encontraron que el menor número de compuertas necesarias para construir el circuito es de 6 (figura 5.10). La gráfica de convergencia de la figura 5.9 muestra que la representación entera A tuvo una convergencia mucho más abrupta que las otras dos representaciones. Sin embargo, en términos generales la representación entera B tuvo un mejor desempeño.

D	**C**	**B**	**A**	**S**
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Tabla 5.10: Tabla de verdad del ejemplo 4.

Las cuatro técnicas heurísticas tuvieron un desempeño similar en cuanto a la obtención de circuitos factibles al encontraron circuitos factibles en al menos el 90 % de las corridas. El mejor circuito encontrado por las cuatro técnicas tiene 6 compuertas sólo que para el caso del PSO binario y el PSO entero A se encontró el 75 % de las veces, para el PSO entero B el 85 % mientras que el MGA sólo lo consiguió el 30 % del tiempo además de que su aptitud promedio fue menor que las tres versiones del PSO. La expresión booleana encontrada por el diseñador humano fue $S = (A \oplus B) \oplus (C \oplus D) + D'(CA) + B(A'D)$ que tiene 11 compuertas y fue obtenido por la autora de esta tesis mediante mapas de Karnaugh y álgebra booleana.

Corrida	Binaria			Entera A			Entera B		
	AP	MA	NC	AP	MA	NC	AP	MA	NC
0	16.52	35	6	29.59	35	6	28.75	35	6
1	15.06	35	6	27.10	34	7	30.25	35	6
2	14.59	29	12	29.33	34	7	32.54	35	6
3	16.97	33	8	33.19	34	7	28.37	35	6
4	16.08	31	10	31.60	35	6	26.75	35	6
5	16.86	35	6	34.08	35	6	28.16	35	6
6	19.58	35	6	33.21	35	6	28.82	35	6
7	17.50	35	6	34.05	35	6	27.21	35	6
8	16.60	33	8	34.16	35	6	23.50	35	6
9	18.38	34	7	33.71	35	6	28.64	35	6
10	19.80	35	6	27.79	32	9	28.43	35	6
11	17.53	35	6	33.21	35	6	29.57	35	6
12	18.52	35	6	33.24	35	6	27.23	35	6
13	18.60	35	6	33.19	35	6	23.74	35	6
14	16.47	35	6	29.25	35	6	27.25	35	6
15	20.81	35	6	33.85	35	6	26.07	34	7
16	17.65	35	6	14.87	15	*	22.09	34	7
17	18.01	35	6	33.96	35	6	31.57	35	6
18	19.01	35	6	33.60	35	6	20.84	34	7
19	18.16	35	6	34.06	35	6	25.61	35	6

Tabla 5.11: Análisis comparativo de 20 corridas del ejemplo 4.

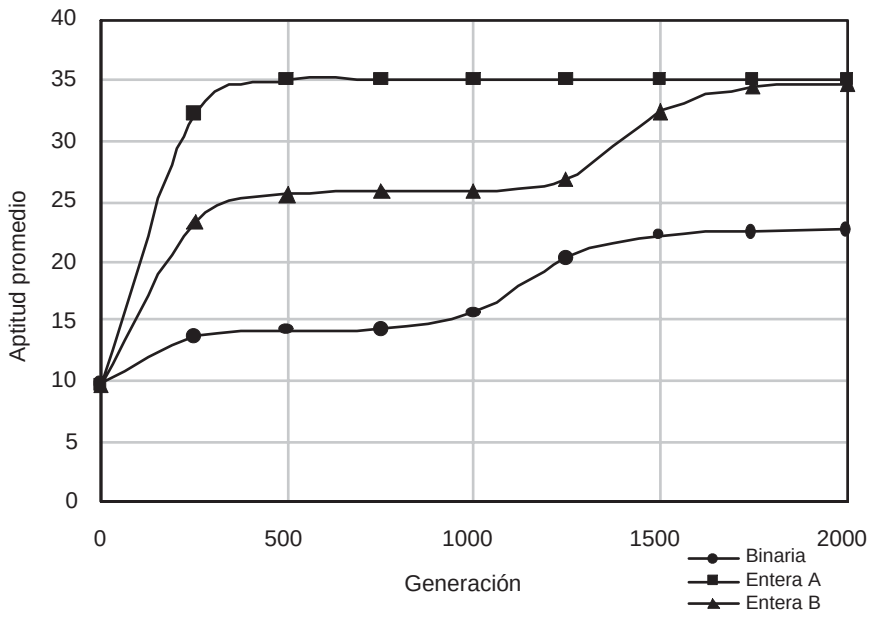
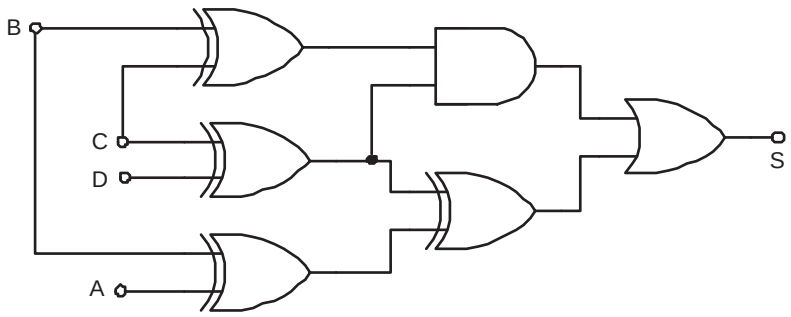


Figura 5.9: Gráfica de convergencia del ejemplo 4 de la corrida ubicada en la mediana.

Parámetros	PSO B	PSO EA	PSO EB	MGA	DH
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE				
Matriz	5×5				-
Tamaño de la población	50			170	-
Número de generaciones	2000			600	-
Evaluaciones	100,000			102,000	-
Número de corridas	20				-
$\phi_1 = \phi_2$	0.8	0.2		-	-
Vmax	3.0	0.4		-	-
Pm	10 %			0.6667 %	-
Pc	-			50 %	-
Tamaño del vecindario	3			-	-
Cardinalidad	3			5	-
Número de compuertas de la mejor solución	6	6	6	6	11
Mejor solución	75 %	75 %	85 %	30 %	-
Circuitos factibles	100 %	95 %	100 %	90 %	-
Número de compuertas promedio	6.75	7.30	6.15	9.30	-
Aptitud promedio	34.25	33.70	34.85	31.70	-
Desviación estándar	1.6181	4.4615	0.3664	6.2669	-

Tabla 5.12: Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 4.



$$S = (C \oplus D)(B \oplus C) + ((B \oplus A) \oplus (C \oplus D))$$

Figura 5.10: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 4.

5.6. Ejemplo 5

La tabla de verdad del archivo blif *rnd4-1* de este ejemplo se muestra en la tabla 5.13. Los resultados obtenidos por las tres representaciones son mostrados en la tabla 5.14. La representación binaria encontró circuitos factibles en todas las corridas mientras que la entera A y B no lo lograron (tabla 5.15). A pesar de que la representación binaria y entera B encontraron que un circuito de 6 compuertas, la representación entera encontró un circuito válido con 5 compuertas (figura 5.12) pero sólo apareció una vez en todas las corridas. A pesar de que la gráfica de convergencia de la figura 5.11 muestra que la representación entera A tuvo una mejor aptitud promedio en la mediana de la corrida, la representación entera B tuvo un mejor desempeño global.

D	C	B	A	S
0	0	0	0	0
0	0	0	1	1
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	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Tabla 5.13: Tabla de verdad del ejemplo 5.

MGA también encontró un circuito de 5 compuertas con la diferencia de que la representación entera A lo halló el 5 % de las veces mientras que el MGA lo hizo en el 15 % de las corridas. Sin embargo, este último sólo encontró circuitos factibles en el 65 % de las corridas mientras que el PSO entero A hizo lo propio el 85 % de las veces. A pesar de que el PSO binario y entero B sólo encontraron un circuito de 6 compuertas, la aptitud promedio fue mucho mayor que el MGA. La expresión booleana encontrada por el diseñador humano fue $S = C((A \oplus B) \oplus D)) + D(A \oplus C)' + AD'C'$ que tiene 12 compuertas y fue obtenido por la autora de esta tesis mediante mapas de Karnaugh y álgebra booleana.

Corrida	Binaria			Entera A			Entera B		
	AP	**MA**	**NC**	**AP**	**MA**	**NC**	**AP**	**MA**	**NC**
0	16.60	34	7	33.98	36	5	14.92	15	*
1	15.85	34	7	32.36	35	6	17.65	33	8
2	16.19	35	6	33.25	35	6	22.71	34	7
3	19.92	34	7	28.59	32	9	28.35	34	7
4	18.70	34	7	29.55	34	7	14.91	15	*
5	15.61	31	10	14.92	15	*	24.70	35	6
6	16.68	34	7	27.48	33	8	14.88	15	*
7	21.13	35	6	14.95	15	*	27.26	34	7
8	17.05	32	9	32.22	34	7	25.37	35	6
9	15.96	32	9	33.53	35	6	23.15	35	6
10	16.96	34	7	30.38	32	9	23.27	34	7
11	14.51	15	*	31.14	34	7	21.82	34	7
12	14.85	31	10	30.01	32	9	20.37	35	6
13	16.84	34	7	14.86	15	*	21.98	34	7
14	17.29	32	9	29.92	33	8	28.89	34	7
15	21.15	34	7	31.68	35	6	27.88	35	6
16	17.99	34	7	33.03	34	7	28.19	35	6
17	15.37	34	7	33.23	35	6	26.62	35	6
18	15.87	34	7	25.98	31	10	19.04	34	7
19	16.13	31	10	34.08	35	6	20.95	34	7

Tabla 5.14: Análisis comparativo de 20 corridas del ejemplo 5.

