CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

Unidad Zacatenco

Departamento de Ingeniería Eléctrica

Área de Computación

Localización de Funciones Booleanas Balanceadas con Máxima No Linealidad

Tesis que presenta

Daniel López Fernández

para obtener el grado de

Maestro en Ciencias

en la especialidad de

Ingeniería Eléctrica

Director de la Tesis

Dr. Guillermo Morales Luna

México, D.F. **Agosto 2006**

Dedicatoria

A Cinthya Rivera mi querida esposa y musa inspiradora,
quien vivió a mi lado la aventura de hacer la maestría,
tu apoyo y alegría por estar a mi lado
fue la luz que iluminaron mis noches .

Agradecimientos

Agradezco al Consejo Nacional de Ciencia y Tecnología *CONACyT* por el respaldo financiero otorgado por medio de la beca. que me brindo para la realización de esta tesis.

Agradezco a los profesores de la sección de computación del Centro de Investigación y de Estudios Avanzados del IPN por el conocimiento, tiempo y esfuerzo que me brindaron durante la maestría.

Agradezco al Dr. Guillermo Morales Luna la oportunidad de desarrollo este trabajo de Tesis y por su apoyo incondicional para extender mi visión y conocimiento sobre las ciencias de la computación.

Agradezco al Dr. Carlos Artemio Coello Coello y Debrup Chakraborty el haber revisado este trabajo y por los valiosos comentarios que permitieron mejorar este documento.

Agradezco a mis amigos por brindarme su amistad y apoyo durante los momentos más difíciles en la maestría dentro y fuera del salón de clases, en especial a Hilda Chable, Hector Acosta, Oscar Alvarado y Renato Zacapala.

A Sofía Reza y Flor Córdova por las atenciones que tuvieron conmigo y que sobrepasan el trato laboral.

Resumen

En el ámbito de la criptografía, las funciones booleanas juegan un papel importante ya que éstas son el elemento principal en el desarrollo de algunos cripto-sistemas como: los cifradores por bloques, funciones hash y principalmente cajas de sustitución. Para que éstas puedan ser utilizadas, deben satisfacer uno o más criterios, como el del balance para evitar que los cripto-sistemas proporcionen información del texto en claro cuando el atacante posee el texto cifrado y el criterio de no linealidad, con el que los sistemas criptográficos se hacen resistentes al criptoanálisis lineal desarrollado por Matsuy. El optimizar la propiedad de no linealidad de las funciones booleanas manteniendo el criterio de balance es un problema difícil de tratar ya que el número de funciones booleanas crece de manera doblemente exponencial.

En este trabajo presentamos el desarrollo de un método para localizar funciones booleanas balanceadas con máxima no linealidad utilizando los algoritmos de búsqueda local y recocido simulando, y un método para seleccionar puntos sobre el espacio de búsqueda, el cual utiliza una estructura de gráfica para representar el espacio de las funciones booleanas balanceadas, una representación de las funciones booleanas basadas en patrones y la teoría de cúmulos.

Nuestro propósito ha sido localizar patrones que guíen búsquedas para la maximización de la no-linealidad sujetos a la restricción de balance y hemos optado por considerar algoritmos deterministas. El uso de algoritmos de tipo heurístico o evolutivo queda fuera de los alcances de esta tesis.

Dentro del desarrollo de la metodología se presenta una biblioteca de funciones para medir y optimizar la no linealidad de las funciones booleanas, para que nuevas heurísticas sean desarrolladas a partir de los resultados obtenidos.

Abstract

In Cryptography, Boolean functions are extremely important, since they contribute quite relevantly in some cryptsosystems as stream block ciphers, hash maps and substitution boxes. The used Boolean functions should fulfill supplementary criteria, such as balancedness, in order to avoid that the crypstosystem can provide clear text information when an attacker has a corresponding cipher text, and non-linearity, in order to provide resistance against linear cryptoanalysis e. g. that due to Matsuy. Optimization of non-linearity subject to balancedness is a hard problem to deal with, since the number of Boolean maps grows doubly exponentially with respect to the involved number of variables.

Here we introduce a method to find balanced Boolean maps tending to maximize non-linearity with local search algorithms and variants, e.g. simulated annealing, and a method for point selection over the search space through a graph structure to represent the collection of balanced Boolean maps, as well as a pattern representation of maps and current techniques of Cluster Analysis to classify them.

Our purpose has been to recognize patterns leading the search when maximizing non-linearity subject to balancedness and we have restricted ourselves to deterministic procedures. The use of heuristic or onary algorithms is beyond the scope of this thesis.

Within the methodology we have built a program library to measure and optimize non-linearity of Boolean maps. This library is the experimental platform in this work, and we look toward an extensive use of it in other researches.

Índice general

Dedicatoria III

Agradecimientos V

Resumen VII

Abstract IX

Lista de Figuras XIII

Lista de Tablas XV

1. Introducción **1**
1.1. Objetivo 3
1.2. Descripción de la Tesis 4

2. Funciones Booleanas **5**
2.1. Criterio de Balance 7
2.2. Función de No Linealidad 8

3. Teoría de Gráficas **17**
3.1. Gráfica de Funciones Booleanas Balanceadas 20

4. Algoritmos de Búsqueda **25**
4.1. Recocido Simulado 29
4.2. Cálculo de Parámetros 32
4.3. Optimización de la No Linealidad de las Funciones Balanceadas . 40

5. Heurística **49**
5.1. Teoría de Cúmulos 55
5.2. Selección de Puntos 60
5.3. Resultados 62

6. Conclusiones **67**
6.1. Trabajo a Futuro 69

Lista de Figuras

2.1. Diagrama de Transformación 11
2.2. Cálculo de la Transformada de Walsh 14
2.3. Tiempo del cálculo de la no-linealidad 15

3.1. Puentes de Königsberg 17
3.2. Gráfica dirigida y no dirigida 18
3.3. Gráficas completas no dirigidas de grado $0 \leq n \leq 6$ 20
3.4. La gráfica $\mathcal{G}(2)$. 21
3.5. La gráfica $\mathcal{G}(2)$. 23
3.6. La gráfica $\overline{\mathcal{G}}(2)$. 24

4.1. Gráfica de error del algoritmo de búsqueda local 33
4.2. Gráfica de error del algoritmo de búsqueda local para $5 \leq n \leq 12$ 34
4.3. Enfriamiento del sistema 35
4.4. Enfriamiento del sistema 36
4.5. Criterio de paro uno 37
4.6. Criterio de paro uno 37
4.7. Criterio de paro dos 38
4.8. Criterio de paro dos 38
4.9. Gráfica de error del algoritmo de recocido simulado 39
4.10. Gráfica de error del algoritmo de recocido simulado 40
4.11. Generación de funciones balanceadas 42
4.12. No-linealidad promedio en nodos de $\mathcal{G}(n)$ 47

5.1. Representación en patrones de la función $f : 10101010$. 50
5.2. Representación en patrones de $\mathbb{B}_2^3$ y $\mathbb{B}_2^4$. 51
5.3. Representación en patrones de $\mathbb{B}_2^3$ con máxima nl. 52
5.4. Representación en patrones de vecinos intermedios. 55

5.5. Algoritmo de k-means 57
5.6. Representación en patrones de vecinos intermedios. 59
5.7. Patrones de $0 \times 845BCBAC$ $0 \times 2a79DAE$. 59
5.8. Extrapolación de patrones de 7 a 8 variable. 60
5.9. Selección de vecinos 61
5.10. Selección de vecinos 65

Lista de Tablas

2.1. Crecimiento de $\mathbb{F}_2^n$ 6
2.2. Tabla de verdad 6
2.3. Tiempo del cálculo de la no-linealidad 15

3.1. Crecimiento de $\mathcal{G}(n)$ 22

4.1. No linealidad promedio 26
4.2. Método algebraico 27
4.3. Tabla de verdad 41
4.4. Resultados preliminares 43
4.5. Máxima no Linealidad de funciones con n variable ($2 \leq n \leq 5$) . 43
4.6. Resultados de la búsqueda local 45
4.7. Resultados del recocido simulado 46
4.8. No-linealidad promedio en $\mathcal{G}(5)$ y $\mathcal{G}(6)$ 46

5.1. Resultados preliminares 62
5.2. Número de variables 6; Número de vecinos: 223; Número de ciclos: 4 63
5.3. Número de variables : 6; Número de vecinos: 213; Número de ciclos: 3 63
5.4. Número de variables : 6; Número de vecinos: 213; Número de ciclos: 1 64
5.5. Número de variables : 7; Número de vecinos: 231; Número de ciclos : 5 64
5.6. Parámetros iniciales 65
5.7. Nuevos parámetros 65

Capítulo 1

Introducción

La criptografía se define como la ciencia de cifrar y descifrar información utilizando técnicas matemáticas que hagan posible el intercambio de mensajes de manera que sólo puedan ser leídos por las personas a quienes van dirigidos [1].

Actualmente las aplicaciones criptográficas son utilizadas en la seguridad de las comunicaciones, procesos de identificación y autenticación, certificación, comercio electrónico, etc.

Un algoritmo criptográfico es un método matemático que se emplea para cifrar y descifrar un mensaje. Generalmente éstos funcionan empleando una o más claves (números o cadenas de caracteres) como parámetros, de tal forma que estos sean necesarios para recuperar el mensaje a partir del texto cifrado; un criptosistema, es un sistema para cifrar y descifrar información compuesto por un conjunto de algoritmos criptográficos, claves y posiblemente, varios textos en claro con sus correspondientes versiones en texto cifrado.

Un criptoanálisis consta de un conjunto de procedimientos, procesos y métodos empleados para romper un algoritmo criptográfico, descifrar un texto cifrado o descubrir las claves empleadas para generarlo.

Dentro del ámbito de la criptografía las funciones booleanas juegan un papel importante, pues éstas son un bloque fundamental en la construcción de criptosistemas tales como *cifradores por bloques*, funciones *hash*, *cajas de sustitución*, etc. Pero para que las funciones booleanas puedan ser utilizadas en el desarrollo de cripto-sistemas resistentes a los diferentes cripto-análisis desarrollados, éstas deben cumplir con ciertos criterios, tales como el de *balance* (es decir la función

toma la misma cantidad de números $1's$ y $0's$) con el que se evita que el sistema proporcione información estadística del texto en claro cuando se conoce el texto cifrado, y la *no-linealidad* la que permite desarrollar cripto-sistemas resistentes al cripto-análisis lineal desarrollado por Matsui en 1993.

Funciones Booleanas

Sea $\mathbb{F}_2 = \{0, 1\}$ el campo primo de dos elementos y $\mathbb{F}_2^n$ su n-ésima potencia cartesiana, se cumple que para toda $n \in \mathbb{N}$ que la cardinalidad de $\mathbb{F}_2^n$ es igual a 2^n.

Una función booleana es una asociación $\mathbb{F}_2^n \rightarrow \mathbb{F}_2$. Para cualquier vector en $\mathbb{F}_2^n$ una función booleana toma valores 0 o 1 en consecuencia la cardinalidad del espacio de funciones booleanas con n variables es igual a 2^{2^n}.

Dado que el espacio de las funciones crece exponencialmente con respecto al número de variables n, no es posible realizar búsquedas exhaustivas para valores de n relativamente pequeños, que garanticen la localización de funciones booleanas con propiedades criptográficas importantes. Esto ha forzado a desarrollar métodos de búsqueda que optimicen la no linealidad sin que éstos exploren todo el espacio de búsqueda.

Métodos de Búsqueda

A lo largo de la historia, el problema de localizar funciones con máxima no linealidad ha sido tratado con diferentes estrategias y enfoques, los cuales pueden ser clasificados en tres tipos: métodos aleatorios, algebraicos y heurísticos.

Los métodos aleatorios son lo más simples ya que evitan especificar propiedades combinatorias, aunque no son capaces de localizar valores óptimos. Por otra parte, los métodos algebraicos pueden alcanzar valores altos de no-linealidad conservando el criterio de balance, pero éstos tienden a arrojar propiedades cualitativas pobres. Finalmente, los métodos heurísticos, muestran ser capaces de generar fácilmente funciones booleanas balanceadas con alta no linealidad. Estas técnicas ofrecieron una seria alternativa a la construcción algebraica. Posteriormente surgieron métodos híbridos (combinación de heurísticas evolutivas y construcciones algebraicas) que fueron más efectivas que los métodos simples [2].

En este trabajo se optó por utilizar un método determinista (recocido simulado) para localizar funciones booleanas balanceadas con máxima no-linealidad,

debido a que el mecanismo de búsqueda que utiliza permite localizar patrones en el comportamiento de la no-linealidad y es mediante estos patrones que es posible guiar la búsqueda hacia óptimos globales con un menor esfuerzo.

1.1. Objetivo

El objetivo principal de este trabajo es localizar funciones booleanas con máxima no linealidad, debido a su importancia en el desarrollo de cripto-sistemas resistentes a los diferentes criptoanálisis que han sido desarrollados.

Debido a que el espacio de funciones booleanas crece de forma doblemente exponencial con respecto al número de variables, una búsqueda exhaustiva que garantice localizar óptimos globales resulta imposible. Por otra parte la probabilidad de encontrar funciones booleanas con propiedades criptográficas como la alta no-linealidad disminuye cuando el número de variables n aumenta. Por esta razón, un método de búsqueda eficiente que sea capaz de localizar funciones con propiedades criptográficas importante, sin que éste explore todo el espacio de búsqueda se vuelve necesario.

En el presente trabajo desarrollamos un método de búsqueda que tiene como característica el ser un método **simple** capaz de localizar funciones booleanas balanceadas con máxima no linealidad (básicamente, variantes del *Recocido Simulado*).

Además del método de búsqueda propondremos un nuevo enfoque en la búsqueda de funciones booleanas al utilizar una estructura de gráfica para representar el espacio de las funciones booleanas balanceadas, con la que se tiene una descripción más exacta del espacio de búsqueda, una regla de selección que disminuye el tiempo de cómputo para que los algoritmos converjan hacia los valores óptimos (parciales) reduciendo el número de iteraciones del algoritmo de recocido simulado así como el número de funciones que éste necesita consultar en cada iteración.

Presentamos una biblioteca de funciones en lenguaje 'C' utilizando el conjunto de programas de Aritmética de Precisión Múltiple de GNU (*Multiple Precision Aritmethic*) [3] bajo la plataforma LINUX para implementar la heurística y contar con una plataforma que permita desarrollar programas y pruebas de forma rápida.

1.2. Descripción de la Tesis

El presente documento se divide en 5 partes en las que se describe el trabajo desarrollado para localizar funciones booleanas balanceadas con alta no-linealidad utilizando los algoritmos de recocido simulado y búsqueda local, una representación del espacio de funciones booleanas balanceadas más precisa a través de la gráfica de funciones booleanas, con la que se propuso un método para determinar el comportamiento de la no-linealidad aplicando la teoría de cúmulos.

En el capítulo 2 definimos las funciones booleanas, sus características y sus representaciones en tablas de verdad, sus representaciones en forma normal algebraica y una más basada en patrones. Por otra parte, definimos el criterio de balance, la noción de no linealidad, sus propiedades, los diferentes métodos algebraicos propuestos para evaluarla y el método que es implementado para calcularla. Por último, definimos una estructura de gráfica para representar el espacio de funciones booleanas balanceadas y algunas propiedades generales de ésta.

En el capítulo 3 describimos los métodos de búsqueda utilizados para localizar funciones booleanas con propiedades criptográficas importantes, enfocándonos en los algoritmos de búsqueda local y recocido simulado, así como la metodología que se siguió para establecer los parámetros de los algoritmos de búsqueda en base al número de variables de la función booleana y mostraremos los resultados preliminares obtenidos al aplicar los algoritmos de búsqueda utilizando una selección aleatoria de funciones booleanas balanceadas, una selección sobre puntos en la gráfica y un análisis de cómo la cardinalidad del espacio de búsqueda y el comportamiento de la no-linealidad afectan el desempeño de éstos.

En el capítulo 4 mostramos el desarrollo de una heurística para seleccionar puntos en el espacio de búsqueda utilizando la gráfica de funciones balanceadas definida en el capítulo 2 y la teoría de cúmulos. Con ella se establecerán los bits que deben cambiar en una función booleana con no linealidad actual para incrementar la no linealidad.

Finalmente en el capítulo 5 mencionamos las conclusiones que se obtuvieron al finalizar el desarrollo del trabajo y cúal es el trabajo a futuro.

Capítulo 2

Funciones Booleanas

Siguiendo la convención mencionada, cuando hablemos de funciones nos referiremos a funciones booleanas. Estas juegan un papel importante en el desarrollo de criptosistemas. Son utilizadas como funciones *hash* (MD5, SHA, $\cdots$), cifradores de flujo, cifradores simétricos (DES, AES, $\cdots$) y de manera especial como cajas de sustitución.

Para que las funciones puedan ser utilizadas en el desarrollo de criptosistemas, éstas deben tener ciertas características, para evitar que los sistemas proporcionen información del texto en claro cuando se posea el texto cifrado. Por otra parte,las funciones se utilizan para que los criptosistemas sean resistentes a los criptoanálisis desarrollados, tales como el de tipo lineal o el de tipo diferencial.

Sea $\mathbb{F}_2 = \{0, 1\}$ el campo primo de dos elementos, y sea $\mathbb{F}_2^n$ su n-ésima potencia cartesiana. Se cumple que $\forall n \in \mathbb{N}$ la cardinalidad de $\mathbb{F}_2^n$ es 2^n. Una función es de la forma $f : \mathbb{F}_2^n \to \mathbb{F}_2$. En otras palabras una función es una asociación de vectores en $\mathbb{F}_2^n$ con puntos en $\mathbb{F}_2$. Denotaremos por $\mathbb{F}_2^{\mathbb{F}_2^n}$ al espacio de funciones con n variables [4].