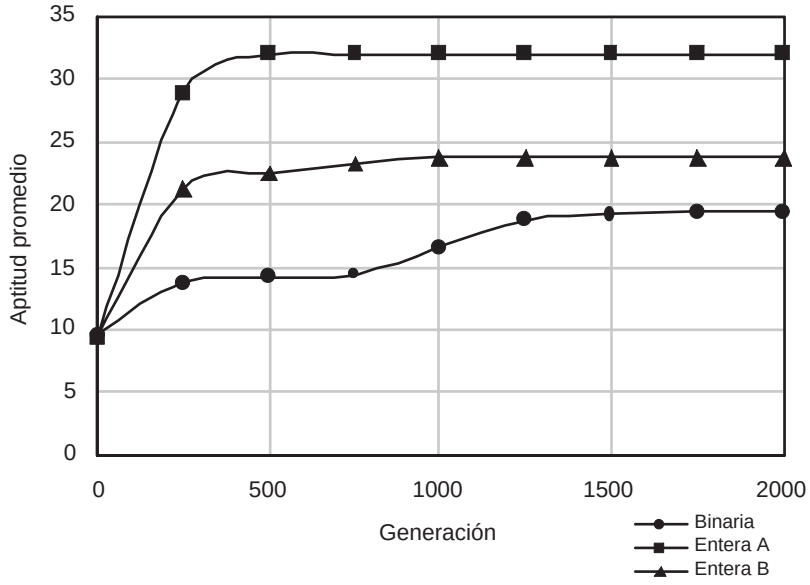
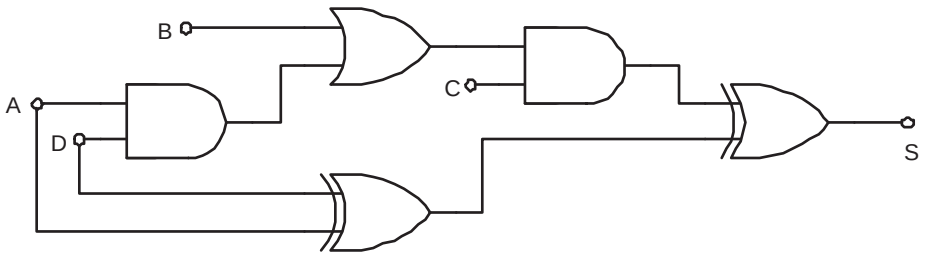


Figura 5.11: Gráfica de convergencia del ejemplo 5 de la corrida ubicada en la mediana.

Parámetros	PSO B	PSO EA	PSO EB	MGA	DH
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE				
Matriz	5×5				-
Tamaño de la población	50			170	-
Número de generaciones	2000			600	-
Evaluaciones	100,000			102,000	-
Número de corridas	20				-
$\phi_1 = \phi_2$	0.8	0.2		-	-
Vmax	3.0	0.4		-	-
Pm	10 %			0.6667 %	-
Pc	-			50 %	-
Tamaño del vecindario	3			-	-
Cardinalidad	3			5	-
Número de compuertas de la mejor solución	6	5	6	5	12
Mejor solución	10 %	5 %	35 %	15 %	-
Circuitos factibles	95 %	85 %	85 %	65 %	-
Número de compuertas promedio	8.60	10	8.60	14.80	-
Aptitud promedio	32.40	31.00	32.40	26.20	-
Desviación estándar	4.2969	7.0187	7.1117	9.1168	-

Tabla 5.15: Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 5.



$$S = C(AD + B) \oplus (A \oplus D))$$

Figura 5.12: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 5.

5.7. Ejemplo 6

La tabla de verdad de este ejemplo se muestra en la tabla 5.16 y fue tomada del archivo *rnd4-4*. Los resultados obtenidos por las tres representaciones son mostrados en la tabla 5.17. La representación entera B encontró circuitos factibles en todas las corridas (tabla 5.18) mientras que las otras dos fallaron en algunas. El mejor circuito encontrado por las tres representaciones tiene 5 compuertas (figura 5.14) y en la representación entera B fue encontrado en más de la mitad de las corridas. La gráfica de convergencia de la figura 5.13 muestra que la representación entera A tuvo una mejor aptitud. Sin embargo la representación entera B tuvo un mejor desempeño global.

D	C	B	A	S
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Tabla 5.16: Tabla de verdad del ejemplo 6.

Para este ejemplo las tres versiones del PSO y el MGA encontraron un circuito de tan solo 5 compuertas. Sin embargo el MGA lo encontró sólo el 10 % de las veces mientras que el PSO binario lo logró el 20 %, el PSO entero A 40 % y el PSO entero B el 55 % de las corridas. Por otro lado, las tres versiones del PSO encontraron circuitos factibles en al menos el 75 % de las corridas mientras que el MGA lo consiguió sólo el 70 % de las veces. La aptitud promedio del MGA fue de 27.30 (13.70 compuertas promedio) mientras que la mejor aptitud promedio del PSO fue de 35.10 (5.90 compuertas promedio) que es un valor mucho mejor. La expresión booleana encontrada por el diseñador humano fue $S = D(B' + B(A \oplus C)) + C'(B \oplus D)$ que tiene 9 compuertas y fue obtenido por la autora de esta tesis mediante mapas de Karnaugh y álgebra booleana.

Corrida	Binaria			Entera A			Entera B		
	AP	MA	NC	AP	MA	NC	AP	MA	NC
0	14.01	15	*	32.72	35	6	20.98	36	5
1	14.98	34	7	29.84	33	8	27.52	35	6
2	15.70	34	7	14.69	31	10	20.28	33	8
3	16.26	34	7	35.12	36	5	20.82	36	5
4	16.54	36	5	29.22	34	7	21.93	35	6
5	14.50	33	8	30.05	36	5	15.11	32	9
6	14.55	32	9	14.78	15	*	29.05	36	5
7	17.16	36	5	29.43	36	5	27.94	36	5
8	17.35	34	7	14.89	15	*	27.72	35	6
9	14.17	27	14	33.69	35	6	16.80	32	9
10	16.58	34	7	28.62	35	6	18.68	36	5
11	18.52	36	5	32.69	36	5	25.61	36	5
12	15.02	33	8	34.53	36	5	24.44	36	5
13	16.67	34	7	34.70	36	5	20.59	35	6
14	14.18	29	12	34.10	36	5	31.69	36	5
15	17.36	36	5	30.64	33	8	23.87	34	7
16	14.35	30	11	31.12	33	8	21.00	35	6
17	14.14	15	*	32.42	34	7	30.99	36	5
18	14.10	15	*	33.16	34	7	26.14	36	5
19	14.74	35	6	34.77	36	5	27.44	36	5

Tabla 5.17: Análisis comparativo de 20 corridas del ejemplo 6.

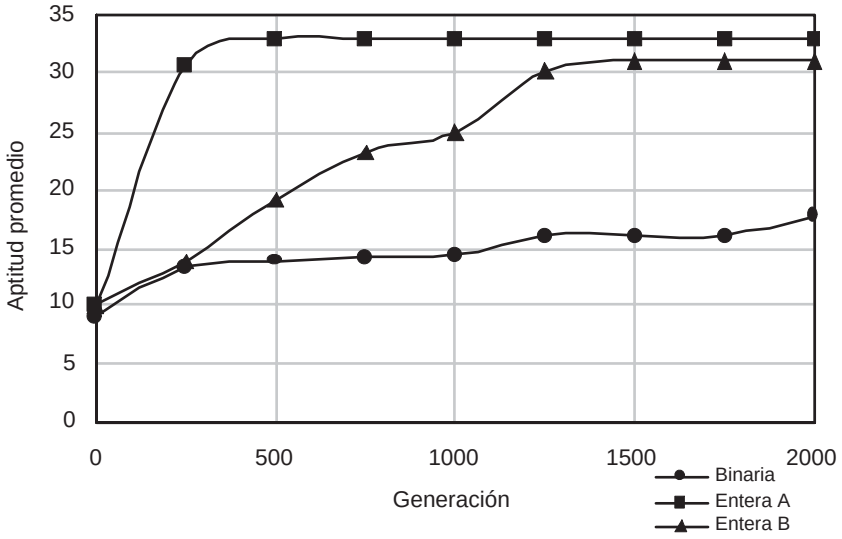
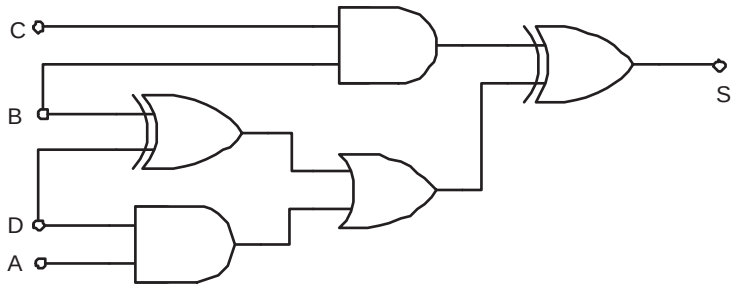


Figura 5.13: Gráfica de convergencia del ejemplo 6 de la corrida ubicada en la mediana.

Parámetros	**PSO B**	**PSO EA**	**PSO EB**	**MGA**	**DH**
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE				
Matriz	5×5				-
Tamaño de la población	50			170	-
Número de generaciones	2000			600	-
Evaluaciones	100,000			102,000	-
Número de corridas	20				-
$\phi_1 = \phi_2$	0.8	0.2		-	-
Vmax	3.0	0.4		-	-
Pm	10 %			0.6667 %	-
Pc	-			50 %	-
Tamaño del vecindario	3			-	-
Cardinalidad	3			5	-
Número de compuertas de la mejor solución	5	5	5	5	9
Mejor solución	20 %	40 %	55 %	10 %	-
Circuitos factibles	85 %	90 %	100 %	70 %	-
Número de compuertas promedio	10.40	8.25	5.90	13.70	-
Aptitud promedio	30.60	32.75	35.1	27.30	-
Desviación estándar	7.1259	6.2313	1.3338	9.3982	-

Tabla 5.18: Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 6.



$$S = CB \oplus (DA + (B \oplus D))$$

Figura 5.14: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 6.

5.8. Ejemplo 7

Este ejemplo fue tomado de un archivo blif llamado *rnd4-9* y su tabla de verdad se muestra en la tabla 5.19. Los resultados obtenidos por las tres representaciones son mostrados en la tabla 5.20. En este ejemplo la representación entera A y B tuvieron un mucho mejor desempeño que la representación binaria ya que esta última encontró circuitos factibles sólo el 15 % de la veces (tabla 5.21) y el mejor de los circuitos que obtuvo tiene 9 compuertas. En contraste las representaciones enteras encontraron circuitos factibles el 60 % de las veces y además se encontró un circuito que necesita sólo 6 compuertas (figura 5.16). En comparación, las representación entera A funcionó mejor que la entera B ya que en la primera de ellas se encontró el circuito de 6 compuertas en 8 corridas mientras que la entera B apenas lo consiguió en 4.

D	C	B	A	S
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Tabla 5.19: Tabla de verdad del ejemplo 7.

La gráfica de convergencia de la figura 5.15 muestra que la representación entera A también tuvo cambios más abruptos en relación a la aptitud promedio pero en este ejemplo su desempeño global sí fue mejor que el de las otras dos representaciones. En este caso las representaciones enteras del PSO fueron mucho mejores que el MGA ya que, a pesar de que el mejor circuito encontrado por las tres técnicas requiere de 6 compuertas, las representación entera A lo halló el 40 % de las veces y la representación entera B el 20 % mientras que el MGA sólo lo encontró en el 5 % de las corridas.

Además las representaciones enteras del PSO hallaron circuitos factibles el 60 % del tiempo mientras que el MGA sólo lo logró el 10 %. La aptitud

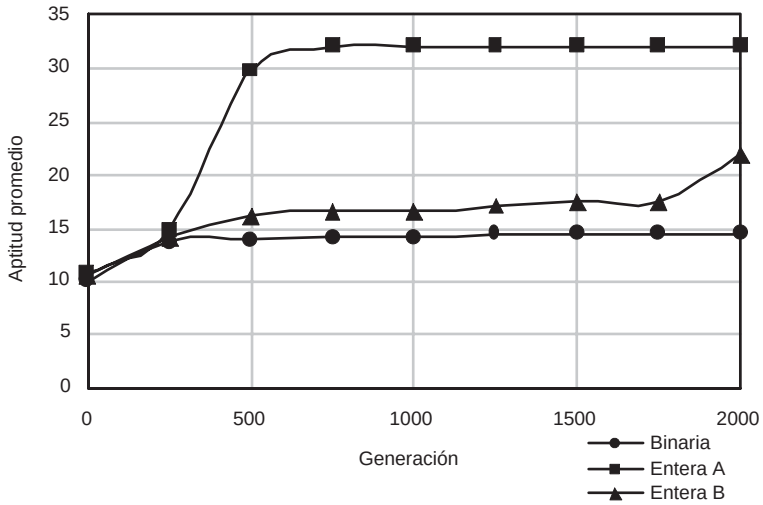
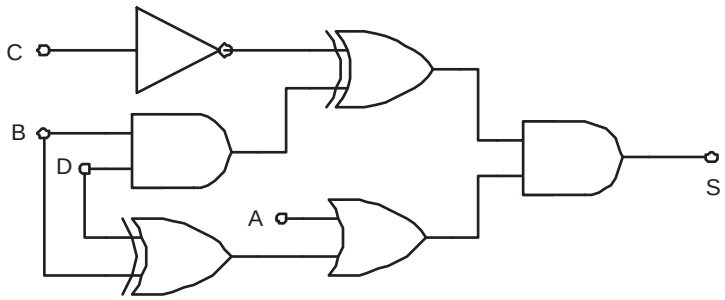


Figura 5.15: Gráfica de convergencia del ejemplo 7 de la corrida ubicada en la mediana.

Corrida	**Binaria**			**Entera A**			**Entera B**		
	AP	**MA**	**NC**	**AP**	**MA**	**NC**	**AP**	**MA**	**NC**
0	21.88	35	6	14.92	15	*	21.88	35	6
1	14.49	15	*	14.82	15	*	14.49	15	*
2	16.50	30	11	14.87	15	*	16.50	30	11
3	20.49	35	6	28.44	32	9	20.49	35	6
4	16.68	34	7	33.22	35	6	16.68	34	7
5	14.83	15	*	14.87	15	*	14.83	15	*
6	19.39	33	9	15.20	30	11	19.39	33	8
7	14.62	15	*	14.69	15	*	14.62	15	*
8	14.68	15	*	32.09	35	6	14.68	15	*
9	14.32	15	*	32.57	35	6	14.32	15	*
10	14.79	15	*	14.45	15	*	14.79	15	*
11	22.17	35	6	31.97	34	7	22.17	35	6
12	14.65	15	*	32.19	34	7	14.65	15	*
13	21.12	34	7	14.90	15	*	21.12	34	7
14	24.58	33	8	33.36	35	6	24.58	33	8
15	18.26	32	9	30.05	35	6	18.26	32	9
16	20.01	35	6	32.67	35	6	20.01	35	6
17	14.87	15	*	14.92	15	*	14.87	15	*
18	20.48	33	9	29.15	35	6	20.48	33	8
19	14.91	32	9	33.36	35	6	14.91	32	9

Tabla 5.20: Análisis comparativo de 20 corridas del ejemplo 7.



$$S = (C' \oplus BD)((B \oplus D) + A)$$

Figura 5.16: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 7.

promedio del MGA fue menor que las tres versiones del PSO aunque el mejor circuito encontrado por el PSO binario requiere 9 compuertas. La expresión booleana encontrada por el diseñador humano fue $S = C'(B \oplus D) + A(D'C' + DCB)$ que tiene 9 compuertas y fue obtenido por la autora de esta tesis mediante mapas de Karnaugh y álgebra booleana.

Parámetros	PSO B	PSO EA	PSO EB	MGA	DH
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE				
Matriz	5×5				-
Tamaño de la población	50			170	-
Número de generaciones	2000			600	-
Evaluaciones	100,000			102,000	-
Número de corridas	20				-
$\phi_1 = \phi_2$	0.8	0.2		-	-
Vmax	3.0	0.4		-	-
Pm	10 %			0.6667 %	-
Pc	-			50 %	-
Tamaño del vecindario	3			-	-
Cardinalidad	3			5	-
Número de compuertas de la mejor solución	9	6	6	6	10
Mejor solución	10 %	40 %	20 %	5 %	-
Circuitos factibles	15 %	60 %	60 %	10 %	-
Número de compuertas promedio	23.50	14.50	14.95	24.40	- -
Aptitud promedio	17.50	26.50	26.05	16.60	-
Desviación estándar	6.1087	9.7089	9.3328	5.6606	-

Tabla 5.21: Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 7.

5.9. Ejemplo 8

La tabla de verdad de este ejemplo se muestra en la tabla 5.22. Los resultados obtenidos por las tres representaciones se muestran en la tabla 5.23. Ninguna de las tres representaciones de PSO consiguió encontrar circuitos factibles en todas las corridas. Sin embargo la que obtuvo un porcentaje más alto fue la representación entera B. Por otro lado, el mejor circuito encontrado tiene 7 compuertas (figura 5.18) y fue encontrado en dos corridas de las tres representaciones. En la figura 5.17 se muestra una gráfica de convergencia de las tres representaciones y puede apreciarse que la representación entera A también tuvo una mejor aptitud promedio; sin embargo la representación entera B tuvo un mejor desempeño global.

D	C	B	A	S
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Tabla 5.22: Tabla de verdad del ejemplo 8.

El desempeño del MGA fue mejor que el de las tres versiones del PSO ya que con un menor número de evaluaciones de la función objetivo encontró circuitos factibles en todas las corridas y el mejor circuito de 7 compuertas lo halló el 15 % de las veces. Las tres versiones del PSO también encontraron un circuito de 7 compuertas pero encontraron circuitos factibles en a lo más el 80 % de las corridas. La aptitud promedio del MGA también fue superior a las tres versiones del PSO consiguiendo un número de compuertas promedio de 8.90. Sin embargo, el desempeño del NGA fue bastante pobre. La expresión booleana encontrada por el diseñador humano fue $S = ((D'B) \oplus (A'C')) + ((B'A)(D \oplus C'))$ que tiene 11 compuertas y se obtuvo mediante mapas de Karnaugh. Esta expresión fue reportada por Coello [9].

Corrida	Binaria			Entera A			Entera B		
	AP	MA	NC	AP	MA	NC	AP	MA	NC
0	14.57	32	9	14.91	15	*	14.81	15	*
1	14.77	33	8	30.01	33	8	26.03	34	7
2	14.00	15	*	14.60	15	*	17.38	32	9
3	14.07	15	*	14.94	15	*	21.04	33	8
4	13.74	15	*	23.95	33	8	19.46	33	8
5	13.76	15	*	14.13	15	*	22.21	33	8
6	14.63	33	8	14.90	15	*	19.88	31	10
7	14.40	29	12	31.37	33	8	16.33	32	9
8	15.69	30	11	31.65	34	7	19.90	33	8
9	13.93	15	*	31.89	33	8	14.37	15	*
10	13.96	15	*	30.00	34	7	18.17	30	11
11	15.11	33	8	13.97	14	*	14.60	15	*
12	14.72	32	9	28.98	32	9	14.53	31	10
13	13.85	15	*	28.46	32	9	21.07	33	8
14	13.84	15	*	14.91	15	*	16.57	30	11
15	14.17	33	8	14.85	15	*	20.21	33	8
16	15.54	34	7	29.90	33	8	19.63	32	9
17	15.82	34	7	30.48	32	*	20.42	34	7
18	16.16	33	8	31.19	33	8	14.16	15	*
19	13.84	15	*	25.10	32	*	17.04	30	11

Tabla 5.23: Análisis comparativo de 20 corridas del ejemplo 8.

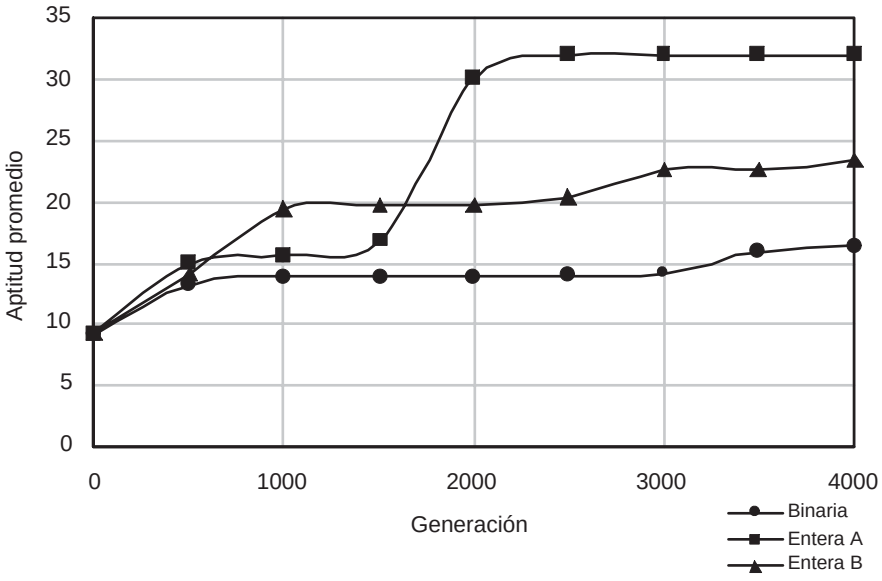
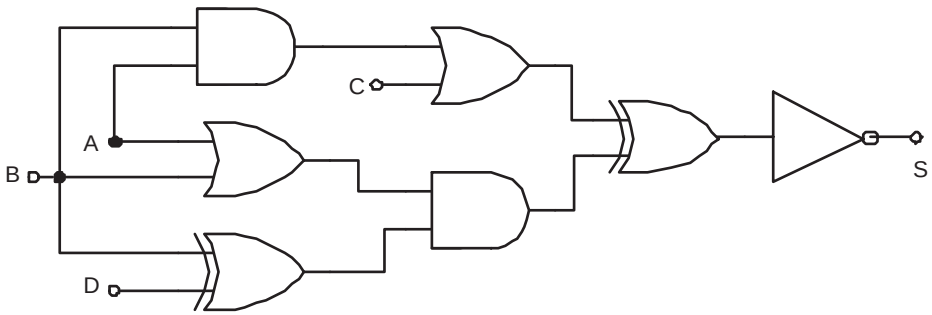


Figura 5.17: Gráfica de convergencia del ejemplo 8 de la corrida ubicada en la mediana.