Para los 2^n vectores en $\mathbb{F}_2^n$, una función $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$ puede tomar cualesquiera valores 0 o 1. Por consiguiente, existen 2^{2^n} funciones. Esta cardinalidad constituye uno de los principales problemas en la búsqueda de funciones con propiedades criptográficas importantes. La tabla 2.1 ilustra el crecimiento del espacio de funciones con n variables.

Una función puede ser expresada de diferentes formas, una de las más simples es mediante las *tablas de verdad*, la cual proporciona una representación explícita de la función. La tabla de verdad de $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$ consta de 2^n valores correspondientes a f sobre todas las posibles entradas en $\mathbb{F}_2^n$. Por ejemplo, pa-

n	$card\{\mathbb{F}_2\}$	$card\{\mathbb{F}_2^n\}$
3	8	256
4	16	65536
5	32	4294967296
6	64	18446744073709551616
7	128	3.402823669209384e+38

Tabla 2.1: Crecimiento de $\mathbb{F}_2^n$

x_1	x_2	x_3	$f(x)$
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	1
1	1	1	1

Tabla 2.2: Tabla de verdad

ra $f(x_1, x_2, x_3) = x_1x_2 + x_3$, su tabla de verdad es la mostrada en la tabla 2.2. La principal desventaja de utilizar tablas de verdad para expresar funciones, es que éstas crecen exponencialmente, por lo que para valores relativamente grandes de n las tablas de verdad resultan poco prácticas.

Otra forma de representar una función es mediante su *Forma Normal Algebraica* (FNA). Una función puede ser expresada mediante una suma (XOR) de productos (AND), como se muestra a continuación en la ecuación 2.1. Toda f posee una única forma reducida FNA.

$$f = \bigoplus_{a_1,\cdots,a_n \in \mathbb{F}_2^n} h(a_1, \cdots, a_n) x_1^{a_1}, \cdots, a_n^{a_n} \tag{2.1}$$

Dada la FNA de la función f, su *grado algebraico*, denotado por $gr(f)$ se define como la mayor de las sumas $\sum_{i=0}^{n} a_i$. De esta forma, por ejemplo, la función $x_1 \oplus x_2$ es de grado 1, $x_1 \oplus x_2 \oplus x_3$ es de grado 3, etc.

Una función $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$ se dice ser una función lineal, es decir, tal que para cualesquiera $x, y \in \mathbb{F}_2^n$ y $c \in \mathbb{F}_2$, $f(x+y) = f(x) + f(y)$ y $f(cx) = cf(x)$. Una función afín es una función lineal trasladada por 1 (esto en el álgebra booleana significa tomar complementos). El conjunto de funciones lineales es un espacio vectorial n-dimensional sobre $\mathbb{F}_2$.

Equivalentemente podemos decir que una función de grado a lo más 1 es una función *afín*. Una función afín con un término constante igual a 0 es lineal. Denotaremos por A_n al conjunto de todas las funciones afines de n variables y la cardinalidad del conjunto de funciones afines en $\mathbb{F}_2^n$ es 2^{n+1}[5].

Dos métricas importantes de las funciones son el *peso de Hamming* y la *distancia de Hamming* a las que denotaremos como p_H y d_H. Para $f_1 \in \mathbb{F}_2^n$, el peso de Hamming de f_1 se define como

$$p_H(f_1) = card\{x \in \mathbb{F}_2^n \,|\, f(x) = 1\}$$

y para $f_1, f_2 \in \mathbb{F}_2^n$, la distancia de Hamming de f_1 a f_2 se define como

$$d_H(f_1, f_2) = card\{x | f_1(x) \neq f_2(x)\}.$$

La distancia de Hamming puede ser calculada mediante la relación

$$dH(f_1, f_2) = pH(f_1 \oplus f_2).$$

Un operador utilizado en el cálculo de la no-linealidad es el *producto de Kronecker* de matrices (véase la ecuación 2.4), el cual para una matriz A de orden $m \times n$ y una matriz B de orden $s \times t$, es $A \otimes B$ definida según 2.2:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{bmatrix} \tag{2.2}$$

donde a_{ij} es el elemento en la i-ésima fila y j-ésima columna de la matriz A [6].

2.1. Criterio de Balance

Las aplicaciones criptográficas, tales como el diseño de cajas de sustitución fuertes, a menudo requieren que cuando la entrada de una función es seleccionada de forma independiente o aleatoria, la salida de la función debe tener una

distribución normal, en otras palabras, la función debe ser balanceada. Mediante el criterio de balance se evita que el cripto-sistema proporcione información estadística del texto en claro cuando se posee el texto cifrado [7].

Para definir este criterio, primero definiremos las nociones de *soporte* y *parte nula* de una función.

Para $f \in \mathbb{F}_2^n$ denotaremos su soporte como $Spt(f)$ y se define [4] como

$$Spt(f) = f^{-1}[1] = \{\delta \in \mathbb{F}_2^n \,|\, f(\delta) = 1\}$$

y la parte nula de la función f a la que denotaremos como $Nul(f)$ se define [4] como

$$Nul(f) = f^{-1}[1] = \{\delta \in \mathbb{F}_2^n \,|\, f(\delta) = 0\}.$$

En otras palabras, el soporte de una función es el número de $1's$ que aparecen en su tabla de verdad y la parte nula el número de $0's$.

A partir de la definición de soporte y parte nula de una función, se dice que $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$ es balanceada si $card(Spt(f)) = card(Nul(f)$. Dicho de otra forma, f es balanceada si $card(Spt(f)) = 2^{n-1}$ y $card(Nul(f)) = 2^{n-1}$.

Si denotamos con $\mathbb{B}_2^n$ al espacio de funciones balanceadas de n variables, se cumple que

$$card(\mathbb{B}_2^n) = \binom{2^{n-1}}{2^{n-2}}^2.$$

Por otra parte, definiremos el complemento de una función f como $\overline{f} = f \oplus 1$ donde 1 denota la función cuya tabla de verdad es 1 para las 2^n entradas. Es fácil ver que $Spt(\overline{f}) = Null(f)$ y viceversa $Null(\overline{f}) = Spt(f)$, lo que implica que $d_H(f, \overline{g}) = 2^n - d_H(f, g)$ para cualesquiera $f, g \in \mathbb{F}_2^{\mathbb{F}_2^n}$.

Puede verse fácilmente que si $f \in \mathbb{B}_2^n$ entonces se cumple que $\overline{f} \in \mathbb{B}_2^n$.

2.2. Función de No Linealidad

Para los sistemas criptográficos los métodos de *confusión* y *difusión* introducidos por Shannon son fundamentales para obtener seguridad. La, así llamada, confusión es un reflejo de la no linealidad de ciertas funciones. La no linealidad es un elemento crucial dado que la mayoría de los criptosistemas lineales son fácilmente rompibles [8].

La no linealidad de las funciones puede ser considerada como una forma de medir la fortaleza de un cripto-sistema, ya que la habilidad para violentar un sistema criptográfico es proporcional a que éste pueda ser representado como un conjunto de ecuaciones lineales [6].

Supongamos que se tiene un cifrador por flujo con dos registros de 64 bits y una función de combinación f. Un ataque por fuerza bruta involucraría probar 2^{128} posibles llaves, si por ejemplo, $f(x_1, x_2) = x_1 \oplus x_2$, conociendo la salida de la función f entonces sólo existen dos posibles combinaciones de x_1 y x_2 para generar la salida. Si se observan las primeras 64 salidas producidas por f, es posible reducir el tamaño del espacio de búsqueda de la llave a 2^{64}, lo que nos muestra lo riesgoso que puede ser utilizar funciones lineales. Actualmente se han desarrollado métodos más sofisticados que explotan la linealidad de las funciones utilizadas en los criptosistemas [5].

La no linealidad de una función $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$ a la que denotaremos como nl, se define como la mínima distancia de Hamming de f al conjunto de las funciones afines de n variables : $nl(f) = min_{g \in A_n}(d(f, g))$. Sin embargo, el calcular la no linealidad de una función usando esta ecuación no es práctico.

Una herramienta importante para el análisis de funciones es la llamada *transformada de Walsh*, algunas veces llamada también *distribución espectral* o simplemente el *espectro de la función*. A través de la transformada de Walsh es posible calcular la no-linealidad de una función.

Para $\overline{X} = (X_n, \cdots, X_1)$ y $\overline{w} = (w_n, \cdots, w_1)$ en $\mathbb{F}_2^n$, sea $\overline{X} \cdot \overline{w} = X_n w_n \oplus \cdots \oplus X_1 w_1$ el producto interno para $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$, *la transformada de Walsh* de f en $\overline{X} \in \mathbb{F}_2^n$ se define como 2.3, donde la suma es sobre todos los $\overline{w} \in \mathbb{F}_2^n$ [9].

$$W_f(\overline{X}) = \sum_{\overline{w}} (-1)^{f(\overline{X}) \oplus \overline{X} \cdot \overline{w}} \tag{2.3}$$

La distancia Hamming de una función f y una función afín $g(X) = X \cdot w + b$ con $b \in \mathbb{F}_2$, puede ser calculada usando la transformada de Walsh mediante

$$H_n(f, g) = 2^{n-1} - \frac{(-1)^b W_f(w)}{2}$$

así la no linealidad de la función f puede ser obtenida a partir de la transformada de Walsh como

$$nl(f) = \frac{1}{2}(2^n - max_w(W_f(w))).$$

Se han desarrollado diferentes métodos para calcular la no-linealidad de una función. En primer lugar se encuentran los métodos espectrales, los cuales han sido utilizados en el diseño lógico, de prueba y de clasificación de funciones. Sin embargo, los métodos espectrales no parecían ser prácticos hasta que recientemente fue posible reducir el costo de calcular el espectro [10].

Un método espectral importante para calcular la no linealidad es el basado en la *gráfica de Cayley*. Ésta es una técnica de interés teórico y muestra la equivalencia del espectro de la gráfica de Cayley y el espectro de Walsh con el que es posible calcular la no linealidad de una función [10].

La gráfica de Cayley es una estructura utilizada para relacionar grupos algebraicos con la Teoría de Gráficas. Esta técnica para el cálculo del espectro de Walsh de una función f parte de la representación de una función con base en la definición de un *grupo* específico.

Un grupo $(M, *)$ consiste de un conjunto de elementos M y de un operador binario $*$ sobre M, tal que $*$ es cerrado, asociativo, posee un elemento identidad y cada elemento posee un inverso [10].

El grupo de Cayley $(M, \oplus)$ es utilizado para caracterizar las funciones $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$, donde M consiste de todo los minitérminos en $\mathbb{F}_2^n$, esto es, todos los puntos en el espacio $\mathbb{F}_2^n$ y $\oplus$ es el operador binario del grupo. Este grupo tiene un elemento identidad que corresponde a la función que tiene 0's en las 2^n entradas de su tabla de verdad y adicionalmente para cada $m_i \in M$, $m_i^{-1} = m_i$ [10].

La gráfica de Cayley correspondiente al grupo que representa a la función f tiene como vértices al conjunto V, tal que para cada $v_i \in V$ le corresponde un elemento $m_i \in M$ y el conjunto de aristas E está definido por

$$E = \{(m_i, m_j) \in B^n \times B^n | f(m_i \oplus m_j = 1)\}.$$

La matriz de adyacencia A de la gráfica de Cayley es una matriz cuadrada de dimensión $2^n \times 2^n$ con $a_{ij} = 1$ si $f(m_i \oplus m_j) = 1$ y $a_{ij} = 0$ en otro caso, dado que $m_i \oplus m_j = n_j \oplus m_i$ la matriz de adyacencia A es simétrica.

El espectro de la gráfica se define como el conjunto de los eigen-valores de la matriz de adyacencia de la función f y éste es idéntico al espectro de Walsh.

Para calcular la matriz de adyacencia se propuso un método similar a los diagramas de mariposa de transformación rápida, haciendo posible calcular las $n_1, \ldots, n_n$ filas de la matriz de adyacencia a partir de la fila n_0 por una serie de transposiciones como se muestra en la figura 2.1

Pero en la práctica, los métodos basados no son del todo atractivos; ya que éstos involucran una conversión del dominio booleano al dominio espectral. De-

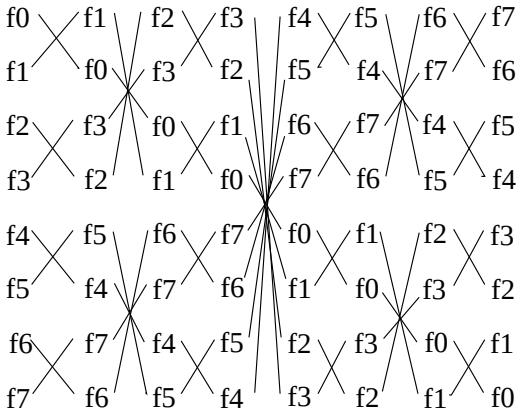


Figura 2.1: Diagrama de Transformación

safortunadamente, estos métodos se basan en el uso de matrices, y resultan ineficientes para funciones grandes. Porwirk propuso un método alternativo al método espectral, el cual se basa en los coeficientes de la transformada de Walsh [7].

Una función $f(x_1, x_2, \cdots, x_n)$ puede ser transformada del dominio $\mathbb{F}_2$ al dominio espectral mediante la transformación lineal $H \cdot Y = R$, donde H es una matriz de transformación ortogonal de $2^n \times 2^n$, $y = [y_0, y_1, \cdots, y_n]$ es el vector de verdad de la función $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$ y $R = [r_0, r_1, \cdots, r_n]$ es el vector de coeficientes espectrales [7].

Una matriz importante en el cálculo del espectro de una función, es la matriz de Sylvester-Hadamard, la cual se define por la regla recursiva 2.4, donde $\bigotimes$ denota el producto de Kronecker.

$$H_0 = [1], \quad H_n = \begin{bmatrix} 1 & 1 \\ 1 & \text{-}1 \end{bmatrix} \bigotimes H_{n-1}, \quad n = 1, 2, \ldots \tag{2.4}$$

Otra forma de generar la matriz 2.4 es con la fórmula 2.5.

$$H_0 = [1], \quad H_n = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & \text{-}H_{n-1} \end{bmatrix} \bigotimes H_{n-1}, \quad n = 1, 2, \ldots \tag{2.5}$$

Por ejemplo

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & \text{-1} \end{bmatrix}; \quad H_4 = H_2 \bigotimes H_2 = \begin{bmatrix} H_2 & H_2 \\ H_2 & \text{-}H_2 \end{bmatrix}$$

$$H_4 = \begin{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & \text{-1} \end{bmatrix} & \begin{bmatrix} 1 & 1 \\ 1 & \text{-1} \end{bmatrix} \\ \begin{bmatrix} 1 & 1 \\ 1 & \text{-1} \end{bmatrix} & \begin{bmatrix} \text{-1} & \text{-1} \\ \text{-1} & 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \text{-1} & 1 & \text{-1} \\ 1 & 1 & \text{-1} & \text{-1} \\ 1 & \text{-1} & \text{-1} & 1 \end{bmatrix}$$

Adicionalmente tenemos que $H_n = H_n^T$ y $H_n \cdot H_n^T = 2^n \cdot I_n$, donde I_n es la matriz identidad de orden 2^n. Dado que $H_n^-1 = \frac{1}{2} H_n^T$, la matriz H_n es ortogonal [7].

El coeficiente espectral calculado de acuerdo a 2.4 es llamado *coeficiente de Walsh*. Esta transformación es conocida como la transformada de Walsh-Hadamard (WHT) [7].

Cualquier función $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$ puede ser expresada a través de los coeficientes de Walsh-Hadamard como un polinomio

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) = \; & \tfrac{1}{2^n}[r_0 + r_1 \cdot (-1)^{x_n} + r_2 \cdot (-1)^{x_{n-1}} + r_3 \cdot (-1)^{x_n \oplus x_{n-1}} \\ & + \ldots + r_{2^{n-1}} \cdot (-1)^{x_n \oplus x_{n-1} \ldots \oplus x_1}] \end{aligned}$$

donde $\oplus$ es la adición módulo 2, y $r_0, r_1, \ldots, r_{2^n-1} \in \mathbb{R}$ son los coeficientes espectrales [7].

De la definición de la función de Walsh, para cualquier función afín f_k tenemos lo siguiente: para $c = 0$ se cumple 2.6 y para c=1 se cumple 2.7.

$$y_k = f_k(x) = \frac{1}{2}(1 - wal(k, t)) \tag{2.6}$$

$$y_k = f_k(x) = \frac{1}{2}(1 - ((-1) \cdot wal(k, t))) \tag{2.7}$$

De lo anterior se tiene que cualquier función puede ser generada a partir de la matriz de Hadamard.

$$\begin{aligned} &\text{Para } c = 0 \text{ dé } h_n \\ &\text{Para } c = 1 \text{ dé } h_n = -1 \cdot H_n \end{aligned} \tag{2.8}$$

Del significado de la transformada de Walsh, se pueden encontrar únicamente 2^n funciones lineales. Por otra parte, cualquier función f puede ser caracterizada por el espectro de la transformada de Walsh, donde $k = c + \sum_{i=1}^{n} = a_i 2^i, a_j, c \in \mathbb{F}_2$ tienen el mismo significado y $x = 0, 1, 2, \ldots, 2^n - 1$.

$$r_x = \begin{cases} +2^{n-1} & \text{para } x = 0 \\ -2^{n-1} & \text{para } x = \frac{k}{2} \quad \Leftrightarrow c = 0 \\ +2^{n-1} & \text{para } x = \frac{k-1}{2} \quad \Leftrightarrow c = 1 \\ 0 & \text{en otro caso} \end{cases} \tag{2.9}$$

Una función f de n variables es afín si y sólo si $r_0 = 2^{n-1}$ y el valor del coeficiente espectral de n-ésimo orden es $\pm 2^{n-1}$ [7].