Parámetros	PSO B	PSO EA	PSO EB	MGA	NGA	DH
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE					
Matriz	5×5					-
Tamaño de la población	50			170		-
Número de generaciones	4000			400		-
Evaluaciones	200,000			68,000		-
Número de corridas	20					-
$\phi_1 = \phi_2$	0.8	0.2		-		-
Vmax	3.0	0.4		-		-
Pm	10 %			0.6667 %		-
Pc	-			50 %		-
Tamaño del vecindario	3			-		-
Cardinalidad	3					-
Número de compuertas de la mejor solución	7	7	7	7	10	11
Mejor solución	10 %	10 %	10 %	15 %	5 %	-
Circuitos factibles	55 %	60 %	80 %	100 %	5 %	-
Número de compuertas promedio	17.50	15.35	12.30	8.90	25.45	-
Aptitud promedio	24.55	25.65	28.70	32.10	15.55	-
Desviación estándar	8.9353	9.0454	7.1311	1.2523	3.6774	-

Tabla 5.24: Comparación de resultados entre el PSO, un diseñador humano, el MGA y el NGA para el ejemplo 8.



$$S = ((C + BA) \oplus (D \oplus B)(B + A))'$$

Figura 5.18: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 8.

5.10. Ejemplo 9

Este ejemplo fue tomado del archivo blif llamado *Roth-2ip* cuya tabla de verdad se muestra en la tabla 5.25.

E	D	C	B	A	S
0	0	0	0	0	0
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	0
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	0
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

Tabla 5.25: Tabla de verdad del ejemplo 9.

Este ejemplo tiene una complejidad mayor ya que tiene cinco entradas y una salida por lo que se llevaron a cabo más evaluaciones que en los circuitos anteriores para encontrar los resultados que son mostrados en la tabla 5.26. Para este ejemplo la representación binaria no tuvo un buen desempeño ya que no encontró ninguna solución factible mientras que las representaciones enteras encontraron soluciones factibles en poco más de la mitad de las corridas (tabla 5.27). El mejor circuito encontrado por las representaciones enteras fue de 10 compuertas (figura 5.20) y fue hallado más frecuentemente por la representación entera A que la B. La gráfica de convergencia de

la figura 5.19 muestra un comportamiento distinto al de los otros ejemplos ya que la representación entera A converge más lentamente que la entera B además de que esta última tiene un mejor desempeño global.

En este ejemplo la versión binaria del PSO no produjo ningún circuito factible. Las versiones enteras del PSO lograron mejores resultados que el MGA ya que a pesar de que las tres técnicas encontraron un circuito con 10 compuertas, el MGA sólo lo logró el 5 % de las veces, mientras que el PSO entero A lo consiguió el 10 % del tiempo y el PSO entero B en el 25 % de las corridas. Por otro lado las representaciones enteras del PSO encontraron circuitos factibles en más de las mitad de las corridas mientras que el MGA sólo produjo circuitos factibles el 25 % de las veces. La aptitud promedio de la representación entera B del PSO fue de 44.85 mientras que la aptitud promedio del MGA fue de tan sólo 35.60. La expresión booleana encontrada por el diseñador humano fue $S = (C \oplus B)(E'(D \oplus A) + A'(E \oplus D)) + (B \oplus A)(C'(E \oplus D) + D'(E \oplus C)) + EC'(B'(A \oplus D))$ que tiene 23 compuertas y fue obtenido por la autora de esta tesis mediante álgebra booleana.

Corrida	**Binaria**			**Entera A**			**Entera B**		
	AP	**MA**	**NC**	**AP**	**MA**	**NC**	**AP**	**MA**	**NC**
0	26.27	29	*	56.73	58	10	42.06	57	11
1	27.01	28	*	50.92	57	11	45.07	58	10
2	26.33	29	*	29.10	30	*	35.34	56	12
3	26.90	30	*	48.42	58	10	29.32	31	*
4	26.57	28	*	40.46	57	11	35.66	55	13
5	26.95	29	*	48.43	54	14	42.14	57	11
6	26.72	28	*	35.82	56	12	28.65	30	*
7	26.78	28	*	30.34	31	*	29.61	31	*
8	26.78	30	*	30.37	31	*	30.27	31	*
9	26.59	29	*	48.27	54	14	34.50	55	13
10	27.23	30	*	29.07	30	*	29.07	31	*
11	26.82	29	*	28.91	29	*	31.01	54	14
12	26.52	28	*	29.62	30	*	43.34	58	10
13	26.42	28	*	30.02	31	*	28.45	31	*
14	26.55	31	*	53.11	56	12	28.79	30	*
15	26.58	28	*	44.48	57	11	28.96	30	*
16	26.96	28	*	27.95	28	*	39.28	55	13
17	27.35	31	*	30.66	31	*	36.95	58	10
18	26.51	29	*	33.26	55	13	29.15	31	*
19	26.56	29	*	30.73	31	*	39.76	58	10

Tabla 5.26: Análisis comparativo de 20 corridas del ejemplo 9.

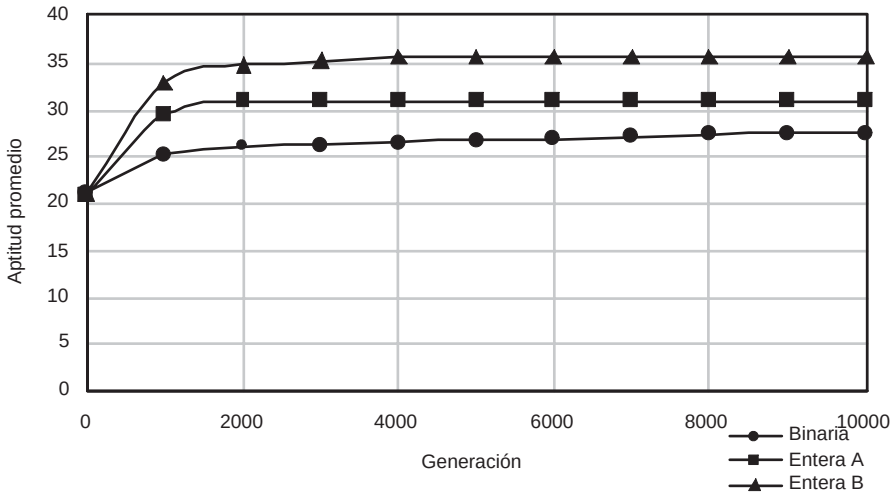
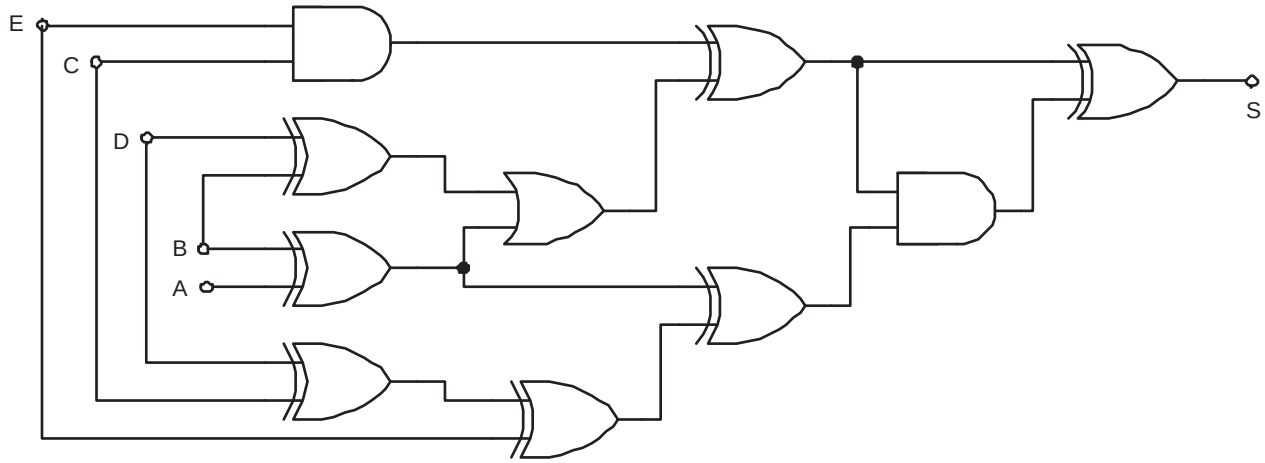


Figura 5.19: Gráfica de convergencia del ejemplo 9 de la corrida ubicada en la mediana.

Parámetros	PSO B	PSO EA	PSO EB	MGA	DH
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE				
Matriz	6×6				-
Tamaño de la población	50			330	-
Número de generaciones	10000			1600	-
Evaluaciones	500,000			528,000	-
Número de corridas	20				-
$\phi_1 = \phi_2$	0.8	0.2		-	-
Vmax	3.0	0.4		-	-
Pm	10 %			0.6667 %	-
Pc	-			50 %	-
Tamaño del vecindario	3			-	-
Cardinalidad	3			5	-
Número de compuertas de la mejor solución	*	10	10	10	23
Mejor solución	0 %	10 %	25 %	5 %	-
Circuitos factibles	0 %	50 %	55 %	25 %	-
Número de compuertas promedio	*	24.80	23.15	32.40	-
Aptitud promedio	28.95	43.20	44.85	35.60	-
Desviación estándar	0.9986	13.3952	13.2119	11.8028	-

Tabla 5.27: Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 9.



$$S = (((B \oplus D) + (A \oplus B)) \oplus CE) \oplus (((B \oplus D) + (A \oplus B)) \oplus CE)((E \oplus (D \oplus C)) \oplus (A \oplus B))$$

Figura 5.20: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 9.

Capítulo 6

Circuitos de múltiples salidas

El diseño de circuitos lógicos combinatorios de una salida es distinto al diseño de circuitos de múltiples salidas debido a que para estos últimos es necesario diseñar cada salida por separado y después localizar las compuertas que puedan utilizarse en más de una salida.

Para resolver el problema del diseño de circuitos de múltiples salidas lo que se hizo fue que en lugar de usar una matriz para el diseño de cada salida y después buscar la reutilización de compuertas se combinaron todas las salidas en la misma matriz de forma que cada salida pudiera utilizar cualquier celda para su implementación consiguiendo con ello la reutilización de bloques funcionales completos de compuertas.

El algoritmo básico utilizado en este trabajo para el diseño de circuitos de una y múltiples salidas es el mismo. Sin embargo, al aumentar el número de salidas se incrementa la complejidad del circuito y por ello se requirieron más evaluaciones de la función objetivo que las evaluaciones necesitadas para el diseño de circuitos de una sola salida. Al igual que para los circuitos de una salida se llevaron a cabo 20 corridas independientes de cada una de las representaciones (binaria, entera A y entera B) y las comparaciones con respecto al algoritmo genético de cardinalidad N (NGA) [6] y el algoritmo genético multiobjetivo (MGA) [9].