Como ya se ha mencionado, la no linealidad se define como

$$nl(f) = min(d_H(f, \varphi_i)), i = 1, 2, \ldots, 2^{n+1}$$

que es igual a

$$min\{w(f \oplus \varphi_i), i = 1, 2, \ldots, 2^{n+1}\},$$

donde $\phi = \{\varphi_1, \varphi_2, \ldots \varphi_{2^{2n+1}}\}$ es el conjunto de funciones afines.

Sin embargo, el calcular la no linealidad de esta forma, resulta inconveniente ya que es necesario realizar 2^{n+1} operaciones de comparación. En cambio con el método espectral, la no linealidad puede ser calculada en $n \cdot 2^n$ [7].

La matriz de Sylvester-Hadamard está estrechamente relacionada con las funciones lineales. Sea $H_n = \begin{bmatrix} l_0 \\ l_1 \\ \vdots \\ l_2^n - 1 \end{bmatrix}$ donde l_i es una fila de H_n, entonces l_i es la secuencia de $h_i =< \alpha_1, \alpha >$, una función lineal, donde α_i es un vector $\mathbb{F}_2$ cuya representación entera es i y $x = (x_1, \ldots, x_n)$. Consecuentemente la secuencia de cualquier función lineal en $\mathbb{F}_2^{\mathbb{F}_2^n}$ es una fila de H_n [6].

En la práctica se utilizó el siguiente método para calcular la transformada de Walsh. En primer lugar sabemos que la longitud de una función f es una potencia de 2 y que cada pareja de valores de la función produce dos resultados, los cuales reemplazan a los valores de la pareja original en la misma posición. La parte izquierda del resultado será la suma de los elementos mientras que el resultado de la parte derecha será igual al primer valor menos el segundo, es decir, para la entrada (a, b) el resultado es

$$(a', b') = (a + b, a - b)$$

así para los valores $(1, 0)$ el resultado es $(1 + 0, 1 - 0) = (1, 1)$, el procedimiento inicia calculando las parejas de elementos adyacentes. En el siguiente paso se toma el $2do$ elemento, en el siguiente se toma el $4to$ elemento y así sucesivamente

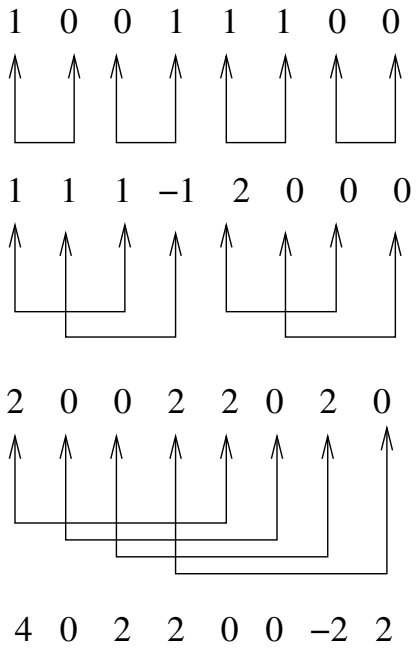


Figura 2.2: Cálculo de la Transformada de Walsh

hasta que la siguiente pareja ya no pueda ser calculada. Por ejemplo, sea f una función cuyo vector de verdad es 10011100 entonces la transformada de Walsh de f se muestra en la figura 2.2 y por lo tanto, la no linealidad de la función de 3 variables f es:

$nl(f) = nl(f) = \frac{1}{2}(2^n - max_w(W_f(w))) = \frac{1}{2}(2^3 - 4) = \frac{1}{2}(4) = 2.$

Este método para calcular la no linealidad es simple, pero el número de operaciones crece proporcionalmente al número de variables. Calculando el tiempo que toma calcular la no-linealidad utilizando una computadora con procesador Intel Centrino a 1 Ghz, 768 MB de RAM, se obtuvo la gráfica de la figura 2.3.

Es claro que el tiempo crece exponencialmente con respecto al número, ya que el número de bits aumenta exponencialmente y en consecuencia también crece el número de operaciones. La tabla 2.3 muestra el tiempo que tomaría calcular de forma exhaustiva la no-linealidad del espacio de funciones balanceadas de n variables.

Es claro que no es posible realizar una búsqueda exhaustiva para valores de $n \geq 7$, incluso si se utilizara cómputo paralelo.

Una función f de n variables es llamada *bent* si $W_f(w) = \pm 2^{\frac{n}{2}}$ para cualquier

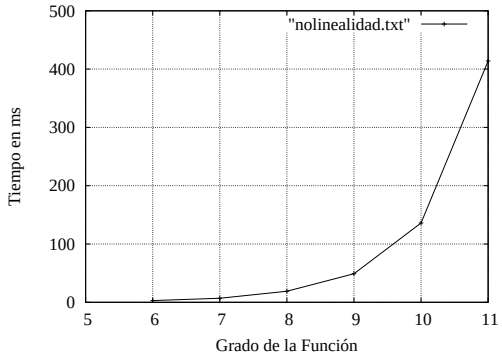


Figura 2.3: Tiempo del cálculo de la no-linealidad

n	Tiempo(nl)	Tiempo(Búsqueda Exhaustiva)
3	3	0.0002387 seg
4	7	0.0602316 seg
5	19	59.4068 mins
6	49	342862 años
7	136	$6.8 \times 10^2 4$ años
8	414	$2.87 \times 10^6 3$ años
9	29.138180	$4.36 \times 10^1 40$ años

Tabla 2.3: Tiempo del cálculo de la no-linealidad

$w \in \{0,1\}^n$. En otras palabras, una función de n variables es llamada *bent* si $nl(f) = 2^{n-1} - 2^{\frac{n}{2}-1}$. Estas funciones son importantes en el ámbito criptográfico dado que éstas poseen la máxima no linealidad posible [11].

Una función $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$ es llamada una función *bent* si cumple

$$2^{-\frac{n}{2}} \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus < B, x >} = \pm 1$$

para todo $B \in \mathbb{F}_2^n$.

De aquí que $f(x) \oplus < B, x >$ es considerada como una función evaluada sobre los reales. La secuencia de una función *bent* es llamada secuencia *bent* [6].

Sea f una función *bent* en $\mathbb{F}_2^{\mathbb{F}_2^n}$ y sea E la secuencia de f, entonces las siguientes sentencias son equivalentes

- f es *bent*
- $< E \cdot l >= \pm 2^{\frac{1}{2}n}$ para cualquier secuencia afín l de longitud 2^n
- $f(x) \oplus f(x \oplus \alpha)$ es balanceada para cualquier vector no cero $\alpha \in \mathbb{F}_2^n$
- $f(x) \oplus < \alpha, x >$ asume el valor $2^{n-1} \pm 2^{\frac{1}{2}n-1}$ para cualquier $\alpha \in \mathbb{F}_2^n$

Sin embargo, desde 1976 se demostró [12] que ninguna función *bent* puede ser balanceada. La máxima no-linealidad posible coincide con el *radio de recubrimiento*[1] de los códigos de Reed-Muller de primer orden [13]. Para $n \geq 15$ impar, el radio de recubrimiento, y en consecuencia la máxima no-linealidad posible, excede el valor de $2^{n-1} - 2^{\frac{n-1}{2}}$ [13]. Seberry mostró que para n impar con $n \geq 29$, es posible construir funciones balanceadas con no-linealidad mayor a $2^{n-1} - 2^{\frac{n-1}{2}}$ [14].

[1] El radio de recubrimiento de un código es el mínimo entero tal que cualquier palabra cuya longitud sea la del código, dista en a lo sumo ese entero a una palabra en el código.

Capítulo 3

Teoría de Gráficas

La teoría de gráficas se ha convertido en una herramienta muy poderosa en la solución de problemas complejos. Las primeras nociones sobre teoría de gráficas surgieron con el trabajo del matemático Leonhard Euler sobre los siete puentes de Königsberg publicado en 1736, en el cual se tienen dos islas en el río de Pregel en Königsberg. Estas están conectadas entre sí y con los márgenes del río por puentes como se muestra en la figura 3.1. El problema consiste en partir desde cualquier lugar y pasar por cada uno de los puentes exactamente una vez y luego volver al punto de partida.

El problema de los siete puentes de Königsberg puede ser considerado como uno de los primeros resultados topológicos en geometría que no depende de ninguna medida y muestra la conexión de la teoría de gráficas y la topología.

Posteriormente, en 1845, Gustav Kirchhoff publicó las leyes del circuito de Kirchhoff para calcular el voltaje y la corriente en circuitos eléctricos.

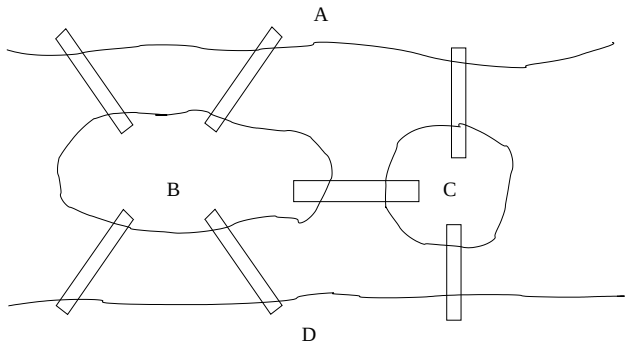


Figura 3.1: Puentes de Königsberg

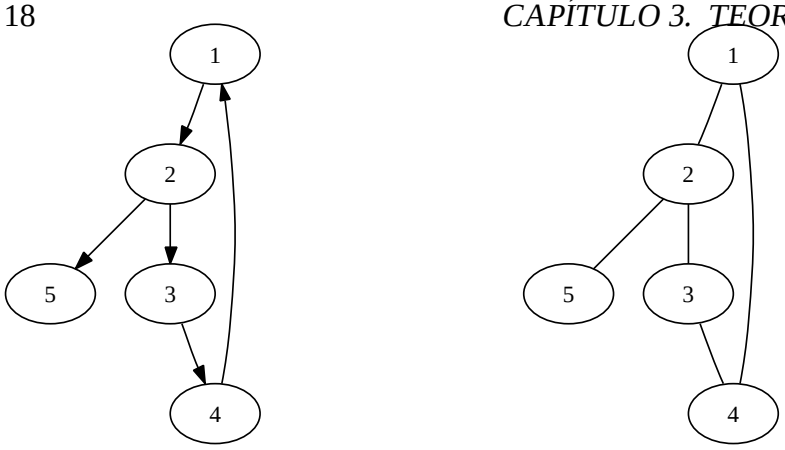


Figura 3.2: Gráfica dirigida y no dirigida

En 1852 Francis Guthrie planteó el problema de los cuatro colores, el cual plantea la pregunta de si es posible colorear con cuatro colores cualquier mapa de países, evitando que el borde de dos países que colinden tengan el mismo color. Este problema, que fue solucionado solamente un siglo más adelante, en 1976, por Kenneth Appel y Wolfgang Haken, puede ser considerado como el que dio pie al nacimiento de la teoría de gráficas.

Ahora bien, una gráfica G consta de un conjunto finito V y una relación binaria sobre V; al conjunto V se le conoce como *vértices* o *nodos* y la relación binaria es representada como una colección E de pares ordenados o como la función $Adj : V \to P(V)$, donde $P(V)$ es el conjunto potencia de V.

Denotaremos por $Adj(v)$ al conjunto de vértice adyacentes de v y la pareja ordenada $(v, w) \in E$ se le conoce como arista. Por lo tanto, $(v, w) \in E$ si y sólo si $w \in Adj(v)$, entonces diremos que w es adyacente a v y w es el punto final de la arista (v, w). Ahora al conjunto $N(v) = v + Adj(v)$ lo llamaremos el vecindario del vértice v

Una gráfica G puede ser representada gráficamente, dibujando un punto por cada $v \in V$ y se dibuja un arco entre dos vértices $v, w \in V$ si $(v, w) \in E$, si los arcos son representados con flechas como se muestra en la figura 3.2, diremos que la gráfica es dirigida.

La *sub-gráfica* de una gráfica G se define como la gráfica H, tal que $H = (V', E')$ con $V' \subset V$ y $E' \subset E$.

Dada una gráfica $G = (V, E)$ y una secuencia de vértices $[v_0, v_1, \cdots, v_m]$, diremos que la secuencia es una *cadena* de longitud m en G si $(v_{i-1}, v_i) \in E$ o $(v_i, v_{i-1}) \in E$ para $i = 1, 2, \cdots, m$. Por otra parte, una secuencia de vértices $[v_0, v_1, \cdots, v_m]$ es un *camino* de v_0 a v_m de longitud m si se cumple que

$(v_{i-1}, v_i) \in E$ para $i = 1, 2, \cdots, m$.

Un camino o cadena en G es llamado *simple,* si ningún vértice aparece más de una vez, lo que resulta trivial si $m = 0$.

Una gráfica G se dice ser *conexa* si entre cualesquiera dos vértices existe una cadena en G que los una, y se dice que G es *fuertemente conexa* si entre cualesquiera dos vértices $x, y \in V$ existe un camino de x a y.

Una secuencia de vértices $[v_0, v_1, \cdots, v_m]$ es un *ciclo* de longitud $m + 1$ ó camino cerrado si $(v_{i-1}, v_i) \in E$ para $i = 1, 2, \cdots, m$ y $(v_0, v_m) \in E$, si $v_i \neq v_j$ para cada $i \neq j$ entonces se dice que el ciclo es *simple*.

La distancia entre dos vértices $v_1, v_2 \in V$ es el número de aristas que hay en el camino más corto que los conecte. Esta métrica también es conocida como distancia geodésica.

La excentricidad ϵ de un vértice $v \in V$ es la distancia más grande entre el vértice v y cualquier otro vértice. El radio de una gráfica es la mínima excentricidad de cualquier vértice. El diámetro de una gráfica es la máxima excentricidad de cualquier vértice en la gráfica. Esto es, la distancia más grande entre cualesquiera dos vértices.

Sea $G = (V, E)$ una gráfica. Definiremos el complemento de G como $\overline{G} = (V, \overline{E})$ donde

$$\overline{E} = \{(x, y) \in V \times V | x \neq y \text{ y } (x, y) \notin E\}$$

Una gráfica se dice ser bipartita, si ésta puede ser expresada como

$$G = (V_1 + V_2, E)$$

donde

- V1 y V2 son distintos y tienen más de un elemento cada uno.
- Una arista en A une un vértice de V1 con uno de V2.
- No existen aristas uniendo dos elementos de V1-análogamente para V2.

Una gráfica G es *completa* si para cualquiera $x, y \in V$ vértices tal que $x \neq y$, x y y son adyacentes y denotaremos con k_n a la gráfica completa de grado n, donde el grado de la gráfica es igual a la cardinalidad de V (la figura 3.3 muestra las gráficas completas de grado 0 hasta grado 6).

Dentro de la teoría se han propuesto diferentes problemas, algunos de éstos tratan de encontrar propiedades en las gráficas que mantienen una relación con

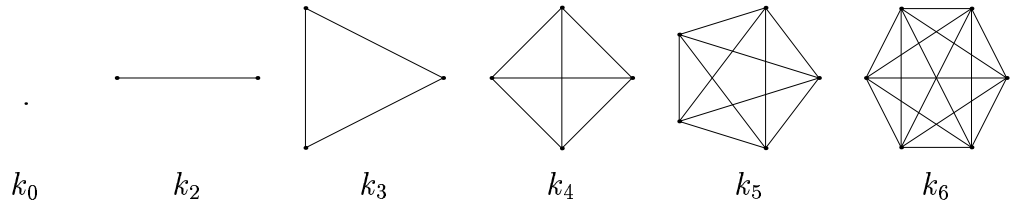


Figura 3.3: Gráficas completas no dirigidas de grado $0 \leq n \leq 6$

respecto a sus sub-gráficas; por ejemplo, está el llamado problema de la sub-gráfica isomórfica, el cual trata de localizar sub-gráficas en alguna gráfica dada.

Algunos problemas tratan de averiguar cuales son las propiedades que puede tener una gráfica si ésta y todas sus sub-gráficas poseen una o más propiedades en común. Algunos otros problemas están relacionados con las rutas que se pueden encontrar en una gráfica, como en el problema de los siete puentes de Königsberg, el problema del camino más corto, el problema del agente viajero el cual es NP-Completo y el camino Hamiltoniano por mencionar algunos.

3.1. Gráfica de Funciones Booleanas Balanceadas

Como se mencionó, es posible solucionar problemas difíciles utilizando gráficas. Es por esta razón que se propone representar el espacio de funciones balanceadas mediante una estructura de gráfica con la esperanza de que esta estructura permita localizar funciones balanceadas con propiedades criptográficas importantes-en este caso la no-linealidad. A dicha gráfica la denotaremos por $\mathcal{G}(n)$ y sus nodos son funciones balanceadas.

Como ya se ha mencionando anteriormente, uno de los problemas más difíciles de tratar es la cardinalidad del espacio de funciones balanceadas debido a que éste crece de manera exponencial. Con el uso de una gráfica se tendrá una estructura más precisa para representar a dicho espacio, y además, se tiene la posibilidad de aplicar la teoría de gráfica a $\mathcal{G}(n)$ para crear nuevos métodos y heurísticas para localizar funciones balanceadas que sean útiles dentro del ámbito de la criptografía.

La gráfica de funciones balanceadas $\mathcal{G}(n)$ se define como la pareja (V, E), donde $V = \mathbb{B}_2^n$(es decir el espacio de funciones balanceadas de n variables) y E es el conjunto de parejas $(f_1, f_2$ en $\mathcal{G}(n)$ tal que $f_1 \oplus f_2$ es también un nodo de $\mathcal{G}(n))$. En otras palabras, para cualesquiera dos nodos f_1 y f_2 en $\mathcal{G}(n)$, existe una arista de f_1 a f_2 si y sólo si $f_1 \oplus f_2$ es una función balanceada.

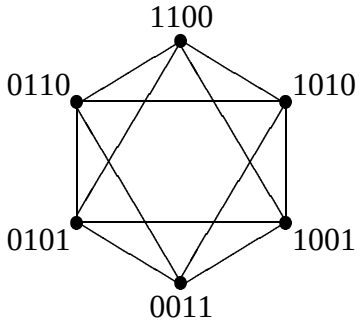


Figura 3.4: La gráfica $\mathcal{G}(2)$.

Por ejemplo, la gráfica $\mathcal{G}(2)$ tiene 6 nodos, éstos pueden ser etiquetados como $1100, 1010, 1001, 0011, 0101, 0110$, y el conjunto de aristas es:

$E = \{$ (1100, 1010), (1100, 1001), (1100, 0101), (1100, 0110),
(1010, 1100), (1010, 1001), (1010, 0011), (1010, 0110),
(1001, 1100), (1001, 1010), (1001, 0011), (1001, 0101),
(0011, 1010), (0011, 1001), (0011, 0101), (0011, 0110),
(0101, 1100), (0101, 1001), (0101, 0011), (0101, 0110),
(0110, 1100), (0110, 1010), (0110, 1001), (0110, 0101)$\}$

Si dibujamos la gráfica colocando cada nodo de la gráfica sobre un hexágono regular en sentido de las manecillas del reloj; $\mathcal{G}(2)$ tiene la apariencia de una estrella de David y cada uno de los nodos está conectado a exactamente 4 vecinos (ver figura 3.4).

Claramente, si (f, g) es un vértice en $\mathcal{G}(n)$, entonces $\triangle(f, g, f + g)$ es un triángulo empotrado en $\mathcal{G}(n)$.

Esta gráfica no es dirigida ya que si (f, g) es un vértice en $\mathcal{G}(n)$ entonces (g, f) también es un vértice en $\mathcal{G}(n)$ por las propiedades del operador $\oplus$.

Para cualquier $n \in N$ se cumple que el número de nodos de la gráfica $\mathcal{G}(n)$ es igual a $\binom{2^n}{2^{n-1}}$ nodos.

El determinar el número de vecinos que tiene cada nodo es simple, sea $(f_1 \in \mathcal{G}(n)$ entonces $(f_2 \in \mathcal{G}(n)$ es adyacente a f_1 si y sólo si

$$card\{Spt(f_1) \cap Spt(f_2)\} = 2^{n-1} \text{ y } card\{Nul(f_1) \cap Nul(f_2)\} = 2^{n-1}$$

Entonces, el número de funciones que cumplen con esta propiedad es igual

Grado	Número de nodos	Número de vecinos por nodo
2	6	4
3	70	36
4	12870	4900
5	601080390	165636900
6	1832624140942590534	361297635242552100

Tabla 3.1: Crecimiento de $\mathcal{G}(n)$

a todas la combinaciones de 2^{n-2} elementos del soporte y la parte nula en las 2^{n-1} posibles posiciones es decir $\binom{2^{n-1}}{2^{n-2}}^2$. Dado que todos los puntos de la gráfica tienen la misma cantidad de vecinos $\mathcal{G}(n)$ es una gráfica regular. En la tabla 3.1 se muestra el crecimiento que tiene la gráfica $\mathcal{G}(n)$.

El número de vecinos que tiene cada punto de la gráfica crece exponencialmente, por lo que se debe ser tomar en cuenta el crecimiento para desarrollar algoritmos de búsqueda.

Una característica de esta gráfica es que permite generar uno o más vecinos de cualquier nodo en $\mathcal{G}(n)$ sin tener que construir todo la gráfica. Esto resulta importante ya que reduce el tiempo de cómputo y almacenamiento con lo que se pueden realizar pruebas o explorar una parte de la gráfica sin tener que generar completamente la gráfica.

Una de las propiedades más importantes de la gráfica $\mathcal{G}(n)$ es que ésta tiene diámetro 2. En otras palabras, para cualesquiera dos funciones balanceadas $f, g \in \mathcal{G}(n)$, entonces una de las siguientes relaciones se cumple :

- $f = g$
- (f, g) es un vértice en la gráfica $\mathcal{G}(n)$
- Existen $h \in \mathcal{B}(n)$ tales que (f, h) y (h, g) son vértices en $\mathcal{B}(n)$

Demostración. Supongamos que $f, g \in \mathcal{G}(n)$ tales que $f \neq g$ y (f, g) no es una arista $\in \mathcal{G}(n)$, i.e. $f + g$ no es balanceada. Sea $u = \text{card}(Nul(f + g))$ y $s = \text{card}(Spt(f + g))$. Al intercambiar los valores $0, 1$ y sin ninguna pérdida de generalidad, podemos asumir que $s < u$. Sea $t = u - s$. Esto no es necesariamente un entero positivo. Los cuatro conjuntos $I_{00} = Nul(f) \cap Nul(g)$, $I_{01} = Spt(f) \cap Nul(g)$, $I_{10} = Spt(f) \cap Nul(g)$ y $I_{11} = Spt(f) \cap Spt(g)$ forman una partición del dominio $\mathbb{F}_2^n$. Sea $i_{00}, i_{01}, i_{10}, i_{11}$ sus respectivas cardinalidades. Claramente: $s = i_{01} + i_{10}$ y $u = i_{00} + i_{11}$. Seleccionamos un conjunto $I_2 \subset I_{00} \cup I_{11}$ que consiste

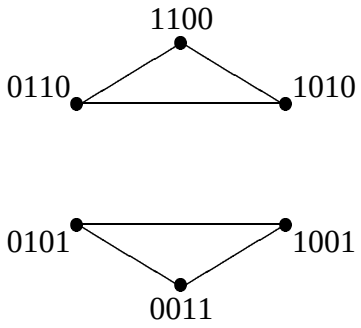


Figura 3.5: La gráfica $\mathcal{G}(2)$.

de exactamente $t/2$ elementos y sea $h \in \mathcal{B}(n)$ tal que $Spt(h) = I_{01} \cup I_{10} \cup I_2$ y $Nul(h) = (I_{00} \cup I_{11}) - I_2$. Entonces $h \in \cap B(n)$ y $(f, h), (h, g)$ son vértices en $\cap G(n)$. □

Una consecuencia de la propiedad anterior es que la gráfica $\mathcal{G}(n)$ es *conexa*; ya que, entre cualesquiera dos vértices $f, g \in V$, existe un camino en $\mathcal{G}(n)$ de longitud a lo sumo 2 que los une.

Dado que $f_1 \oplus f_2 = f_2 \oplus f_1$, la gráfica de $\mathcal{G}(n)$ es simétrica y es posible expresarla como $G = (V_1 + V_2, E)$, donde $V_1 \subset V$ y

$$card\{V_1\} = \frac{card\{V\}}{2} \text{ y } \forall v_1, v_2 \in V_1 \text{ se cumple que } v_2 \neq \overline{v_1}$$

donde para cualquier $v' \in V_2$, existe un elemento $v \in V_1$ tal que $v' = \overline{v}$, en consecuencia $V_1 \cap V_2 = \emptyset$. Retomando la gráfica $\mathcal{G}(2)$ como ejemplo, tenemos que si $\mathcal{G}(2) = (V, E)$ con $V = \{$ 1100, 1010, 1001, 0011, 0101, 0110 $\}$ entonces $V_1 = \{$ 0110 , 1100, 1010 $\}$ y $V_2 = \{$ 1001, 0011, 0101 $\}$ y colocados sobre un hexágono regular en sentido de las manecillas del reloj, se tiene la gráfica de la figura 3.5.

Por otra parte, el complemento de $\mathcal{G}(n)$ es la gráfica $\overline{\mathcal{G}}(n) = (V, \overline{E})$ tal que

$$\overline{E} = \{(f, g) \in V \times V | (f, g) \notin E\} = \{(f, g) \in V \times V | f \oplus g \notin V\}.$$

Tomando la gráfica $\mathcal{G}(2)$ con los nodos 1100, 1010, 1001, 0011, 0101, 0110 y colocados sobre un hexágono regular en sentido de las manecillas del reloj, se tiene la gráfica de la figura 3.6.

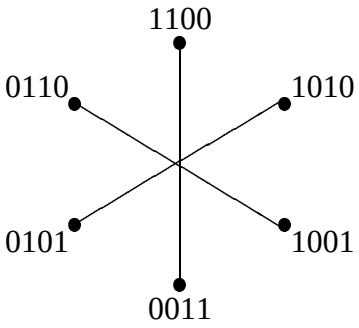


Figura 3.6: La gráfica $\overline{\mathcal{G}}(2)$.

Ésta tiene características similares a la gráfica $\mathcal{G}(2)$, como el número de vecinos que tiene cada nodo, que es muy fácil de calcular. Sabemos que para $\mathcal{G}(2)$ el número de vecinos que tiene cada nodo es igual a $\binom{2^{n-1}}{2^{n-2}}$ y el número de nodos que tiene la gráfica es $\binom{2^n}{2^{n-1}}$ y por lo tanto el número de vecinos por nodo en $\overline{\mathcal{G}}(2)$ es igual a $\binom{2^n}{2^{n-1}} - \binom{2^{n-1}}{2^{n-2}}$.

Aunque la gráfica $\mathcal{G}(n)$ tiene algunas propiedades importantes, resulta claro que ésta no es *completa*, y la demostración es simple.

Demostración. Si una función f está en $\mathcal{G}(n)$ entonces $\overline{f} \in \mathcal{G}(n)$, pero $f \oplus \overline{f}$ no es balanceada y por lo tanto existen funciones $f, \overline{f}$ en la gráfica tal que f y $\overline{f}$ no son adyacentes.

Capítulo 4

Algoritmos de Búsqueda

El problema de construir funciones booleanas con buenas propiedades criptográficas, ha recibido mucha atención por su importancia en el ámbito criptográfico. Los métodos para el diseño y construcción de funciones booleanas y cajas de sustitución pueden ser clasificados en 3 tipos: generación aleatoria, generación aritmética y los métodos heurísticos [2].

Los métodos aleatorios son los más sencillos, ya que evitan especificar propiedades combinacionales que son consideradas criptográficamente útiles.

Dado que la cardinalidad del espacio de funciones booleanas es enorme, no es posible generar funciones booleanas con buenas propiedades criptográficas mediante búsquedas aleatorias. En el experimento realizado en [2], se generaron de forma aleatoria 1000000 de funciones balanceadas de 6 variables y se midió la propiedad CI(1). Del total de funciones generadas solo 55 funciones fueron CI(1). Esto demuestra que la probabilidad de localizar funciones con la propiedad CI(1) utilizando métodos aleatorios es muy baja para valores de n suficientemente grandes.

Ahora, si generamos 1000000 de funciones balanceadas de forma aleatoria y medimos la no-linealidad, se tienen los resultados listados en la tabla 4.1.

Al igual que con la propiedad CI(1), la probabilidad de localizar funciones balanceadas con máxima no linealidad se decrementa cuando el número de variables n aumenta.

La cardinalidad del espacio de funciones balanceadas y la distribución de la no-linealidad son las dos características principales del problema cuyo objetivo es localizar funciones balanceadas con máxima no-linealidad.

Mientras que el espacio de funciones balanceadas de 5 variables o menos puede ser examinado en horas, el espacio de funciones con 6 variables puede ser exa-

n	no linealidad promedio	máx. no linealidad
5	9.23	12
6	21.47	26
7	47.85	52
8	103.50	116
9	219.12	230
10	456.82	470

Tabla 4.1: No linealidad promedio

minado exhaustivamente con la ayuda de un análisis de clases y para funciones de 7 variables la búsqueda exhaustiva no es factible, y una búsqueda exhaustiva parcial puede ser una alternativa; pero para valores de $n \geq 7$ no es posible realizar búsquedas exhaustivas [2].

Por consiguiente se hizo inminente la necesidad de desarrollar métodos de búsqueda más efectivos capaces de localizar funciones balanceadas con máxima no-linealidad para un número de variables $n \geq 7$.

Como una alternativa a los métodos aleatorios surgieron los métodos algebraicos, los cuales permiten construir funciones con propiedades criptográficas específicas, incluyendo a las funciones balanceadas. Estos métodos se basan en las propiedades de la funciones y en la construcción de operadores para generar funciones criptográficamente útiles.

Como ejemplo de estos métodos, está el desarrollado por Patterson y Wiedeman, con el que construyeron funciones de 15 variables con no linealidad y peso de Hamming igual a 16492.

Posteriormente, Zhang y Zheng usaron las funciones localizadas con el método de Patterson y Wiedeman para construir funciones booleanas balanceadas con no linealidad mayor a $2^{n-1} - 2^{\frac{n-1}{2}}$ para n impar mayor a 29 [13].

Por su parte, Dobbertin utilizó un procedimiento recursivo para modificar una clase de funciones *bent* con el que obtuvo funciones balanceadas con máxima no linealidad para un número de variables impar. Un caso especial de este procedimiento consiste en modificar la clase de funciones de Mairona-McFarland. Dobbertin conjeturó que para cualquier n par la máxima no linealidad de las funciones booleanas balanceadas satisface que la máxima no-linealidad de las funciones booleanas balanceadas es igual a la ecuación

$$nlbMax(n) = 2^{n-1} - 2^{\frac{n}{2}} + nlbMax(\frac{n}{2}),$$

Espacio	$\mathbb{F}_2^4$	$\mathbb{F}_2^6$	$\mathbb{F}_2^8$	$\mathbb{F}_2^10$	$\mathbb{F}_2^12$	$\mathbb{F}_2^14$
Cota superior	4	26	118	494	2014	8126
Modificación	4	26	116	492	2010	8120
Concatenación	4	24	112	480	1984	8064

Tabla 4.2: Método algebraico

donde $nlbMax$ representa la máxima no linealidad de las funciones booleanas balanceadas de n variables [13].

Por otra parte, Dobbertin, Shakar y Maitra demostraron que dada una función balanceada $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$, con no linealidad igual a x, es posible construir una función booleana balanceada $f' \in\in \mathbb{F}_2^{\mathbb{F}_2^n}$ tal que la no linealidad de f' es mayor a $x - 2$ y grado igual a $n - 1$. A partir de este resultado los autores obtuvieron como resultado funciones booleanas balanceadas de 15 variables con no linealidad mayor a 16256 y demostraron que es posible construir una función $f' \in \mathbb{F}_2^{F^{15}}$ con no linealidad 16276 [13].

Shakar y Maitra desarrollaron dos algoritmos recursivos para construir funciones booleanas balanceadas con no linealidad mayor a $2^{n-1} - 2^{\frac{n-1}{2}}$ para $n = 15, 17, 19, 21, 23, 25, 27$, utilizando funciones *bent* como funciones de entrada para los algoritmos y para n impares con $n \geq 29$. Estos algoritmos son una variante del método de Patterson y Wiedemann para obtener balance manteniendo una no linealidad mayor a la obtenida con la concatenación *bent*, utilizando como entrada funciones booleanas balanceadas con no linealidad mayor a 2^{n-1} [13].

Seberry, Zhang y Zheng mostraron que con la concatenación, partición, alteración y multiplicación (en el sentido de Kronecker) es posible producir funciones booleanas balanceadas a partir de la modificación y multiplicación de secuencias, para alcanzar una no linealidad mayor a la obtenida con métodos de construcción previos. Los resultados de este método son los listados en la tabla 4.2 [6].

Por otra parte, se tiene que una función *bent* no es balanceada y en 1976 Rothaus demostró que cualquier secuencia de longitud 2^{2k} tienen un peso de Hamming de $2^{2k-1} \pm 2^{k-1}$, y demostró que, para cualquier número n, las funciones bent son construidas de la siguiente forma:

Sea $n = 2m$ entonces las funciones de la forma

$$f(x_1, x_2, \ldots, x_n) = g(x_1, x_2, \ldots, x_n) + x_1 x_{m+1} + \ldots + x_2 x_{m+1} + \ldots + x_m x_n$$

son bent, donde $g(x_1, x_2, \ldots, x_n)$ es una función completamente aleatoria de m variables [15].

El método de construcción de funciones se basa en este teorema, simplemente cambiando 2^{k-1} bits en la secuencia bent, haciendo que la función resultante sea balanceada. El método utilizado en [15] complementa 2-bits cada ciclo hasta que la función sea balanceada. En este caso es necesario $C^2_{2^{k-1}} + C^2_{2^{k-3}} + \ldots + C^2_{2^{k-1}+2^{k-2}}$ rondas y sólo esto se ejecuta una sola vez.

Los métodos algebraicos son directos y tienen un buen desempeño, pero estos presentan un problema cuando se desea generar funciones booleanas con dos o más propiedades criptográficas; esto se debe a que al optimizar una propiedad de las funciones como la no-linealidad, otras propiedades criptográficas como la auto correlación son ignoradas. Es por esta razón que algunos investigadores consideran que los métodos algebraicos no proporcionan una variedad suficiente de funciones con una combinación de propiedades criptográficas.

Una alternativa a los métodos algebraicos son los métodos heurísticos, los cuales tienen la habilidad de optimizar una función con respecto a más de una propiedad criptográfica. Estos métodos tienen la habilidad de localizar funciones booleanas con propiedades superiores a los generados con los métodos algebraicos. Los métodos heurísticos incluyen *hill climbing*, algoritmos genéticos o una combinación de éstos.

Uno de los métodos que han reportado un buen desempeño ha sido el *Hill Climbing*. En el año de 1997 Millan propuso la localización de funciones con alta no-linealidad y auto-correlación utilizando el método de *hill climbing*. La idea básica del método propuesto por Millan es alterar ligeramente una función booleana para aumentar la no-linealidad o la auto-correlación. Los resultados reportados por Millan muestran que el *hill climbing* puede mejorar la no linealidad considerablemente en comparación con los métodos aleatorios [15].