6.1. Ejemplo 1

El ejemplo 1 fue tomado del archivo blif *ieee_adder_2ip* y es un circuito de cuatro entradas y dos salidas cuya tabla de verdad se muestra en la tabla 6.1. En las tablas 6.2 y 6.3 se puede ver que la representación binaria encontró circuitos de 7 compuertas (figura 6.2) en varias de sus corridas además de que se encontraron circuitos factibles el 95 % de las veces. Sin embargo, algunos de ellos contienen muchas más compuertas que el mejor circuito encontrado. Por otro lado la representación entera A también encontró circuitos de 7 compuertas pero en más corridas que la representación binaria aunque sólo encontró circuitos factibles el 70 % de las veces.

D	**C**	**B**	**A**	S_0	S_1
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	1
1	1	1	1	0	1

Tabla 6.1: Tabla de verdad del ejemplo 1.

En este ejemplo la representación que tuvo el mejor desempeño fue la representación entera B ya que encontró siempre circuitos factibles. Además, encontró circuitos de 7 compuertas en más de la mitad de las corridas. En la figura 6.1 se muestra una gráfica de convergencia comparativa (de la corrida en la mediana) de las tres representaciones. Aquí puede observarse que la representación entera A fue la que convergió más rápido al igual que en la mayoría de los ejemplos de los circuitos de una salida aunque no tuvo el mejor desempeño global.

En este experimento las tres versiones del PSO tuvieron un mejor desempeño que el MGA ya que a pesar de que todas las técnicas encontraron un circuitos de 7 compuertas, el PSO binario lo encontró el 30 % de las veces, el PSO entero A y el 40 % y el PSO entero B en 60 % de las corridas mientras

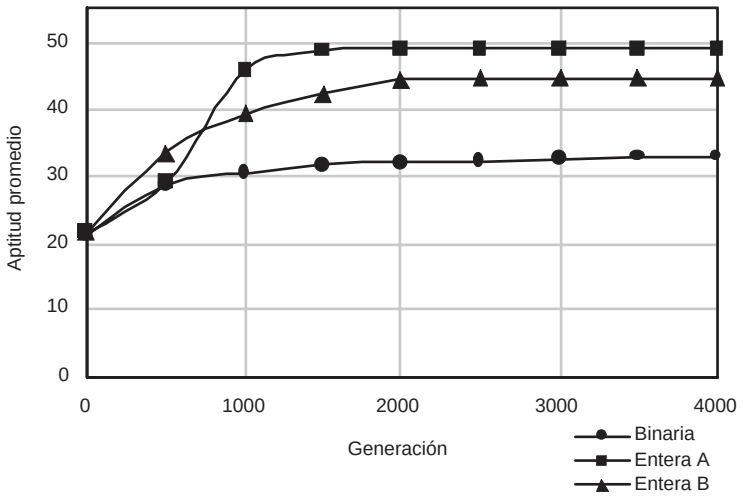


Figura 6.1: Gráfica de convergencia del ejemplo 1 de la corrida ubicada en la mediana.

Corrida	**Binaria**			**Entera A**			**Entera B**		
	AP[1]	**MA**[2]	**NC**[3]	**AP**	**MA**	**NC**	**AP**	**MA**	**NC**
0	30.15	47	10	30.88	31	*	36.53	48	9
1	30.30	49	8	30.91	49	8	42.16	50	7
2	31.49	49	8	47.90	50	7	35.85	49	8
3	31.03	50	7	30.81	31	*	35.93	47	10
4	30.85	47	10	40.94	50	7	45.36	49	8
5	30.82	50	7	30.24	31	*	39.93	50	7
6	31.16	49	8	46.97	50	7	38.45	50	7
7	32.29	50	7	48.57	50	7	44.82	50	7
8	29.53	42	15	48.13	50	7	44.78	50	7
9	31.07	50	7	30.77	31	*	39.96	49	8
10	29.98	45	12	43.72	47	10	40.47	50	7
11	31.02	50	7	30.91	31	*	44.98	50	7
12	31.06	50	7	35.59	45	12	42.72	50	7
13	29.78	48	9	48.10	50	7	44.24	50	7
14	30.48	47	10	47.40	50	7	34.37	50	7
15	30.62	45	12	48.80	50	7	40.05	50	7
16	29.23	31	*	44.66	49	8	44.28	48	9
17	29.74	46	11	30.74	31	*	44.42	49	8
18	30.20	47	10	46.49	48	9	47.53	50	7
19	29.82	47	10	44.60	47	10	33.61	46	11

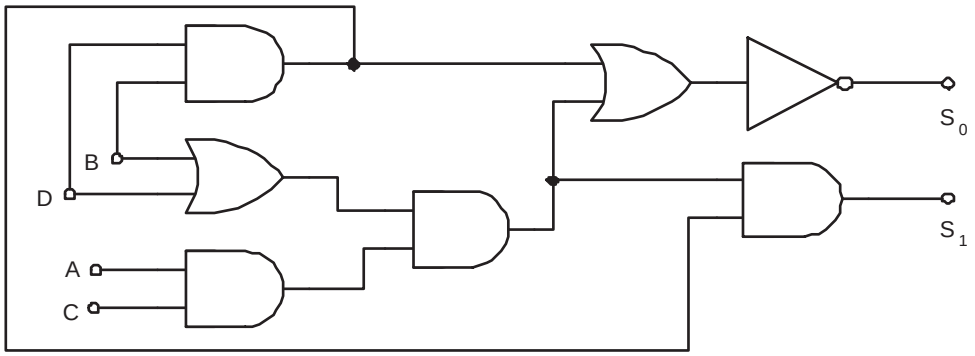
[1]Aptitud promedio [2]Mejor aptitud [3]Número de compuertas

Tabla 6.2: Análisis comparativo de 20 corridas del ejemplo 1.

que el MGA sólo produjo este circuito 25 % del tiempo. Por otro lado, la representación entera B del PSO y la versión binaria encontraron circuitos factibles en al menos 95 % de las corridas y el MGA lo logró sólo en el 70 %. Por último, la aptitud promedio del MGA fue menor que la aptitud promedio del PSO entero B pues mientras el número de compuertas promedio del PSO entero B fue de 7.75, del MGA fue de 13.40. Sin embargo la aptitud promedio del MGA fue ligeramente mayor que el PSO entero A. La expresión booleana encontrada por el diseñador humano fue $S_0 = B'D' + C'A'(D' + B')$ y $S_1 = BD(A+C)$ que en total requiere de 12 compuertas y se obtuvo mediante mapas de Karnaugh por la autora de esta tesis.

Parámetros	**PSO B**	**PSO EA**	**PSO EB**	**MGA**	**DH**
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE				
Matriz	5×5				-
Tamaño de la población	50			330	-
Número de generaciones	4000			610	-
Evaluaciones	200,000			201,300	-
Número de corridas	20				-
$\phi_1 = \phi_2$	0.8	0.2		-	-
Vmax	3.0	0.4		-	-
Pm	10 %			0.6667 %	-
Pc	-			50 %	-
Tamaño del vecindario	3			-	-
Cardinalidad	3			5	-
Número de compuertas de la mejor solución	7	7	7	7	12
Mejor solución	30 %	40 %	60 %	25 %	-
Circuitos factibles	95 %	70 %	100 %	75 %	-
Número de compuertas promedio	10.05	13.45	7.75	13.40	-
Aptitud promedio	46.95	43.55	49.25	43.60	-
Desviación estándar	4.33	8.53	1.16	8.09	-

Tabla 6.3: Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 1.



$$S_0 = ((CA)(B + D) + BD)'$$
$$S_1 = (CA)(B + D)(BD)$$

Figura 6.2: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 1.

6.2. Ejemplo 2

El ejemplo 2 es un sumador de dos bits y es un circuito de cuatro entradas y tres salidas S_0,S_1 y S_2 siendo S_0 el bit más significativo y cuya tabla de verdad se muestra en la tabla 6.4.

D	**C**	**B**	**A**	S_0	S_1	S_2
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Tabla 6.4: Tabla de verdad del ejemplo 2.

En las tablas 6.5 y 6.6 se puede ver que la representación binaria no encontró ningún circuito factible mientras que las representaciones enteras

A y B encontraron circuitos factibles en al menos la mitad de las corridas. Por otro lado, ambas representaciones encontraron un circuito con 7 compuertas (figura 6.2) en al menos dos de las 20 corridas. Cabe hacer notar que la representación binaria en este ejemplo no encontró ningún circuito válido principalmente porque requiere de un mayor número de evaluaciones de la función objetivo que las otras dos representaciones para encontrar resultados satisfactorios. Finalmente, en la figura 6.3 se muestra una gráfica de convergencia en donde puede observarse que el comportamiento de las representaciones enteras A y B fue muy similar mientras que la binaria se quedó por debajo de ellas.

Para este ejemplo, la representación binaria del PSO no encontró ningún circuito factible. El mejor circuito encontrado por las representaciones enteras del PSO requiere de 7 compuertas para su implementación mientras que el mejor circuito encontrado por el MGA necesita 8 compuertas es decir una compuerta más que las representación entera A y la entera B del PSO. A pesar de que el MGA no encontró el circuito de 7 compuertas, produjo circuitos factibles en el 65 % de las corridas y las representaciones enteras A y B lo lograron el 50 % de las veces. La aptitud promedio de estos tres enfoques fue muy similar, teniendo entre 7 y 9 compuertas promedio para la imple-

Corrida	**Binaria**			**Entera A**			**Entera B**		
	AP	**MA**	**NC**	**AP**	**MA**	**NC**	**AP**	**MA**	**NC**
0	41.05	46	*	47.91	61	12	44.83	47	*
1	40.84	47	*	56.29	66	7	55.66	64	9
2	40.47	46	*	53.70	65	8	43.69	47	*
3	39.73	44	*	49.00	59	14	50.75	66	7
4	41.08	45	*	45.38	47	*	46.65	65	8
5	40.61	46	*	48.58	65	8	44.37	47	*
6	41.06	47	*	46.05	47	*	44.28	47	*
7	41.59	45	*	45.50	47	*	45.04	47	*
8	40.40	44	*	46.17	47	*	45.39	47	*
9	39.97	44	*	52.39	65	8	50.83	64	9
10	40.20	43	*	52.43	62	11	45.21	47	*
11	41.09	46	*	49.10	64	9	45.39	47	*
12	40.88	46	*	44.51	46	*	56.57	66	7
13	41.54	45	*	49.39	60	13	45.22	61	12
14	40.74	44	*	58.84	66	7	58.45	66	7
15	40.01	44	*	45.69	47	*	49.82	65	8
16	39.77	46	*	46.18	47	*	45.26	64	9
17	41.02	46	*	54.31	63	10	45.75	47	*
18	40.45	44	*	44.03	46	*	44.91	47	*
19	41.01	44	*	45.68	47	*	45.95	64	9

Tabla 6.5: Análisis comparativo de 20 corridas del ejemplo 2.