Por otra parte, Clark y Jacob en el año 2000 [2], introdujeron el uso de *recocido simulado* (Simulated Annealing) en el diseño de funciones con propiedades criptográficas útiles. Este es un tipo de búsqueda local que fue propuesto por Kirck Patric a inicios de los 80's. Este método de búsqueda está inspirado en los procesos de enfriamiento del fundido de metales. Una característica importante del recocido simulado es que es un método de búsqueda probabilístico que puede escapar a los óptimos locales. En la siguiente sección se describirá con más detalle este método de búsqueda.

John Clark, Jacob, Maiyta y Stancia [5] desarrollaron un método de búsqueda para localizar funciones booleanas balanceadas con alta no-linealidad, utilizando el recocido simulado, el cual consiste en cambiar el valor de 2-bits en la tabla de verdad de la función actual para generar el espacio de búsqueda y la elección de los bits está basada en las propiedades de la transformada de Walsh.

Una combinación de los métodos de *hill climbing* y recocido simulado fue desarrollado en [16] para optimizar la no linealidad, auto-correlación y grado algebraico en funciones booleanas. Este método utiliza el algoritmo de recocido simulado para minimizar el valor de la transformada de Walsh-Hadamard. La función generada por el recocido simulado es utilizada como función inicial del *hill climbing*, y es mediante el *hill climbing* con el que se optimizan las propiedades de no-linealidad y la auto-correlación.

4.1. Recocido Simulado

Un problema de optimización es un problema tanto de minimización como de maximización. Un problema de optimización está especificado por un conjunto de instancias. Una instancia de un problema de optimización puede ser formalizado como una pareja $(S;f)$, donde S denota al conjunto finito de todas las posibles soluciones y f denota a la función de costo, la cual es una asociación definida como:

$$f : S \to \mathbb{R}.$$

Cuando se habla de minimización, el problema consiste en localizar una solución $i_{opt} \in S$, tal que

$$f(i_{opt}) \leq f(i), \text{ para todo } i \in S$$

y en el caso de maximización, el problema es localizar una solución $i_{opt} \in S$ tal que

$$f(i_{opt}) \geq f(i), \text{ para todo } i \in S,$$

donde el elemento i_{opt} es llamado *solución óptima global*, *máximo* o *mínimo*, o simplemente *óptimo*, $f(i_{opt})$ denota el costo del óptimo y S_{opt} el conjunto de soluciones óptimas.

Sea (S, f) una instancia de un problema de optimización, una estructura de vecinos es una asociación

$$N : S \to 2^S$$

la cual está definida para cada solución $i \in S$ como un conjunto de soluciones $S_i \subset S$. El conjunto S_i es llamado el vecindario de la solución i, y cada $j \in S_i$ es llamado un vecino de i [17].

El problema de localizar óptimos ha sido tratado con diferentes enfoques con el propósito de desarrollar algoritmos de búsqueda capaces de localizar óptimos globales entre un número de soluciones sin explorar todo el espacio de búsqueda.

A principios de los ochenta, Kirkpatrick, Gelatt, Vecchi e independientemente Cerny introdujeron el concepto del recocido en los problemas de optimización combinatorial. Este concepto está basado en una fuerte analogía entre los fenómenos físicos del recocido.

En el proceso de condensación de materiales, el recocido es conocido como un proceso termal para obtener un estado de energía bajo a partir de un sólido en una temperatura alta. Este proceso consta de dos pasos:

- Incrementar la temperatura del material sólido hasta un valor máximo.
- Decrementar cuidadosamente la temperatura del sólido hasta que las partículas del sólido se hayan reordenado en el estado base del sólido.

En 1953, Metrópolis, Rosenbluth y Teller introdujeron un algoritmo simple para simular la evolución de un sólido en altas temperaturas hasta un estado de equilibrio termal. Este algoritmo está basado en técnicas Monte Carlo, el cual genera una secuencia de estados a partir del sólido de la siguiente forma. Dado un estado actual i del sólido con energía E_i, entonces un estado subsecuente j es generado al aplicar un mecanismo de perturbación que transforma el estado actual a un estado siguiente con una pequeña distorsión. La energía del siguiente estado es E_j. Si la diferencia $E_j - E_i$ es menor o igual a cero entonces el nuevo estado es aceptado como el estado actual. En otro caso, el estado j es aceptado con una cierta probabilidad

$$exp\left(\frac{E_i - E_j}{k_b T}\right)$$

donde T denota la temperatura del sólido, k_B es una constante física conocida como la constante de Boltzmann.

La regla de aceptación descrita es conocida como el criterio de *Metrópolis*, y el algoritmo que se basa en este criterio es conocido como *Algoritmo de Metrópolis* [17].

Es posible aplicar el algoritmo de Metrópolis para generar una secuencia de soluciones de un problema de optimización; para hacer esto, en primer lugar se asume una analogía entre los sistemas físicos multi-partículas y un problema de optimización basado en la siguiente equivalencia:

- Las soluciones del problema de optimización son equivalentes a los estados del sistema físico.
- El costo de una solución es equivalente a la energía del estado.

El algoritmo del recocido simulado puede ser visto como una iteración del algoritmo de Metrópolis, evaluado con el parámetro de control de energía. Entonces se debe asumir una estructura de vecinos y un mecanismo de generación. Si (S, f) denotan una instancia de un problema de optimización e i y j son dos soluciones con costo $f(i)$ y $f(j)$ respectivamente, entonces el criterio de aceptación determina cuando j es aceptado a partir de y al aplicar la siguiente probabilidad de aceptación

$$P_c\{\text{se acepta } j\} = \begin{cases} 1 & \text{si } f(j) \leq f(j) \\ exp\left(\frac{f(i)-f(j)}{c}\right) & \text{si } f(j) \geq f(j) \end{cases}$$

donde $c \in \mathbb{R}^+$ denota el parámetro de control. Los mecanismos de generación corresponden al mecanismo de perturbación en el algoritmo de Metrópolis y el criterio de aceptación corresponde al criterio de Metrópolis. A diferencia de otros algoritmos de búsqueda como el de búsqueda local, el algoritmo de recocido simulado puede escapar de los óptimos locales [17].

El algoritmo de recocido simulado queda delineado en 1.

Algoritmo 1 *Recocido Simulado*

begin
f_i = solución inicial;
repeat
  for $l := 1$ to k **do**
    seleccionar $f_j \in \mathbb{F}_2^n$;
    if $nl(f_j) \leq nl(f_i)$ **then**
      $f_i := f_j$
    else
      if $exp\left(\frac{1}{c_k}\left(nl(f_i) - nl(f_j)\right)\right) > random[0,1)$ **then**
        $f_i := f_j$
      end if
    end if
  end for
  $c_k := c_k \cdot \alpha$
until Criterio de paro
end

Para que el algoritmo muestre un buen desempeño, éste necesita una estructura de vecindario adecuada y parámetros de control adecuados. Estos parámetros

son la cardinalidad del vecindario en el que itera, temperatura inicial, un criterio de enfriamiento y el criterio de paro. Desafortunadamente, no existe un método específico para establecer dichos parámetros y como primer paso lo que se hizo fue desarrollar un conjunto de pruebas para establecer parámetros que fueran apropiados. Dichas pruebas consistieron en calcular la cardinalidad del espacio de búsqueda a través del algoritmo de búsqueda local y de proponer valores iniciales para la temperatura inicial y el criterio de paro.

4.2. Cálculo de Parámetros

El primer parámetro que se estableció fue la cardinalidad del vecindario. Para esto, se desarrollaron un conjunto de pruebas utilizando el algoritmo de *búsqueda local*, el cual es un algoritmo de aproximación, que está basado en una serie de pasos en los que se mejora el valor de la función de costo actual mientras se explora sobre un conjunto de vecinos. El uso de este tipo de algoritmos presupone la definición de las soluciones, una función de costo y una estructura de vecinos.

El algoritmo de búsqueda local itera sobre un número de soluciones o vecindario, éste comienza con una solución inicial la que generalmente, es seleccionada de forma aleatoria; posteriormente, se aplica continuamente un mecanismo de bajo costo que trate de encontrar una mejor solución buscando en el vecindario de la solución actual. Si se encuentra una mejor solución a la actual, ésta es remplazada con la nueva solución y en otro caso, el algoritmo continúa con la solución actual. El algoritmo termina cuando no se puede obtener una mejor solución con respecto a la que se tiene.

Un concepto importante en el análisis del algoritmo de búsqueda local es el de *óptimo local*. Sea (S, f) una instancia de un problema de optimización y sea N una estructura de vecinos, entonces $\hat{i} \in S$ es llamado *solución óptima local* o simplemente, *óptimo local* con respecto a N si es mejor o igual la solución de todos los vecinos [17].

El algoritmo de búsqueda local se caracteriza por localizar usualmente un óptimo local a menos que la estructura del vecindario sea exacta; por esta razón, este algoritmo no garantiza el localizar el óptimo global. La calidad del óptimo obtenido a partir del algoritmo de búsqueda local, usualmente depende de la solución inicial y éste se encuentra expresado en el algoritmo 2. El único parámetro del sistema es la cardinalidad del vecindario k [17].

Para determinar la cardinalidad del vecindario en el algoritmo de búsqueda local, se diseñó un experimento que consta de dos partes. La primera, consiste

Algoritmo 2 *Búsqueda Local*

begin
f_c = función inicial;
repeat
seleccionar $F \subset \mathbb{F}_2^{\mathbb{F}^n}$; $card(F) = k$
if $\exists f \in F : \ nl(f) \geq nl(f_c)$ **then**
$f_c := f$
end if
until ninguna mejora se haya hecho
end

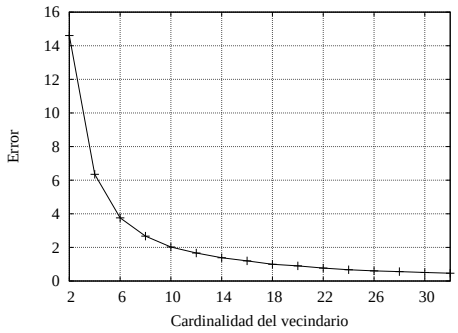


Figura 4.1: Gráfica de error del algoritmo de búsqueda local

en la inicialización del sistema. Esto es, generar de forma aleatoria un espacio de búsqueda $S \subset \mathbb{N}$ con cardinalidad m (si en el momento de generar S siempre se va almacenando el óptimo global, entonces, al finalizar la generación del espacio se conoce el óptimo global) y se inicializa a k la cardinalidad del vecindario con valor igual a n_0.

La segunda parte del experimento consistió en ejecutar el algoritmo de búsqueda local con el parámetro k. Al final de la ejecución se midió la distancia del valor obtenido con respecto al óptimo global, se incrementó el valor de k y se ejecutó nuevamente el algoritmo de búsqueda local hasta que k fuera igual a un valor n_f. La figura 4.1 muestra la distancia de la solución localizada por el algoritmo de búsqueda local al óptimo global; a dicha distancia le llamaremos error.

Los resultados obtenidos con este primer experimento muestran que el desempeño del algoritmo de búsqueda local está relacionado con la cardinalidad del vecindario sobre el cual itera. Si la cardinalidad k es mayor entonces el error que

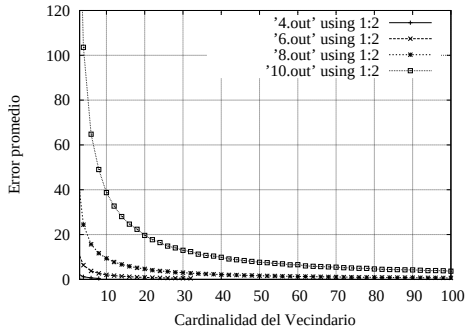


Figura 4.2: Gráfica de error del algoritmo de búsqueda local para $5 \leq n \leq 12$

se obtendrá será menor; pero, también se puede ver en la figura 4.1 que a partir de un cierto valor de k el error ya no disminuye de forma considerable y el crecimiento se vuelve asintótico.

Este experimento se extendió a espacios de búsqueda más grandes donde el comportamiento observado fue el mismo que en el anterior. Como se muestra en la figura 4.2, al aumentar la cardinalidad k del vecindario el error disminuye y el error no disminuye a partir de ciertos valores.

Por otra parte, en ciertos problemas no es posible aumentar demasiado la cardinalidad del vecindario, ya que si el tiempo que toma calcular la función de costo es muy alto, la ejecución del algoritmo de búsqueda local no es factible.

Tomando en cuenta el crecimiento doblemente exponencial del espacio de funciones y con los resultados obtenidos experimentalmente, se estimó que el mejor valor para el número de vecinos muestreados k sea fijado por el polinomio cúbico 4.1.

$$f(n) = 1.30381n^3 - 14.8007n^2 + 50.8046n - 44.2. \tag{4.1}$$

Este polinomio de grado 3 se utilizó como la cardinalidad del vecindario para el algoritmo de recocido simulado.

Por otra parte, la temperatura inicial c_k y el criterio de enfriamiento son parámetros importantes debido a que la función de aceptación depende de éstos y en consecuencia el desempeño del algoritmo de recocido simulado.

Debido a que no existe un método preciso para establecer los valores para la temperatura inicial y el criterio de enfriamiento, el valor de la temperatura inicial se propuso de forma aleatoria y se seleccionaron los valores cuyos resultados fue-

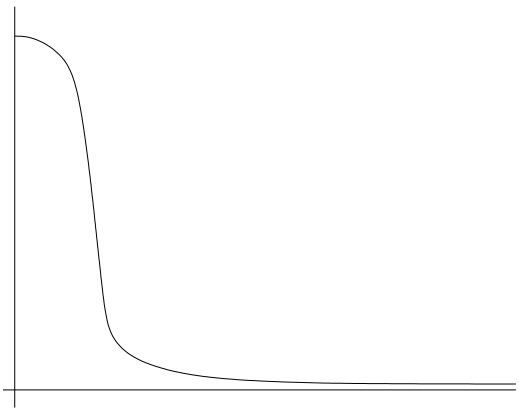


Figura 4.3: Enfriamiento del sistema

ron los mejores y esto se hizo en paralelo con el cálculo de los otros parámetros. Por otra parte, el criterio de enfriamiento determina con qué rapidez se estabiliza el sistema. Si el enfriamiento del sistema es muy rápido, la probabilidad de aceptar un nuevo estado que no mejore la solución actual tiende a ser cero rápidamente, por lo que el número de puntos que no mejoraron la solución actual es reducido y esto puede afectar de forma negativa el desempeño del algoritmo, ver figura 4.3.

Por otra parte, si el enfriamiento es muy lento como se muestra en la figura 4.4, el sistema aceptará estados que no mejoren la solución actual durante un periodo de tiempo prolongado y por consiguiente es necesario ejecutar una gran cantidad de veces el algoritmo de Metrópolis para localizar óptimos. De lo contrario, hay una probabilidad muy alta de que algoritmo arroje como resultado un óptimo local.

El criterio de enfriamiento que se utilizó es el llamado criterio de enfriamiento geométrico, el cual consta de multiplicar la temperatura del sistema por una constante c [5]. En nuestro caso la constante que mejores resultados arrojó fue $c = 0.85$.

El siguiente parámetro a establecer es el criterio de paro. Este es un parámetro que es difícil de establecer ya que, en algunos problemas, no es posible determinar cuándo se ha encontrado un óptimo global.

Un primer método para determinar cuándo se ha encontrado un óptimo global es considerar la distancia entre la solución actual y la solución anterior. Es decir, si después de un cierto número de iteraciones del algoritmo de recocido simulado la distancia entre la solución anterior y la solución actual es menor a una constante ϵ, entonces suponemos que el algoritmo ha llegado a un óptimo global y que la

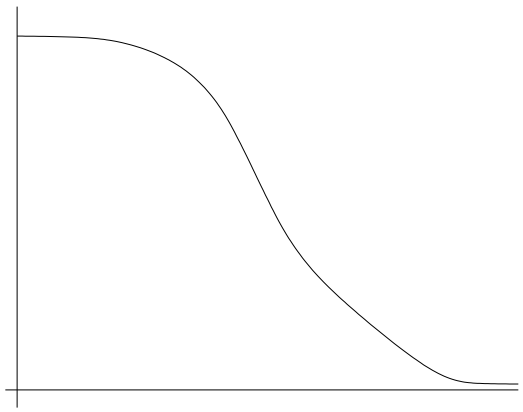


Figura 4.4: Enfriamiento del sistema

solución no puede ser mejorada, como se ilustra en la figura 4.5.

Pero este criterio de paro no es la mejor opción, ya que, si después de un cierto número de iteraciones el algoritmo se encuentra en un óptimo local y no se tiene una selección de puntos adecuada, el algoritmo podría tomar un punto cuya distancia al óptimo local sea menor a la constante ϵ y entonces el algoritmo terminaría dando como resultado un óptimo local como se muestra en la figura 4.6

Un segundo criterio de paro es el considerar la distancia que hay entre dos puntos y el valor de la función objetivo de éstos; es decir, dados dos puntos $i, j \in S$ tal que la distancia del punto i a j es menor a una constante ϵ_0 y la distancia entre $f(i)$ y $f(j)$ es menor a una constante ϵ_0, entonces consideraremos que el algoritmo ha localizado un óptimo global, como se muestra en la figura 4.7
este criterio de paro tampoco resulta del todo efectivo, ya que si después de un cierto número de iteraciones el algoritmo de recocido simulado cae en el vecindario de un óptimo-local como se muestra en la figura 4.8; el algoritmo dará como resultado un óptimo local.

Como ya se ha visto el determinar cuándo el algoritmo ha localizado un óptimo global no es tarea sencilla y este problema también depende de las propiedades que poseen los elementos del espacio de búsqueda y la función de costo, y el tener una selección de puntos adecuada para generar el vecindario.

Sin embargo, un tercer criterio de paro se basa en el número de iteraciones que ejecute el algoritmo de Metrópolis. Este criterio no se basa en las propiedades de los puntos del espacio de búsqueda o de la función objetivo. Por esta razón el algoritmo evita caer en alguno de los casos anteriores. El utilizar este criterio implica establecer cuál es el número necesario de iteraciones del algoritmo de

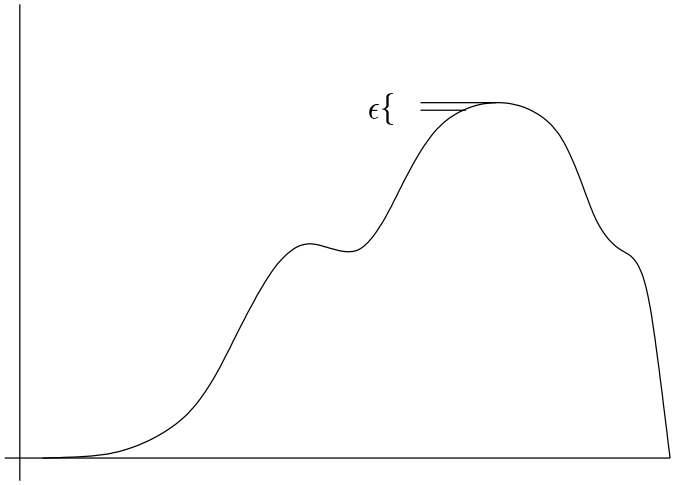


Figura 4.5: Criterio de paro uno

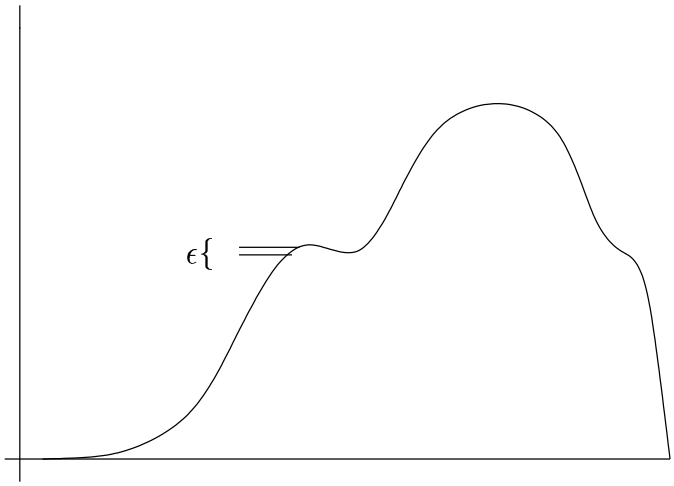


Figura 4.6: Criterio de paro uno

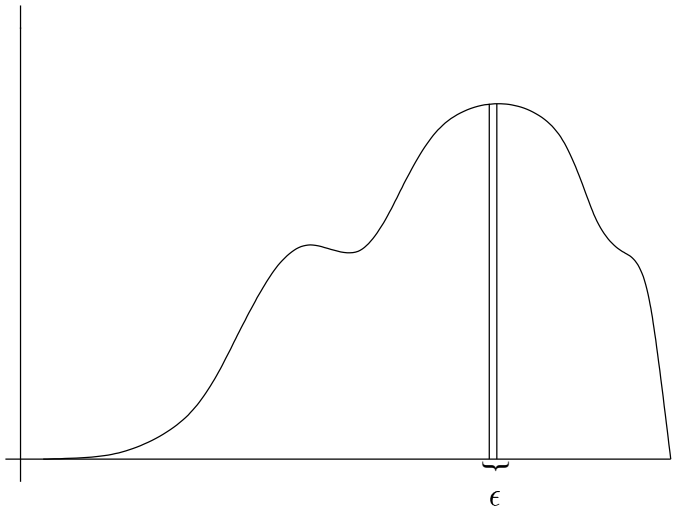


Figura 4.7: Criterio de paro dos

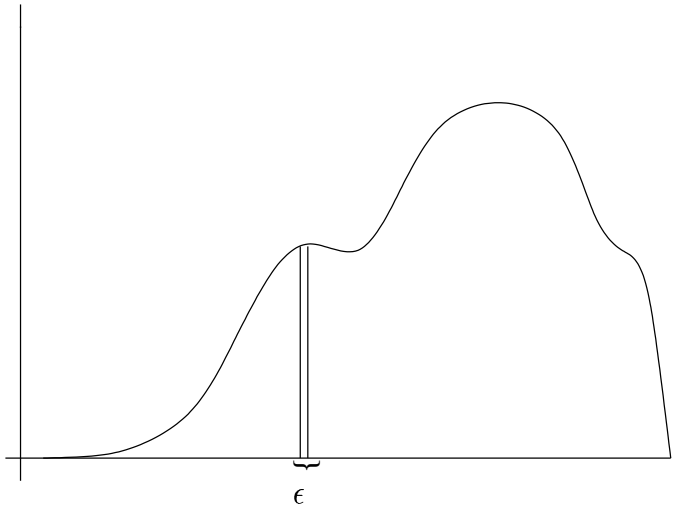


Figura 4.8: Criterio de paro dos

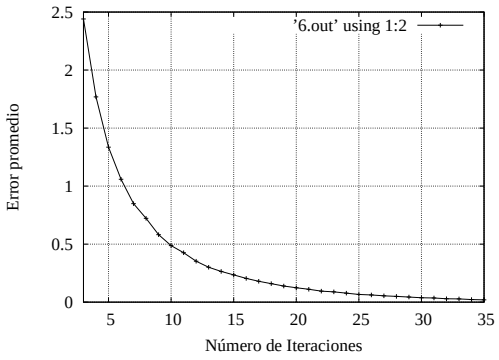


Figura 4.9: Gráfica de error del algoritmo de recocido simulado

Metrópolis para localizar óptimos globales.

Para determinar el número de iteraciones necesarias para localizar un óptimo global con el recocido simulado, se utilizó el esquema del experimento anterior el cual consta de dos partes; en la primera parte inicializamos el sistema generando el espacio de búsqueda con cardinalidad m, el valor del óptimo global, y se inicializó la cardinalidad del vecindario k con el polinomio 4.1.

La segunda parte del experimento consistió de ejecutar el algoritmo de recocido simulado con temperatura inicial c_k, y criterio de paro $k = k_0$; al final se mide el error que hay de la solución obtenida al óptimo global. Posteriormente se incrementa el valor de k y se ejecuta nuevamente el algoritmo de recocido simulado hasta que k sea igual a un valor n_f. La figura 4.9 muestra el error que existe del óptimo localizado por el algoritmo de recocido simulado al óptimo global.

Este primer experimento muestra el desempeño del algoritmo de recocido simulado, el cual tiene una similitud con el comportamiento del algoritmo de búsqueda local. Mientras más grande sea el número de iteraciones del algoritmo de Metropolis el error disminuye, es decir mientras el valor de k sea mayor se tendrá un error menor; pero, también se puede observar que a partir de cierto valor de k el error ya no disminuye de forma considerable y el crecimiento se vuelve asintótico.

Al extender este experimento a espacios de búsqueda más grandes, se observo el mismo comportamiento, como se muestra en la figura 4.10.

A partir de estos experimentos, se probaron diferentes valores para la temperatura inicial c_k y el valor de c_k, que se han fijado por el polinomio 4.2

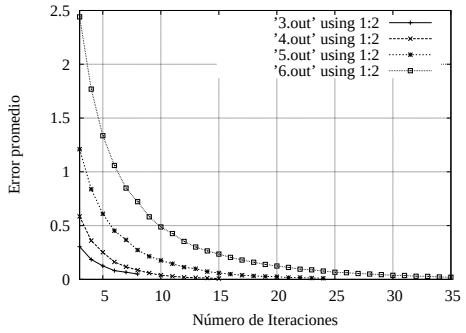


Figura 4.10: Gráfica de error del algoritmo de recocido simulado

$$\begin{aligned} c_k &= 0.0731352\ n^4 - 1.08566\ n^3 + 5.17046\ n^2 \\ &\quad -8.24651\ n + 4.33334 \end{aligned} \tag{4.2}$$

y el criterio de paro del algoritmo se ajusta con el polinomio de tercer grado 4.3

$$f(n) = 1.30381n^3 - 14.8007n^2 + 50.8046n - 44.2. \tag{4.3}$$

La metodología presentada en esta sección no es la única forma de encontrar los parámetros del sistema, pero permite observar el comportamiento del sistema.

4.3. Optimización de la No Linealidad de las Funciones Balanceadas

La optimización de la no-linealidad en funciones sean balanceadas o no, ha recibido mucha atención por la importancia que tienen éstas en el desarrollo de criptosistemas resistentes a los cripto-análisis, como el lineal o el diferencial. El problema de localizar funciones booleanas balanceadas con máxima no linealidad puede ser definido como la pareja ($\mathbb{B}_2^n$;nl), donde $\mathbb{B}_2^n$ denota al conjunto finito de todas las posibles soluciones; es decir, el espacio de las funciones balanceadas de n variables, nl denota a la función costo (la función de no-linealidad) y el objetivo es localizar las funciones $f \in \mathbb{B}_2^n$ tal que

$$nl(f) \geq nl(g), \text{ para cualquier } g \in S.$$

x_1	x_2	x_3	$f(x)$
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	1
1	1	1	0

Tabla 4.3: Tabla de verdad

Para aplicar los algoritmos de búsqueda local 2 y recocido simulado 1 definidos en la sección anterior, es necesario tener una representación adecuada de las funciones booleanas.

Sea f una función balanceada de n variables entonces definimos el *vector de verdad* W_f de la función f como el $W_f \in \mathbb{F}_2^n$, tal que $W_f =< w_0, w_1, \cdots, w_{2^n} >$ donde $w_i \in \mathbb{F}_2$ para $i = 0, \cdots 2^n$, tal que

$$w_0 = f(0,0,\cdots,0), w_1 = f(0,0,\cdots,1), \cdots w_{2^{n-1}} = f(1,1,\cdots,0), \\ w_{2^n} = f(1,1,\cdots,1)$$

por ejemplo, sea f una función balanceada de n variables y sea 4.3 su tabla de verdad, entonces el vector de verdad de f es

$$W_f = \begin{pmatrix} \text{f(000)} & \text{f(001)} & \text{f(010)} & \text{f(011)} & \text{f(100)} & \text{f(101)} & \text{f(110)} & \text{f(111)} \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

A partir de la definición de vector de verdad, es posible asociar una función a un número entero $m \in \mathbb{N}$, tal que la representación en base 2 de m es igual a W_f. Por ejemplo, tomando la función balanceada cuya tabla de verdad es 4.3; entonces, la representación en base 2 de m es igual a 01010110 y por lo tanto el $m_{10} = 86$; si lo representamos en hexadecimal entonces $m_{16} = 56$.

Utilizando esta convención, se tiene que para representar a una función de n variables, se necesita un número entero con 2^n bits, ésto representa un problema; ya que, para una función de 4 bits es necesario un numero de 32 bits; pero para $n > 5$, los tipos básicos del lenguaje C no son suficientes y es necesario definir

$$\text{m} = \underbrace{0_0 \quad 0_1 \quad \cdots \quad 0_{2^{n-1}-1}}_{\text{0's}} \quad \underbrace{0_{2^{n-1}} \quad \cdots \quad 0_{2^n-2} \quad 0_{2^n-1}}_{\text{1's}}$$

$$0_0 \quad 0_1 \quad \cdots \quad 0_{2^{n-1}-1} \qquad 1_0 \quad 1_1 \quad \cdots \quad 0_{2^n-1}$$

Figura 4.11: Generación de funciones balanceadas

un nuevo tipo de dato o utilizar una biblioteca de funciones que permita manejar números enteros grandes, y por esta razón se opto por utilizar la biblioteca *GMP* (**G**NU **M**ultiple **P**recision Arithmetic Library) de *GNU*. Esta biblioteca fue escrita en lenguaje *C*, y opera sobre números enteros, racionales y números de punto flotante; esta biblioteca tiene 140 funciones aritméticas y lógicas para números enteros con signo, y permite expresar números enteros de gran tamaño.

Una vez establecidos los algoritmos de búsqueda y sus parámetros, y la representación de las funciones es necesario establecer un mecanismo para generar funciones balanceadas, y así construir el vecindario sobre el cual iteran los algoritmos de búsqueda.

Este primer método de generación de funciones balanceadas se basa en una selección aleatoria de puntos. Si $m_n \in \mathbb{N}$ es la representación binaria de una función balanceada f de n bits éste tiene 2^n, de los cuales la mitad de éstos, es decir 2^{n-1} bits, debe ser igual a 1 y por lo tanto, los otros 2^{n-1} bits, deben ser 0. Dado que m_n es un número entero, entonces, cuando éste es igual a 0 equivale a que éste tenga 2^n bits igual a cero y entonces sólo es necesario seleccionar 2^{n-1} bits para asignarles el valor 1.

Algoritmo 3 *Selección Aleatoria*

begin
n = número de variables;
asignar $f = 0_0 0_1 \cdot 0_{2^n}$;
seleccionar S con $card(S) = 2^{n-1}$ de f
intercambiar los valores de S
end

Este método para construir funciones balanceadas está expresado en el algoritmo 3. Al aplicar los algoritmos de búsqueda local y recocido simulado generando el espacio de búsqueda mediante el algoritmo anterior, se obtuvieron los resultados mostrados en la tabla 4.4.

Los resultados obtenidos con ambos algoritmos supera el promedio de la selección aleatoria, pero éstos no alcanzan la máxima no-linealidad reportada en

n	no linealidad promedio	Búsqueda Local	Recocido Simulado
5	9	10	10
6	21	24	24
7	47	50	52
8	103	108	108
9	219	226	226
10	456	462	466
11	941	954	956
12	1926	1946	1946
13	3916	3940	3940
14	7927	7966	7966
15	15995	16034	16038
16	32199	32274	32268

Tabla 4.4: Resultados preliminares

n	$card\{\mathbb{B}_2^n\}$	Funciones máxima no-linealidad	máxima no-linealidad
2	6	6	0
3	70	56	2
4	12870	10920	4
5	601080390	8693888	12

Tabla 4.5: Máxima no Linealidad de funciones con n variable ($2 \leq n \leq 5$)

funciones balanceadas. El utilizar un enfoque aleatorio en la búsqueda de funciones balanceadas con máxima no linealidad no es suficiente; ya que se tiene un espacio de búsqueda con cardinalidad $\binom{2^{n-1}}{2^{n-2}}^2$ y con una cantidad reducida de óptimos globales, la probabilidad de localizar óptimos globales tiende a ser cero rápidamente. Para tener una idea de la cantidad de óptimos globales que hay en el espacio de funciones balanceadas con n variables, se realizó una búsqueda exhaustiva con $2 \leq n \leq 5$ ya que para $n \geq 6$ la búsqueda exhaustiva tomaría demasiado tiempo. Los resultados encontrados se listan en la tabla 4.5.

Para resolver este problema se tienen dos opciones. La primera es aumentar la cardinalidad del vecindario en relación a la cardinalidad del espacio de búsqueda; lo que resulta en incremento del tiempo de cómputo y para valores de n suficientemente grandes esto ya no es viable. Una segunda opción es utilizar una estructura de vecindario más precisa con la cual sea posible aumentar la probabilidad de que

los algoritmos de búsqueda converjan hacia los óptimos globales.

La estructura que se utilizó para mejorar el desempeño fue la gráfica de funciones balanceadas, la cual se definió en el capítulo 3 página 21; ésta cubre a todo el espacio de funciones balanceadas y es posible generar la gráfica en cualquier momento.

Para utilizar la gráfica de funciones con los algoritmos de búsqueda, se implementó un algoritmo para seleccionar puntos en la gráfica, el cual consiste en considerar como elementos del vecindario a los puntos en la gráfica que sean vecinos de la solución actual; es decir, para una $f \in \mathcal{G}(n)$, el vecindario es el conjunto de funciones balanceadas $\mathcal{V}_f(n) = \{g \in \mathcal{G}(n) | f \oplus g \in \mathcal{G}(n)\}$.

La relación que mantienen los puntos en la gráfica nos da la opción de generar el vecindario sin tener que construir toda la gráfica, lo que representa un ahorro en memoria y tiempo de cómputo.

El cómo generar una función $g \in \mathcal{G}(n)$ tal que para cualquier $f \in \mathcal{G}(n)$ se cumpla que $f \oplus g \in \mathcal{G}(n)$ es un problema de combinatoria, dados dos funciones $f, g \in \mathcal{G}(n)$, $g \oplus f$ es balanceado si y solo si $card\{Spt(f) \cap Spt(g)\} = 2^{n-1}$ y $card\{Nul(f) \cap Nul(g)\} = 2^{n-1}$. Entonces si en primer lugar hacemos que g sea igual a f basta con invertir 2^{n-1} elementos en $Spt(g)$ y $Nul(g)$. Estos elementos pueden ser seleccionados de forma arbitraria. El algoritmo para hacer esto ésta expresado en 4.