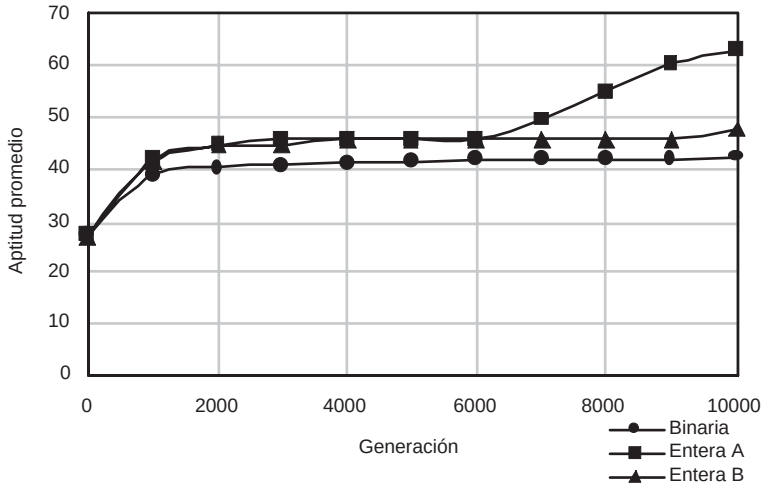
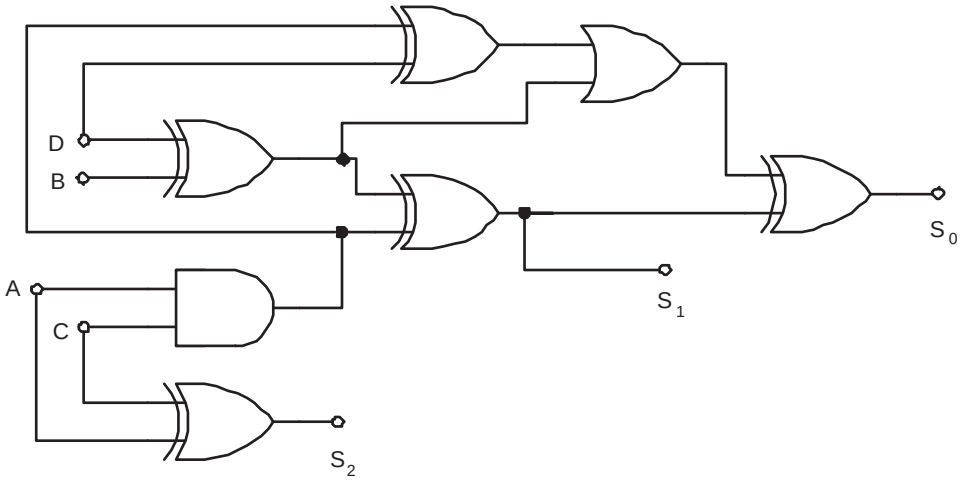


Figura 6.3: Gráfica de convergencia del ejemplo 2 de la corrida ubicada en la mediana.

Parámetros	**PSO B**	**PSO EA**	**PSO EB**	**MGA**	**DH**
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE				
Matriz	5×5				-
Tamaño de la población	50			490	-
Número de generaciones	10,000			1022	-
Evaluaciones	500,000			500,780	-
Número de corridas	20				-
$\phi_1 = \phi_2$	0.8	0.2		-	-
Vmax	3.0	0.4		-	-
Pm	10 %			0.6667 %	-
Pc	-			50 %	-
Tamaño del vecindario	3			-	-
Cardinalidad	3			5	-
Número de compuertas de la mejor solución	*	7	7	8	11
Mejor solución	0 %	10 %	15 %	5 %	-
Circuitos factibles	0 %	55 %	50 %	65 %	-
Número de compuertas promedio	*	17.15	17.25	17.15	-
Aptitud promedio	45.1	55.85	55.75	55.85	-
Desviación estándar	1.17	8.61	9.04	7.63	-

Tabla 6.6: Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 2.

mentación del circuito. La expresión booleana encontrada por el diseñador humano fue $S_0 = (AC)(B \oplus D) + BD$, $S_1 = C'(B \oplus D) + C(A \oplus (B \oplus D))$ y $S_2 = A \oplus C$ que en total requiere de 11 compuertas y se obtuvo mediante mapas de Karnaugh por la autora de esta tesis.



$$S_0 = (AC \oplus (B \oplus D)) \oplus ((D \oplus AC) + (B \oplus D))$$
$$S_1 = AC \oplus (B \oplus D)$$
$$S_2 = C \oplus A$$

Figura 6.4: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 2.

6.3. Ejemplo 3

El ejemplo 3 fue tomado del archivo blif *planar_11* que es un circuito de cinco entradas y tres salidas cuya tabla de verdad se muestra en la tabla 6.7. En las tablas 6.8 y 6.9 se puede ver que aunque este ejemplo tiene un mayor número de entradas y salidas que el ejemplo anterior, la representación binaria pudo encontrar circuitos factibles además de algunos circuitos con 7 compuertas (figura 6.6) que fue la cantidad menor de compuertas necesarias para la implementación de este circuito.

Por otro lado, la representación entera A y B lograron que la mitad de las corridas encontraran circuitos factibles y que en cuatro de las veinte corridas

se tuviera un circuito con 7 compuertas. En la figura 6.5 se puede apreciar que las tres representaciones convergieron casi al mismo tiempo a pesar que el desempeño global de las representaciones enteras fue mejor que el de la representación binaria.

E	**D**	**C**	**B**	**A**	S_0	S_1	S_2
0	0	0	0	0	1	1	0
0	0	0	0	1	1	0	0
0	0	0	1	0	1	1	0
0	0	0	1	1	1	0	0
0	0	1	0	0	1	1	1
0	0	1	0	1	1	0	1
0	0	1	1	0	1	1	0
0	0	1	1	1	1	1	0
0	1	0	0	0	1	1	0
0	1	0	0	1	1	0	0
0	1	0	1	0	1	1	0
0	1	0	1	1	1	0	0
0	1	1	0	0	1	1	1
0	1	1	0	1	1	0	1
0	1	1	1	0	1	1	0
0	1	1	1	1	1	1	0
1	0	0	0	0	0	1	0
1	0	0	0	1	0	0	0
1	0	0	1	0	0	1	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	1	1
1	0	1	0	1	0	0	1
1	0	1	1	0	0	1	0
1	0	1	1	1	0	1	0
1	1	0	0	0	0	1	0
1	1	0	0	1	0	0	0
1	1	0	1	0	0	1	0
1	1	0	1	1	0	0	0
1	1	1	0	0	1	1	1
1	1	1	0	1	1	0	1
1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	0

Tabla 6.7: Tabla de verdad del ejemplo 3.

En este ejemplo el desempeño del MGA fue ligeramente mejor que el de las tres representaciones del PSO, principalmente porque se encontraron circuitos factibles en el 65 % de las corridas mientras que el PSO entero B lo hizo en el 50 %. El mejor circuito encontrado por las 4 técnicas requiere de 7 compuertas y fue encontrado el 20 % de las veces por el MGA y las representaciones enteras del PSO y sólo el 10 % del tiempo por la versión binaria. La aptitud promedio del MGA fue ligeramente mejor que la aptitud promedio del PSO entero A ya que la primera requirió 17.05 compuertas promedio y la segunda 18.65.

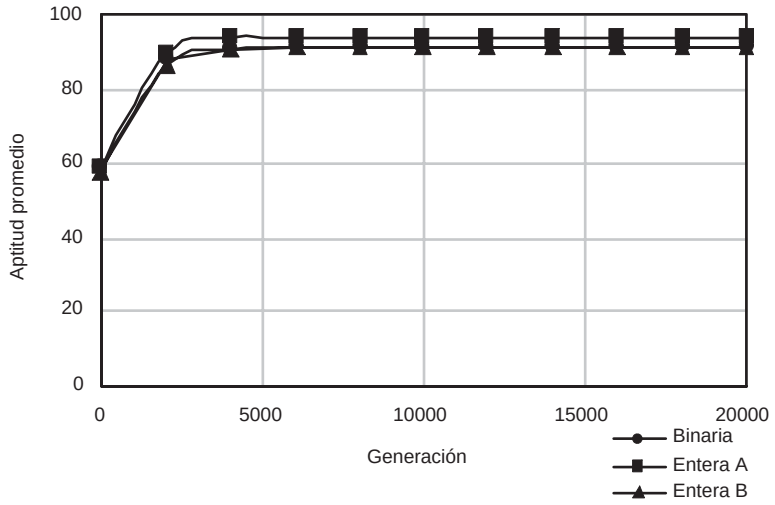


Figura 6.5: Gráfica de convergencia del ejemplo 3 de la corrida ubicada en la mediana.

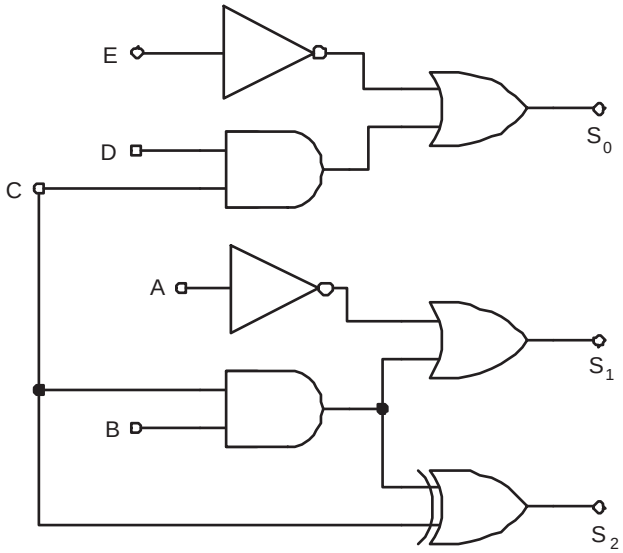
Corrida	Binaria			Entera A			Entera B		
	AP	MA	NC	AP	MA	NC	AP	MA	NC
0	88.09	92	*	103.50	114	7	89.04	92	*
1	89.98	111	10	110.88	112	9	89.45	92	*
2	84.99	92	*	91.77	92	*	89.27	92	*
3	85.44	109	12	91.75	92	*	99.71	114	7
4	88.61	92	*	104.40	114	7	94.06	114	7
5	89.20	92	*	112.04	114	7	89.52	92	*
6	96.28	114	7	109.07	114	7	103.44	113	8
7	96.27	114	7	90.86	92	*	91.53	92	*
8	87.49	92	*	91.82	92	*	86.13	88	*
9	90.87	92	*	96.97	113	8	102.82	114	7
10	88.60	92	*	107.65	113	8	94.58	110	11
11	97.59	113	8	91.80	110	11	90.14	112	9
12	89.62	92	*	87.87	88	*	90.57	92	*
13	88.87	92	*	111.95	113	8	90.50	92	*
14	86.04	92	*	110.56	112	9	97.18	112	9
15	87.62	92	*	91.84	92	*	92.97	111	10
16	87.45	92	*	88.74	92	*	96.20	114	7
17	86.90	92	*	91.47	92	*	87.14	88	*
18	88.46	92	*	93.61	94	*	88.83	92	*
19	88.91	92	*	91.35	92	*	90.50	92	*

Tabla 6.8: Análisis comparativo de 20 corridas del ejemplo 3.