Algoritmo 4 *Selección de Puntos sobre la Gráfica*

begin
n = número de variables;
f = función actual de n variables;
asignar g el valor de f
seleccionar $S \subset Spt(g)$ tal que $card(S) = 2^{n-2}$
seleccionar $N \subset Null(g)$ tal que $card(N) = 2^{n-2}$
intercambiar los valores de S
intercambiar los valores de N
end

Con este criterio de selección en cada iteración, el número de puntos se reduce a $\binom{2^n}{2^{n-1}}$ a $\binom{2^{n-1}}{2^{n-2}}^2$. Una característica importante de la gráfica $\mathcal{G}(n)$ es que el diámetro de ésta es igual a 2, es decir, que el camino más corto para llegar de un punto en la gráfica hacia cualquier otro punto es 2. En consecuencia, en cualquier iteración de los algoritmos de búsqueda es posible alcanzar un óptimo global a lo

n	no linealidad promedio	Búsqueda Local	Búsqueda Local en $\mathcal{G}(n)$
5	9	10	10
6	21	24	22
7	47	50	52
8	103	108	108
9	219	226	226
10	456	462	468
11	941	954	956
12	1926	1946	1948
13	3916	3940	3948
14	7927	7966	7974
15	15995	16034	16056
16	32199	32274	32288

Tabla 4.6: Resultados de la búsqueda local

sumo en dos pasos, por lo que no hay pérdida de generalidad. Utilizando el algoritmo 4 con los algoritmos de búsqueda local y recocido simulado se obtuvieron los resultados listados en las tablas 4.6 y 4.7 respectivamente.

Los resultados obtenidos al utilizar la gráfica $\mathcal{G}(n)$ mejoran los obtenidos con la selección aleatoria. Esto se debe a que la gráfica tiene una estructura más específica, pero estos resultados no mejoran significativamente, esto se debe a que la generación del vecindario sigue siendo aleatoria al momento de seleccionar los 2^{n-1} elementos. Tanto de la parte nula como del soporte de la función no se sigue un orden específico y no se puede garantizar una propiedad específica en los vecinos generados.

Si generamos aleatoriamente una función balanceada y medimos la no-linealidad de 1000000 de vecinos de la función generada, tendríamos los resultados mostrados en la tabla 4.8. Si graficamos el comportamiento de la no linealidad para diferentes nodos en $\mathcal{G}(5)$, $\mathcal{G}(6)$, $\mathcal{G}(7)$ y $\mathcal{G}(8)$ (ver figura 4.12), tenemos que existen nodos cuya media de no-linealidad es mayor que la de otros; es decir, existen nodos que tienen una mayor cantidad de óptimos globales.

Esto nos indica que para obtener mejores resultados es necesario desarrollar un mecanismo de selección de puntos sobre $\mathcal{G}(n)$ que guíe la búsqueda.

n	no linealidad promedio	Recocido Simulado	Recocido Simulado en $\mathcal{G}(n)$
5	9	10	12
6	21	24	24
7	47	52	54
8	103	108	110
9	219	226	230
10	456	466	470
11	941	956	960
12	1926	1946	1952
13	3916	3940	3952
14	7927	7966	7976
15	15995	16038	16068
16	32199	32268	32304

Tabla 4.7: Resultados del recocido simulado

$\mathcal{G}(5)$

Función 0×16	nl Promedio
9c0e3ecc	8.543510
45aeb60b	8.243972
639abe5	9.119946
9f4a14b5	8.567566
c51544fd	7.731670
3238eb47	7.729218
4b993137	8.670640
62d665ca	8.569256
559e3ce	8.957816
495fc8aa	8.987646
657905b6	8.987000
293325f5	8.666398
79d22c0f	8.662380
598a6b63	8.250646
cc67bc11	8.565746

$\mathcal{G}(6)$

Función 0×16	nl Promedio
e6bb4706ac2964cb	20.466384
ddc5db8991c78614	21.169384
bf1860924f65bc9c	20.780298
44a9f954f232d8a7	20.591854
9b3685f24b07ec1a	20.978996
a27ce8d86ee1941b	20.651110
4c02c1d532d6777d	20.013056
9ad4759b99a1c694	20.012484
c0f0713befd43270	19.187056
6f48c3bbd7121694	20.906324
b9027ac4d631d667	19.110020
62b38badb8174b16	21.066684
6eec338d4b31c893	15.966658
157ca57b8c58856b	15.957108
685314fd7a26b589	19.555118

Tabla 4.8: No-linealidad promedio en $\mathcal{G}(5)$ y $\mathcal{G}(6)$

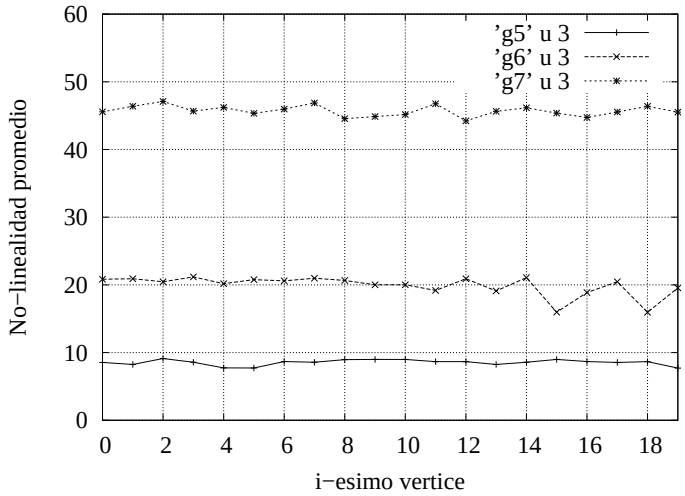


Figura 4.12: No-linealidad promedio en nodos de $\mathcal{G}(n)$

Capítulo 5

Heurística

Para mejorar los resultados obtenidos con los algoritmos de búsqueda de la sección anterior, es necesario un mecanismo para seleccionar puntos; dicho mecanismo debe tener la característica de ser simple y utilizar la gráfica $\mathcal{G}(n)$ para generar funciones balanceada que mejoren los resultados obtenidos.

El mecanismo que se desarrolló para mejorar el mecanismo de selección utilizado anteriormente, se basa en una regla que determina qué bits en una función balanceada deben ser alterados para incrementar la no-linealidad.

Una regla de selección ideal sería capaz de determinar los bits que deben ser modificados para alcanzar un óptimo global desde cualquier vértice de la gráfica en dos pasos, pero como se verá más adelante esto resulta muy difícil (o imposible) de realizar.

Como se mencionó al inicio, la regla que se desarrolló utiliza la gráfica $\mathcal{G}(n)$ por ser una estructura precisa para representar el espacio de funciones balanceadas; pero, además, ésta se basa en el hecho de que el diámetro de $\mathcal{G}(n)$ es igual 2 para determinar el comportamiento de la no-linealidad en funciones balanceadas de n variables.

A grandes rasgos, la regla se divide en dos pasos. El primero constan en realizar un proceso de precómputo con el que se estimará el comportamiento de la no-linealidad sobre los puntos de la gráfica. El segundo paso utiliza el comportamiento de la no-linealidad previamente calculado para proponer los bits que deberán permanecer con valor 0, 1, y cuáles deberán ser modificados para aumentar la no-linealidad de una función dada.

De los dos procesos en que se encuentra dividida la regla de selección, el precómputo es el más difícil de resolver ya que, el determinar el comportamiento de la no-linealidad involucra varios procesos. La solución que se propone es

Figura 5.1: Representación en patrones de la función f : 10101010.

visualizar el comportamiento de la no-linealidad mediante una representación de las funciones basada en patrones, la cual se define de la siguiente manera: para cualquier $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$, f está asociada a una cadena de celdas (blancas, negras) de longitud 2^n, tal que, si $f(\boldsymbol{\delta}) = 1$, entonces la celda indexada por $\boldsymbol{\delta}$ será de color negro y en consecuencia si $f(\boldsymbol{\delta}) = 0$ la celda será de color blanca.

Para clarificar esta idea, se muestra el siguiente ejemplo para una instancia de $n = 3$, si

$$f = \begin{pmatrix} 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

entonces su representación con patrones se muestra en la figura 5.1.

Debido a que el número de bits en una función booleana crece exponencialmente, el tamaño de la imagen que se genera también lo hará y se tendrán los mismos problemas que se tienen con las tablas o vectores de verdad. Un ejemplo más completo de la representación en patrones se presenta en la la figura 5.2 la cual muestra el espacio de funciones booleanas balanceadas con 3 y 4 variables.

Es claro que no es posible graficar todo el espacio de funciones para $n > 4$ en una hoja de papel, pero en una computadora sí es posible observarlo completamente al dividir a dicho espacio en intervalos pequeños. En la practica no es de gran utilidad el observar un espacio de funciones tan grande ya que sólo estamos interesados en las funciones con máxima no-linealidad.

Si graficamos sólo las funciones balanceadas con máxima no-linealidad, la gráfica para $n = 2$ sería igual a la figura 5.2, ya que todas las funciones de 2 variables tiene no-linealidad igual a 0. Por otra parte, en la figura 5.3 se muestran las funciones de 3 variables con máxima no-linealidad. A través de esta figura es posible observar algunas propiedades de las funciones booleanas. Por ejemplo, como se mencionó en el capítulo 2, para cualquier función $f \in \mathbb{F}_2^{\mathbb{F}_2^n}$ con $nl(f)$ se cumple que $nl(f) = nl(\overline{f})$, donde $\overline{f}$ es el complemento de f; esto se muestra en la figura 5.3.

Aunque el espacio de las funciones balanceadas con máxima no-linealidad es menor al de las funciones balanceadas, se sabe que para $n = 4$ el número de funciones con máxima no-linealidad es 10920 ver tabla 4.5 página 43 lo que com-

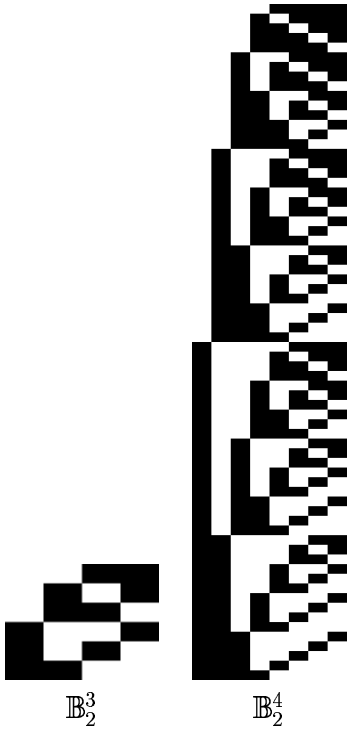


Figura 5.2: Representación en patrones de $\mathbb{B}_2^3$ y $\mathbb{B}_2^4$.

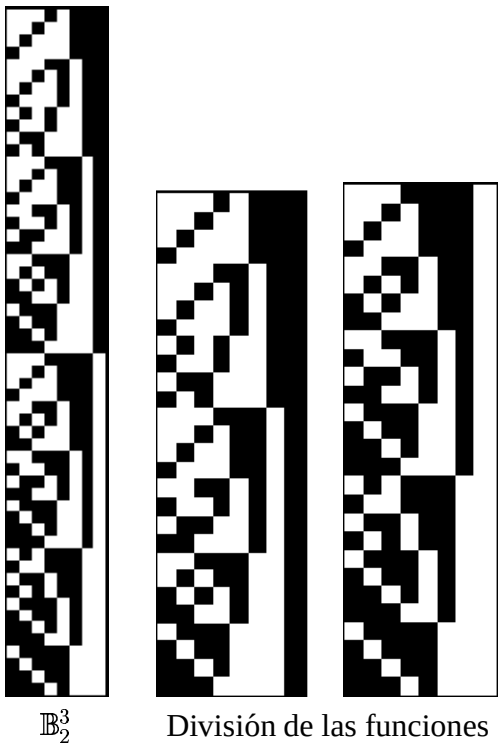


Figura 5.3: Representación en patrones de $\mathbb{B}_2^3$ con máxima nl.

plica la visualización de la representación en patrones de dicho conjunto. Aunado a los problemas de visualización, se tiene el problema de calcular el conjunto de funciones booleanas balanceadas con máxima no linealidad, ya que, para valores de $n > 7$ ni siquiera es posible calcular dicho conjunto.

Estos factores incrementan el problema de desarrollar un método para determinar el comportamiento de la no-linealidad, por lo que es necesario reducir el número de funciones que deben ser utilizadas en este método.

Para reducir el número de funciones se utilizó la gráfica $\mathcal{G}(n)$ por las propiedades que ésta posee. Específicamente, la propiedad del diámetro y porque existe más de un camino que parte de f y llega a g. Del conjunto de funciones seleccionaremos aquellas que aporten información relevante para encontrar un patrón en la no-linealidad; dicha selección utiliza como punto de partida una función balanceada f con no-linealidad parcial y una función g con no-linealidad máxima, tal que, $nl(f) \leq nl(g) - 2$; entonces se consideraran las funciones $h \in \mathcal{G}(n)$ que satisfacen la condición de $f \oplus h \in \mathcal{G}(n)$, $g \oplus h \in \mathcal{G}(n)$ y con $nl(f) \leq nl(h) \leq nl(g)$, es decir, se seleccionan únicamente las funciones que aumenten la no linealidad de una función balanceada. Esta regla queda delineada en el algoritmo 5.

Algoritmo 5 Criterio de selección

Seleccionar $f_1, f_2 \in \mathcal{G}(n)$ tales que $f_1 \oplus f_2 \notin \mathcal{G}(n)$ y $nl(f_1) <= nl(f_2) - 2$
Calcular $\mathcal{G}'(n) \subset \mathcal{G}(n)$, tal que $\forall g \in \mathcal{G}'(n)$ se cumple

- $f_1 \oplus g \in \mathcal{G}(n)$ y $f_2 \oplus g \in \mathcal{G}(n)$
- $nl(f_1) < nl(g) < nl(f_2)$

Determinar el comportamiento de la no linealidad en $\mathcal{G}'(n)$

El generar una función $h \in \mathcal{G}(n) | f \oplus h \in \mathcal{G}(n)$ y $g \oplus h \in \mathcal{G}(n)$ es un problema de combinatoria, el cual consta de variar un número suficiente de bits en una función para generar funciones balanceadas. El algoritmo 6, cuya entrada son las funciones $f, g \in \mathcal{G}(n)$ con $f \oplus g \notin \mathcal{G}(n)$, específica el número de bits y qué bits deben ser modificados para generar una función balanceada $f \oplus h \in \mathcal{G}(n)$ y $g \oplus h \in \mathcal{G}(n)$

Para generar todos los vecinos, basta con calcular todas la combinaciones de $\frac{r}{2}$ elementos en $Null(h)$ o $Spt(h)$ según sea el caso; del conjunto resultante se seleccionan sólo aquellas funciones cuya no-linealidad sea mayor $nl(f)$ y menor a $nl(g)$.

Algoritmo 6 *Generación de vecino*

begin
n = número de variables;
$f, g \in \mathcal{G}(n)$ $//f \oplus g \notin \mathcal{G}(n)$
asignar $h = f \oplus g$
$Spt(h) = (Spt(f) \cap Null(g)) \cup (Null(f) \cap Spt(g))$
$Null(h) = (Spt(f) \cap Spt(g)) \cup (Null(f) \cap Null(g))$
if $card(Null(h)) > card(Spt(h))$ **then**
  $r = card(Null(h)) - card(Spt(h))$ // Dado que $f, g \in \mathcal{G}(n)$, r es par
  Alterar $\frac{r}{2}$ elementos de $Null(h)$
else
  Alterar $\frac{r}{2}$ elementos de $Spt(h)$
end if
$f \oplus h$ es balanceado y $g \oplus h$
end

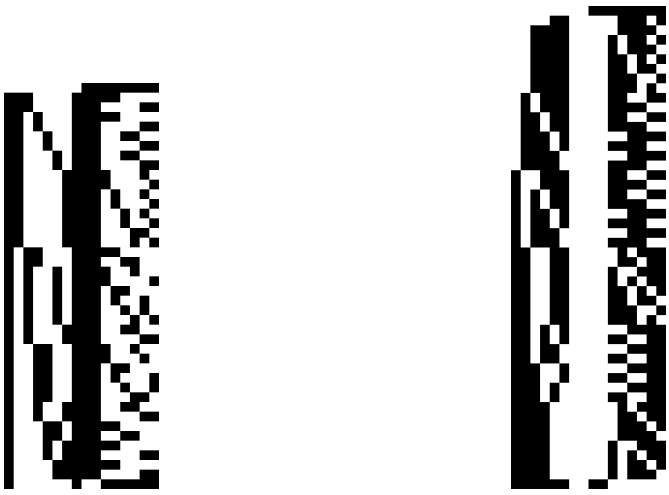
Esto disminuye considerablemente el número de funciones que pueden ser consideradas para determinar el comportamiento de la no-linealidad, aunque no es posible determinar con exactitud el número de funciones con no-linealidad intermedia para dos funciones; la única forma de saber cuantas funciones existen es mediante los algoritmos descritos.

En la figura 5.4 se muestran las funciones que cumple con la regla anterior para las funciones balanceadas de 4 variables $f = 0 \times FF$ con $g_1 = 00 \times 813F$ y $g_2 = 00 \times FCC0$.

Lo que si se sabe es que, el número de funciones que cumplen con esta condición es diferente para cada par de funciones f y g, como se muestra en la figura 5.4.

Hasta este momento sólo se ha propuesto un método para generar el conjunto de funciones que serán utilizadas para determinar el comportamiento de la no-linealidad, pero aun no se ha descrito el procedimiento a seguir.

El localizar patrones en un diagrama es un problema complicado, y existen diferentes formas de resolverlo. La regla que se desarrolló propone tres patrones. El primero es el de los bits que deben tener valor 0, el segundo es el de los bits que deben tener valor 1 y el tercero el de los bits que deben variar (bits con valor indeterminado) para aumentar la no-linealidad; para determinar esto no basaremos en la frecuencia con que los bits de la i-ésima columna de la representación en patrones de una pareja de funciones aparecen con valor 0 o 1, lo que significa



(a) $f = 0 \times FF$ con $g_1 = 0 \times 813F$ (b) $f = 0 \times FF$ con $g_2 = 0 \times FCC0$

Figura 5.4: Representación en patrones de vecinos intermedios.

agrupar los bits en tres cúmulos.

5.1. Teoría de Cúmulos

El objetivo de la teoría de cúmulos es el organizar objetos en grupos llamados cúmulos (clusters), de tal manera que los individuos de un mismo cúmulo tengan propiedades semejantes con respecto a las variables medidas.