En este ejemplo no se hace la comparación entre los resultados de las heurísticas y los resultados obtenidos por el diseñador humano pues al ser un circuito con una mayor cantidad de entradas y salidas que los circuitos anteriores, el método de mapas de Karnaugh y el método de Quine McCluskey (por no mencionar la simplificación algebraica) se complican demasiado logrando que la expresión simplificada obtenida mediante estos métodos implique un gran número de compuertas y entradas a ellas.

Parámetros	**PSO B**	**PSO EA**	**PSO EB**	**MGA**
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE			
Matriz	5×5			
Tamaño de la población	50			970
Número de generaciones	20,000			1032
Evaluaciones	1,000,000			1,101,040
Número de corridas	20			
$\phi_1 = \phi_2$	0.8	0.2		-
Vmax	3.0	0.4		-
Pm	10 %			0.6667 %
Pc	-			50 %
Tamaño del vecindario	3			-
Cardinalidad	3			5
Número de compuertas de la mejor solución	7	7	7	7
Mejor solución	10 %	20 %	20 %	20 %
Circuitos factibles	25 %	50 %	45 %	65 %
Número de compuertas promedio	23.95	18.65	20.10	17.05
Aptitud promedio	97.05	102.35	100.90	103.95
Desviación estándar	9.03	10.91	11.03	10.46

Tabla 6.9: Comparación de resultados entre el PSO, un diseñador humano y el MGA para el ejemplo 3.



$$S_0 = E' + CD$$
$$S_1 = CB + A'$$
$$S_2 = C \oplus CB$$

Figura 6.6: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 3.

6.4. Ejemplo 4

El ejemplo 4 es un multiplicador de dos bits que consta de cuatro entradas y cuatro salidas S_0, S_1, S_2 y S_3 siendo S_0 el bit más significativo y cuya tabla de verdad se muestra en la tabla 6.10. En las tablas 6.11 y 6.12 se puede ver que para este ejemplo la representación binaria tampoco logró encontrar circuitos factibles mientras que las representaciones enteras lo consiguieron en un 30 % de las veces además de encontrar un circuito con 7 compuertas (figura 6.8) en dos de las veinte corridas.

Los resultados nos muestran que el algoritmo no tuvo tan buen desempeño como en ejemplos anteriores a pesar de que se han diseñado circuitos de un mayor número de entradas que éste. En la figura 6.7 se puede apreciar que las tres representaciones convergieron casi al mismo tiempo a pesar de que el desempeño de la representación binaria fue pobre en comparación con las representaciones enteras.

D	**C**	**B**	**A**	S_0	S_1	S_2	S_3
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

Tabla 6.10: Tabla de verdad del ejemplo 4.

La representación binaria de PSO no encontró ningún circuito factible para este circuito. Las representaciones enteras del PSO, el MGA y el NGA encontraron un circuito que requiere de tan solo 7 compuertas para su implementación. Sin embargo, las cuatro técnicas sólo lo produjeron en a lo más 15 % de las corridas. La principal diferencia en el desempeño de las técnicas se refiere a la producción de circuitos factibles ya que el MGA lo consiguió en todas las corridas mientras que las representaciones enteras del PSO y el NGA lo consiguieron en alrededor del 30 % de las veces. La aptitud promedio del MGA fue superior a las otras técnicas pues se necesitaron sólo 8.60 compuertas promedio para la implementación del circuito mientras que las otras tres técnicas requirieron de al menos 20 compuertas. La expresión booleana encontrada por el diseñador humano fue $S_0 = (DC)(BA)$, $S_1 = (DB)(CA)'$, $S_2 = CB \oplus DA$ y $S_3 = CA$ que en total requiere de 8 compuertas. Esta expresión fue reportada por Coello [9].

Corrida	Binaria			Entera A			Entera B		
	AP	MA	NC	AP	MA	NC	AP	MA	NC
0	57.45	61	*	62.43	63	*	73.60	82	7
1	57.78	61	*	62.85	63	*	60.02	61	*
2	57.85	60	*	60.81	61	*	60.00	61	*
3	58.52	61	*	63.54	80	9	66.72	82	7
4	58.36	60	*	60.93	61	*	60.25	62	*
5	58.03	61	*	60.56	61	*	61.13	62	*
6	58.20	61	*	61.02	62	*	64.67	82	7
7	57.59	60	*	61.83	62	*	60.90	62	*
8	57.89	61	*	60.88	61	*	60.97	73	16
9	57.63	61	*	61.73	62	*	63.53	78	11
10	58.03	61	*	79.02	82	7	65.56	80	9
11	57.90	60	*	60.76	61	*	59.18	61	*
12	58.45	61	*	60.92	61	*	61.14	62	*
13	58.00	61	*	77.20	81	8	60.29	62	*
14	57.70	61	*	79.43	81	8	60.37	61	*
15	58.73	60	*	76.07	82	7	61.16	62	*
16	58.43	62	*	60.64	61	*	60.38	61	*
17	58.12	60	*	70.72	78	11	60.78	61	*
18	58.53	61	*	60.88	61	*	60.05	61	*
19	58.22	61	*	62.23	63	*	61.15	63	*

Tabla 6.11: Análisis comparativo de 20 corridas del ejemplo 4.

Parámetros	PSO B	PSO EA	PSO EB	MGA	NGA	DH
Conjunto de compuertas	AND,OR,NOT,XOR y WIRE					
Matriz	5×5					-
Tamaño de la población	50			650		-
Número de generaciones	15,000			500		-
Evaluaciones	750,000			350,000		-
Número de corridas	20					-
$\phi_1 = \phi_2$	0.8	0.2		-		-
Vmax	3.0	0.4		-		-
Pm	10 %			0.6667 %		-
Pc	-			50 %		-
Tamaño del vecindario	3			-		-
Cardinalidad	3			5		-
Número de compuertas de la mejor solución	*	7	7	7	9	8
Mejor solución	0 %	10 %	15 %	15 %	10 %	-
Circuitos factibles	0 %	30 %	35 %	100 %	30 %	-
Número de compuertas promedio	*	21.65	22.05	8.60	22.35	-
Aptitud promedio	60.75	67.35	66.95	80.40	66.65	-
Desviación estándar	0.55	9.00	8.64	1.14	7.63	-

Tabla 6.12: Comparación de resultados entre el PSO, un diseñador humano, el MGA y el NGA para el ejemplo 4.

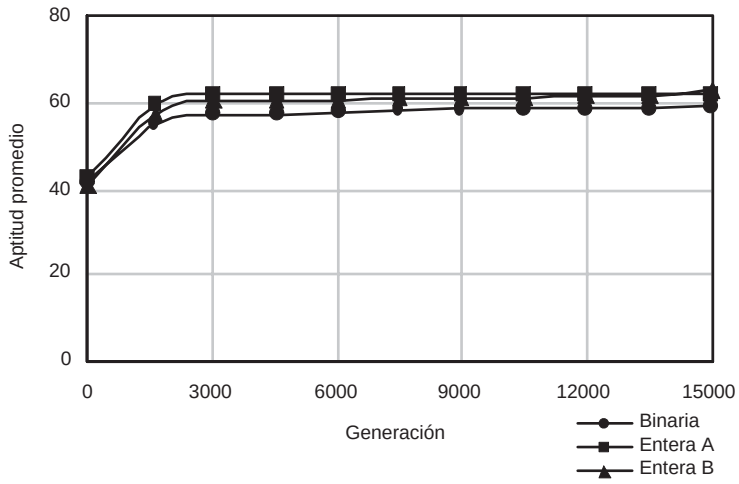
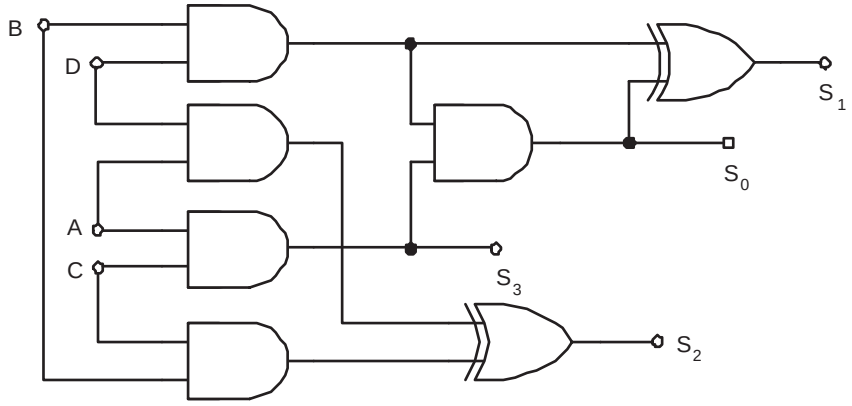


Figura 6.7: Gráfica de convergencia del ejemplo 4 de la corrida ubicada en la mediana.



$$S_0 = (CA)(DB)$$
$$S_1 = DB \oplus (CA)(DB)$$
$$S_2 = DA \oplus BC$$
$$S_3 = CA$$

Figura 6.8: Diagrama y ecuación lógica del mejor circuito encontrado para el ejemplo 4.

Capítulo 7

Conclusiones y trabajo futuro

Las técnicas evolutivas han demostrado ser herramientas poderosas para resolver problemas de optimización, ya que son capaces de obtener resultados competitivos con respecto a las técnicas usadas tradicionalmente para resolver cada uno de estos problemas.

En este trabajo de tesis se presenta un técnica de hardware evolutivo extrínseco, que utiliza como técnica evolutiva un algoritmo de optimización mediante cúmulos de partículas con tres enfoques distintos de representación, los cuales tienen dos objetivos principales: automatizar el proceso de diseño de circuitos lógicos combinatorios y encontrar el circuito de costo mínimo de acuerdo a una cierta métrica.

Los tres enfoques propuestos representaron distintos retos a lo largo del desarrollo de esta tesis. El primer enfoque fue el algoritmo de optimización mediante cúmulos de partículas con representación binaria tuvo como principal problema el tiempo necesario para poder converger a una solución factible, lo que hizo que esta propuesta requiriera más evaluaciones de la función objetivo para poder competir con los resultados obtenidos por su contraparte entera. Por esta misma razón, en algunos de los experimentos, la representación binaria no logró encontrar ningún circuito factible y en algunos otros quedó muy por debajo del desempeño de la representación entera. El segundo y tercer enfoque fueron los algoritmos de optimización mediante cúmulos de partículas con representación entera (A y B) uno de los principales retos fue conseguir el acoplamiento entre la heurística y la representación pues a pesar de que la representación entera A ya había sido probada con problemas de optimización combinatoria [17], no había sido utilizada para el diseño de circuitos lógicos combinatorios. La representación entera B por su parte,

es una de las principales aportaciones de este trabajo de tesis ya que fue nuestra propuesta la que en términos globales obtuvo mejores resultados en comparación con las otras técnicas presentadas.

Con respecto a los experimentos de circuitos de una salida, la representación binaria tuvo un buen desempeño aunque en la mayoría de los casos estuvo por debajo del desempeño alcanzado por las representaciones enteras las cuales tuvieron un desempeño muy similar. Sin embargo, podríamos decir que la representación entera B logró conseguir más circuitos factibles (y en algunos casos con un menor número de compuertas) que la representación entera A. Por otro lado, la rapidez de convergencia de la representación entera A fue ligeramente mayor de manera general que la representación entera B.

En cuanto al trabajo futuro que se puede hacer en torno a la propuesta actual podemos mencionar que una mejora sería representar los circuitos lógicos combinatorios mediante números reales, lo cual no parece natural y quizá sería necesario introducir algunos mecanismos adicionales para conseguirlo. La motivación de utilizar esta representación estriba en que el algoritmo de PSO ha demostrado tener mucho más éxito en resolver problemas de optimización al usar una representación real que con la representación entera o binaria [23]. Por otro lado, se podría trabajar en un enfoque multiobjetivo similar al utilizado en [9] para mejorar las capacidades de búsqueda del algoritmo.

Con respecto al algoritmo de PSO, pueden introducirse algunos otros operadores como el operador de selección propuesto por Angeline en [2] además de introducir también operadores de mutación más sofisticados para mejorar el desempeño del algoritmo. Por último, se podría hacer un estudio más detallado del impacto de los parámetros del PSO en la obtención de resultados [38] para poder seleccionarlos adecuadamente.

En esta propuesta se presentó una codificación de los circuitos mediante una matriz de tamaño fijo que trata de mantener un compromiso entre la poderosa representación de árbol de la programación genética y la relativamente débil representación lineal utilizada comúnmente. Sin embargo, esta representación requiere de un trabajo adicional para interpretar las soluciones que está codificando. Por ello una propuesta viable sería una representación de matriz dinámica con la estructura general de la representación de árbol de la programación genética. Esta propuesta tiene un problema similar al problema de la representación entera usada en esta tesis, que es el del acoplamiento entre el PSO y la representación. Con la representación de árbol es necesario

modificar el algoritmo de PSO para poder llevar a cabo la manipulación de los individuos y lograr que puedan llegar a resultados satisfactorios. Sin embargo, hasta la fecha no se han reportado trabajos en la literatura especializada en donde se intente utilizar el algoritmo de PSO con una representación de árbol.

Por último, es importante mencionar que la propuesta presentada en este trabajo de tesis es el primer intento de utilizar el algoritmo de optimización mediante cúmulos de partículas para diseñar circuitos lógicos combinatorios. El algoritmo presentado parece prometedor ya que produjo resultados competitivos y en la mayoría de los casos mejores en comparación con las heurísticas contra las que se le comparó.

Bibliografía

[1] Peter J. Angeline. Evolutionary optimization versus particle swarm optimization: philosophy and performance differences. In Waagen D. Porto V.W., SaravananÑ. and Eiben A.E., editors, *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 611–618. Springer, 1998.

[2] Peter J. Angeline. Using selection to improve particle swarm optimization. In Saravanan N. Waagen D. y Eiben A.E. Porto V.W., editor, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1998)*, pages 84–89, Anchorage, Alaska, USA., 1998.

[3] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, 1999.

[4] Bill P. Buckles, Arturo Hernández Aguirre, and Carlos Coello Coello. Circuit design using genetic programming: An illustrative study. In *Proceedings of the 10th NASA Symposium on VLSI Design*, pages 4.1–1–4.1–10, Albuquerque NM, 2002.

[5] Carlos A. Coello Coello. *Introducción a la computación evolutiva (Notas de curso)*. CINVESTAV-IPN, Sección de Computación, Departamento de Ingeniería Eléctrica, 2003.

[6] Carlos A. Coello Coello, Alan D. Christiansen, 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*, pages 335–338. Springer-Verlag, University of East Anglia, England, April 1997.

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

[8] Carlos A. Coello Coello, Alan D. Christiansen, and Arturo Hernández Aguirre. Towards Automated Evolutionary Design of Combinational Circuits. *Computers and Electrical Engineering. An International Journal*, 27(1):1–28, January 2001.

[9] Carlos A. Coello Coello, Arturo Hernández Aguirre, and Bill P. Buckles. Evolutionary Multiobjective Design of Combinational Logic Circuits. In Jason Lohn, Adrian Stoica, Didier Keymeulen, and Silvano Colombano, editors, *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware*, pages 161–170. IEEE Computer Society, Los Alamitos, California, July 2000.

[10] Carlos A. Coello Coello, Erika Hernández Luna, and Arturo Hernández Aguirre. Use of particle swarm optimization to design combinational logic circuits. In Pauline C. Haddow Andy M. Tyrell and Jim Torresen, editors, *Evolvable Systems: From Biology to Hardware. 5th International Conference, ICES 2003*, pages 398–409, Trondheim, Norway, 2003. Springer, Lecture Notes in Computer Science Vol. 2606.

[11] Ronald C. Emery. *Digital circuits : logic and design.* Marcel Dekker, New York, 1985.

[12] Andries P. Engelbrecht. *Computational intelligence.* Wiley, 2002.

[13] David B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence.* The Institute of Electrical and Electronic Engineers, New York, 1995.

[14] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading, MA, 1989.

[15] Venu G. Gudise and Ganesh K. Venayagamoorthy. Evolving digital circuits using particle swarm. In *Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*, pages 468 – 472, Portland, OR, USA, 2003.

[16] John H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Harbor : University of Michigan Press, 1975.

[17] Xiaohui Hu, Russell C. Eberhart, and Yuhui Shi. Swarm intelligence for permutation optimization: a case study on n-queens problem. In *Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, pages 243–246, Indianapolis, Indiana, USA., 2003.

[18] Charles H. Roth Jr. *Fundamentals of logic design*. West Pub., St. Paul, 1985.

[19] Tatiana Kalganova. A new evolutionary hardware approach for logic design. In Annie S. Wu, editor, *Proc. of the GECCO'99 Student Workshop*, pages 360–361, Orlando, Florida, USA, 1999.

[20] Tatiana Kalganova and Julian Miller. Evolving more efficient digital circuits by allowing circuit layout and multi-objective fitness. In Adrian Stoica, Didier Keymeulen, and Jason Lohn, editors, *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*, pages 54–63, Los Alamitos, California, 1999. IEEE Computer Society Press.

[21] James Kennedy and Russell C. Eberhart. Particle Swarm Optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages 1942–1948, Piscataway, New Jersey, 1995. IEEE Service Center.

[22] James Kennedy and Russell C. Eberhart. A Discrete Binary Version of the Particle Swarm Algorithm. In *Proceedings of the 1997 IEEE Conference on Systems, Man, and Cybernetics*, pages 4104–4109, Piscataway, New Jersey, 1997. IEEE Service Center.

[23] James Kennedy and Russell C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, California, 2001.

[24] Edgar Galván López. Diseño de circuitos lógicos combinatorios utilizando programación genética postfija con adaptación en línea. Master's thesis, Facultad de Física e Inteligencia Artificial, Universidad Veracruzana, Mayo 2002. (Disponible en: `http://delta.cs.cinvestav.mx/~ccoello/`).

[25] Sushil J. Louis. *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Department of Computer Science, Indiana University, August 1993.

[26] Sushil J. Louis and Gregory J. Rawlins. Using Genetic Algorithms to Design Structures. Technical Report 326, Computer Science Department, Indiana University, Bloomington, Indiana, February 1991.

[27] Morris Mano and Charles R. Kime. *Fundamentos de diseño lógico y computadoras*. Prentice Hall, 1998.

[28] Edward J. McCluskey. *Logic design principles : with emphasis on testable semicustom circuits*. Prentice Hall, 1986.

[29] Benito Mendoza García and Carlos A. Coello Coello. Uso del Sistema de la Colonia de Hormigas para el Diseño de Circuitos Lógicos Combinatorios. In E. Alba, F. Fernández, J.A. Gómez, F. Herrera, J.I. Hidalgo, J. Lanchares, J.J. Merelo, and J.M. Sánchez, editors, *Primer Congreso Español de Algoritmos Evolutivos y Bioinspirados (AEB'02)*, pages 294–301, Mérida, España, 2002. Universidad de la Extramadura.

[30] J. F. Miller, P. Thomson, and T. Fogarty. Designing 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 Kaufmann, Chichester, England, 1998.

[31] 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.

[32] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer, New York, 1999.

[33] Eduardo Serna Pérez. Diseño de circuitos lógicos combinatorios usando programación genética. Master's thesis, Facultad de Física e Inteligencia Artificial, Universidad Veracruzana, Febrero 2001. (Disponible en: `http://delta.cs.cinvestav.mx/~ccoello/`).

[34] Colin B. Reeves. *Modern Heuristics Techniques for Combinatorial Problems*. John Wiley & Sons, Great Britain, 1993.

[35] J. David Schaffer. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum, 1985.

[36] Ellen M. Sentovich, Kanwar Jit Singh, Luciano Lavagno, and et al. Cho Moon. SIS: A system for sequential circuit synthesis. Technical report, University of California at Berkeley, 1992.

[37] Claude E. Shannon. A symbolic analysis of relay and switching circuits. *Transactions American Institute of Electrical Engineers*, 57:713–123, March 1938.

[38] Yuhui Shi and Russell C. Eberhart. Parameter selection in particle swarm optimization. In Waagen D. y Eiben A.E. Porto V.W., SaravananÑ., editor, *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 591–600. Springer, 1998.

[39] Peter J. Bentley Timothy G. W. Gordon. On evolvable hardware. In Seppo J. Ovaska and Les m Sztandera, editors, *Soft Computing in Industrial Electronics*, pages 279–323, Heidelberg, 2002. Eds,.Physica-Verlag.

[40] Ronald. J. Tocci. *Sistemas digitales: principios y aplicaciones*. Prentice Hall, 1993.

[41] Frans van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, Faculty of Natural and Agricultural Science, University of Pretoria, November 2001.

[42] Xin Yao and Tetsuya Higuchi. Promises and Challenges of Evolvable Hardware. In Tetsuya Higuchi, Masaya Iwata, and W. Liu, editors, *Proceedings of the First International Conference on Evolvable Systems: From Biology to Hardware (ICES'96), Lecture Notes in Computer Science, Vol. 1259*, pages 55–78, Heidelberg, Germany, 1997. Springer-Verlag.

[43] Andrej Zemva, Franc Brglez, and Baldomir Zajc. Multi-level logic optimization based on wave synthesis of permissible mutation functions. In *In International Workshop on Logic Synthesis (IWLS'98)*, June 1998.

[44] Demetrius Zissos. *Logic design algorithms.* Oxford University Press, London, 1972.