En el pasado, la teoría de cúmulos ha sido utilizada en un variedad de áreas tales como reconocimiento de patrones, recuperación de información, análisis microbiológico, etc. [18]

Los algoritmos de cúmulos pueden ser clasificados como:

- Exclusivos : Los datos son agrupados en una forma exclusiva, si un cierto objeto pertenece a un cúmulo definido entonces, éste no puede ser incluido en otro cúmulo.
- Overlapping : Estos algoritmos utilizan conjuntos difusos para clasificar los objetos, así cada objeto puede pertenecer a 2 o más cúmulos con diferentes grados de pertenencia.
- Jerárquicos : Estos algoritmos se basan en la unión de los dos cúmulos más cercanos.

- Probabilísticos : Este utiliza un enfoque completamente probabilístico.

Existen diferentes algoritmos de la teoría de cúmulos:

- K-means
- Fuzzy C-means
- Hierarchical clustering
- Mixture of Gaussians

De estos algoritmos, el que se ajusta al problema que se tiene es el de *k-means*; éste fue propuesto por J. B. MacQueen en el año de 1967[19], y es uno de los algoritmos de aprendizaje más simple que resuelve el problema de acumulación.

El algoritmo de k-means sigue un método simple para clasificar un conjunto de objetos en un cierto número de cúmulos (asumiendo que k es el número cúmulos) fijados a priori. La idea es definir k puntos como centro de cada uno de los cúmulos. Estos puntos deben ser seleccionados cuidadosamente. El siguiente paso es tomar cada objeto y asociarlo al cúmulo cuyo centro sea el más cercano. Cuando todos los puntos han sido asociados, el primer paso se ha completado y un primer agrupamiento se ha hecho. En este punto se calculan k nuevos centros para cada uno de los cúmulos resultantes del paso anterior. Ahora se tiene k nuevos centros y una nueva unión debe ser hecha entre los objetos y los nuevos centros. Esto se realiza en un ciclo. Como resultado de este ciclo se puede notar que, los k centros cambian su posición en cada paso, y el ciclo termina hasta que no hayan más cambios. El algoritmo de $k - means$ queda delineado por el algoritmo 7

Algoritmo 7 *k-means*

begin
Sea (**k**) número de cúmulos;
Seleccionar **k** puntos como *centros*.
repeat
  Asignar cada punto al centro más cercano.
  Calcular el nuevo centro del cúmulo.
until Centros no se modifiquen.
end

La selección de los centros de cada cúmulo debe de hacerse cuidadosamente, ya que centros diferentes producen resultados diferentes como se muestra en la figura 5.5

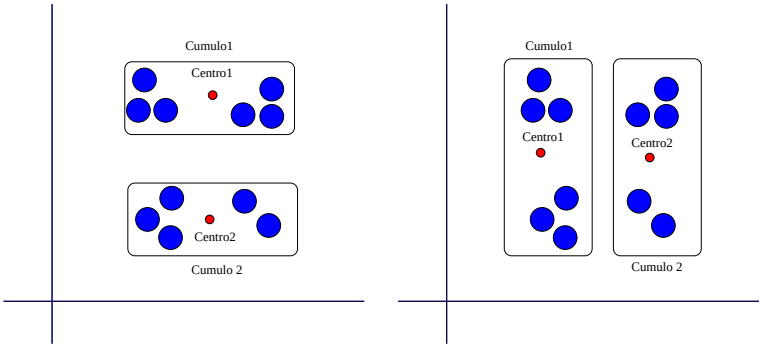


Figura 5.5: Algoritmo de k-means

Uno de los problemas que tiene el algoritmo de k-means, es el presuponer un número k de cúmulos, lo que puede ser una desventaja en la solución de algunos problemas; pero, en este caso, esta desventaja resulta en una característica ideal debido a que nosotros conocemos el número de cúmulos que se desean generar ($k = 3$).

La distancia entre dos puntos puede calcularse con diferentes métricas como son: la distancia euclidiana, distancia Manhattan, distancia de Minkowski de grado m, distancia cuadrática, coeficiente de correlación lineal. Sin embargo, el aplicar el algoritmo de k-means implica modificar la forma de medir la distancia entre puntos; ya que, cada punto en este caso tiene una frecuencia de bits 0 y una frecuencia de bits 1, y éstos no pueden ser utilizados como coordenadas x y y. En consecuencia, no es posible fijar un centro para cada cúmulo con coordenadas (x, y). Estos cambios consistieron en modificar el concepto de centro de un cúmulo y el de la distancia de un elemento al centro de un cúmulo.

En primer lugar definiremos qué es un punto al que denotaremos por p_i, como la pareja $s, n \in \mathbb{N}$, tal que n es igual al número de funciones que en la i-ésima entrada tienen valor 0, y s es igual al número de funciones que en la i-ésima entrada tienen valor 1.

El centro del cúmulo 0 es igual al $p_k =< n_k, s_k >$, tal que, para cualquier $p_i =< n_i, s_i >$ del conjunto de funciones intermedia, $n_k > n_i$ y el centro del cúmulo 1 es igual al $p_l =< n_l, s_l >$, tal que, para cualquier $p_i =< n_i, s_i >$ del conjunto de funciones intermedia, $s_l > s_i$.

Para medir la distancia de un punto hacia el centro de un cúmulo, se utiliza una constante $t \in \mathbb{N}$, de la siguiente forma, sean $p_C =< n_C, s_C >$ el centro del cúmulo 0, $p_U =< n_U, s_U >$ el centro del cúmulo 1 y $p_i =< n_i, s_i >$ un punto, si $|n_C - n_i| \leq t$ entonces el centro más cercano al punto p_i es el centro

del cúmulo 0, si $|s_U - s_i| \leq t$; entonces el centro más cercano al punto p_i es el cúmulo 1 y en otro caso diremos que el centro del cúmulo $1/0$ es el más cercano al punto p_i, donde el cúmulo $0/1$ es el cúmulo de los bits que deben variar. El equivalente de recalcular el centro de cada cúmulo es decrementar los valores de $p_C =< n_C, s_C >$ y $p_U =< n_U, s_U >$, y el criterio de paro consiste en que los cúmulos no se modifiquen. Esto queda delineado en el algoritmo 8

Algoritmo 8 *k-means modificado*

begin
Sea **k**=3 número de cúmulos;
Sea **t** la constante de precisión;
Sea k_0 el cúmulo de los puntos 0
Sea k_1 el cúmulo de los puntos 1
Sea $k_{1/0}$ el cúmulo de los puntos 1/0
repeat
  Asignar cada punto al centro más cercano.
  Decrementar los valores de k_0 y k_1.
until Los cúmulos no se modifiquen.
end

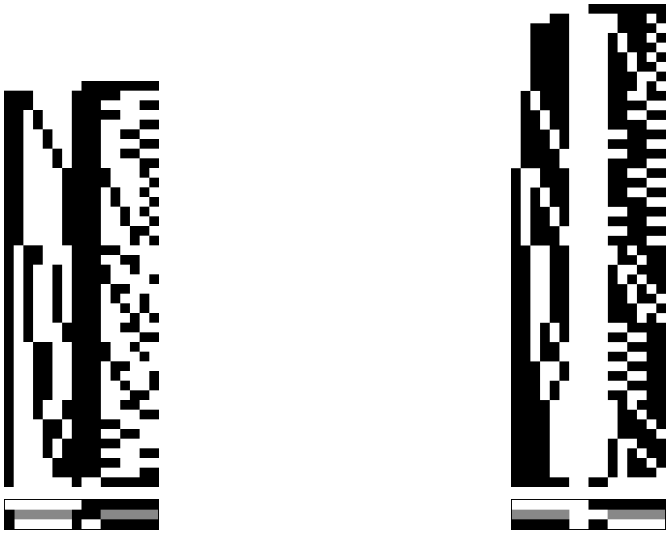
Se mencionó que hay un tercer cúmulo, éste es el de los bits cuyo valor debe cambiar, siguiendo la notación de la representación basada en patrones, tenemos que para la función $f(\boldsymbol{\delta}) = 1/0$ indexada por $\boldsymbol{\delta}$ esta será coloreada de color gris.

En la figura 5.6 se muestran los resultados de ejecutar el algoritmo de k-means modificado con las funciones balanceadas de 4 variables $f = 0 \times FF$ con $g_1 = 0 \times 813F$ y $g_2 = 0 \times FCC0$.

El algoritmo de k-means original tiene la característica de generar resultados diferentes cuando los valores de los centros iniciales difieren. En el caso del algoritmo de k-means modificado, para valores diferentes de la constante de precisión **t**, este arrojará resultados diferentes como se muestra en la figura 5.7.

El principal problema de calcular patrones en funciones balanceadas de n variables es el crecimiento exponencial del número de bits y del tiempo necesario de calcular la no-linealidad; por lo que se debe tener un mecanismo para extrapolar patrones de grado n a patrones de grado $n + 1$ en un tiempo menor.

Un método sencillo para extrapolar este criterio a valores de n más grandes sin que se ejecute el algoritmo 8, es tomar como entrada una patrón previamente calculado y completar el número de bits faltantes con valores indeterminados. Por ejemplo, si tomamos el patrón resultante de las funciones de 6 variables $0 \times$



(a) $f = 0 \times FF$ con $g_1 = 0 \times 813F$ (b) $f = 0 \times FF$ con $g_2 = 0 \times FCC0$

Figura 5.6: Representación en patrones de vecinos intermedios.

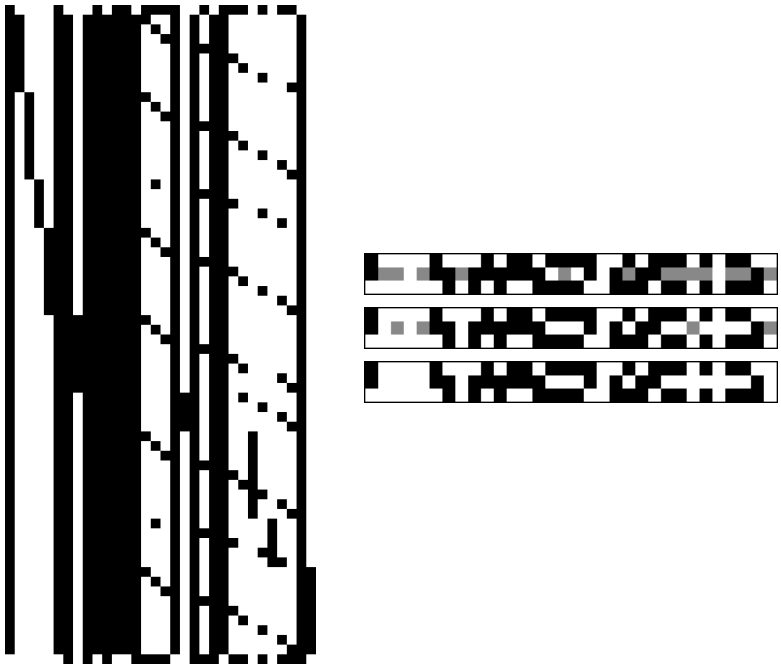


Figura 5.7: Patrones de $0 \times 845BCBAC$ $0 \times 2a79DAE$.

Figura 5.8: Extrapolación de patrones de 7 a 8 variable.

$845BCBAC\,0\times 2A79DAE$ de la figura y lo extrapolamos a 7 variables, tenemos el patrón de la figura 5.8

Este método de extrapolación es muy rápido. Desafortunadamente al extrapolar un patrón para funciones de n variables a $n+1$ se indeterminan 2^n bits por lo que la efectividad del patrón disminuye.

5.2. Selección de Puntos

Una vez que se tiene un patrón del comportamiento de la no-linealidad entonces el siguiente paso es generar funciones balanceadas a partir de éste. Para una función $f \in \mathcal{G}(n)$ y un patrón p, se generará una nueva función $g \in \mathcal{G}(n)$ tal que

$$g(x) = \begin{cases} 0 & \text{Si } f(x) = 0 \text{ y } p_w = 0 \\ 1 & \text{Si } f(x) = 0 \text{ y } p_w = 0 \\ \text{-1} & \text{en otro caso} \end{cases}$$

Esta regla genera una función que no es balanceada y el número de bits indeterminados se incrementa, Este proceso queda descrito en el algoritmo 9, donde p_w es el valor del patrón p en la $w-$ésima posición y si $p_w = -1$ indica que el valor de p en la w posición es indeterminado; el resultado de este proceso es una función con valores indeterminados. Para obtener una función balanceada se seleccionarán $2^{n-1} - Spt(g)$ bits indeterminados y se les asigna el valor 1 y al resto de los valores indeterminados se les asigna el valor 0. El resultado de este proceso es una función balanceada. El algoritmo se muestra en 9. Un ejemplo de la ejecución del algoritmo se muestra en la figura 5.9.

Dos elementos en este algoritmo que pueden resultar algo incómodos son el incrementar el número de bits indeterminados, ya que si éstos incrementan desmesuradamente la efectividad puede disminuir, pero si no se varían una cantidad suficiente entonces tal vez no sea posible escapar de óptimos locales. Por otra parte está la selección aleatoria de los bits que deben ser utilizados para complementar el soporte y la parte de nula de la función que está siendo generada. Sin embargo, en este punto no es posible determinar si la selección aleatoria tiene un efecto negativo con respecto a la optimización de la no-linealidad.

Algoritmo 9 *Selección de Puntos*

begin
sea $G :=$ patrón producido por k-means (k=3).
sea $g := 0$; $\setminus\setminus$ g es el vecino producido
sea $s := 0$; $\setminus\setminus$ s el contador de vls. 1
sea $H :=$ lista vacía ; $\setminus\setminus$ H es una lista de vls. 0/1
for $\delta \in \mathbb{F}_2^2$ **do**
  if $f(\delta) == 1$ y $G(\delta) ==$ negro **then**
    sea $g(\delta) := 1$; $s++$;
  end if
  if $f(\delta) == 0$ y $G(\delta) ==$ blanco **then**
    sea $g(\delta) := 0$;
  else
    agregar δ a la lista H;
  end if
end for
para $2^{n-1} - s$ elementos $\delta \in H$ sea $g(\delta) = 1$;
función de salida g
end

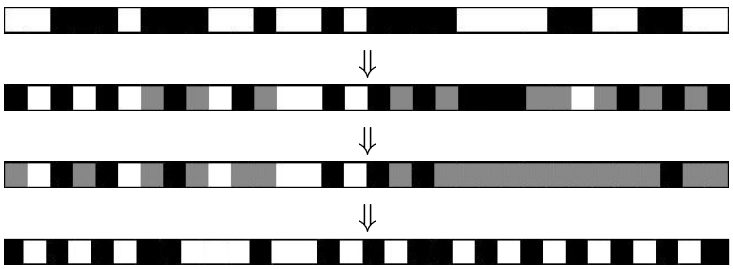


Figura 5.9: Selección de vecinos

Número de variables	$\mathcal{G}(n)$	Heurística
5	12	12
6	24	24
7	54	54
8	110	110
9	230	230
10	472	470
11	960	960
12	1952	1952
13	3952	3954
14	7974	7974
15	16068	16068
16	32288	32292

Tabla 5.1: Resultados preliminares

5.3. Resultados

Sustituyendo la selección de puntos en la gráfica por el algoritmo 9 en el algoritmo de recocido simulado se tienen los resultados reportados en la tabla 5.1

Los resultados obtenidos con la heurística son básicamente los mismos que se obtuvieron con la selección de puntos en la gráfica $\mathcal{G}(n)$. Aunque se tenga un proceso de selección, el número de funciones balanceadas sigue siendo demasiado grande y el número de óptimos globales es reducido.

Aunque la selección de puntos utilizando el patrón no mejora los resultados obtenidos anteriormente, éste presenta una mejora en experimentos realizados cuando se reduce el número de elementos en el vecindario y el número de iteraciones con el algoritmo de recocido simulado.

Las tablas 5.2, 5.3 y 5.4 muestran los resultados de ejecutar la heurística y el recocido simulado con diferentes parámetros. Estos resultados muestran el desempeño de la heurística.

Los resultados muestran que al disminuir el número de elementos en el vecindario el algoritmos de recocido simulado disminuye su eficiencia mientras que la heurística muestra un desempeño más estable. Extendiendo los experimentos para $n = 7$ y cuyos resultados se muestra en la tabla 5.5 se puede observar el mismo comportamiento.

Con base en los resultados obtenidos experimentalmente al reducir la cardina-

$\mathcal{G}(n)$	
f_{16}	no linealidad
cbd188033fee742c	18
d2d9a8033fea64b4	24
d35b28033feb2534	20
c95d20033ffb456c	20
cb5120033ffb752c	22
da5518033fe755a4	22
c95d88033fee456c	22
d1dbb8033fe22474	20
c357a0033ffa153c	18
d957b0033ff21564	18
d953a0033ffa3564	22
d951a0033ffa7564	20
c0d910033ff764fc	20
d0d3a8033fea34f4	22
d9d9b8033fe26464	16
d8d730033ff314e4	20
d85fa0033ffa05e4	16
c253a0033ffa35bc	18
d05988033fee65f4	22
d15b20033ffb2574	20

Heurística	
f_{16}	no linealidad
42771972abe869d8	24
b619875786b36705	24
110eda1a6fc067dd	22
898129edeb752e8c	24
841dac3a72afb591	22
4a6f1fb8c50991ae	24
eb1188a7d6f80dd4	24
dcf24cc4fd60941b	24
1e203c8e8dea6ef2	24
795373883124797e	24
62a3458f10b8aefb	24
f0ab45c9857e4595	22
6538e515a6fd8970	24
92b91af95c63319a	24
e2b0b84ff2693439	24
498c5855bdb7d42c	24
42cf49fa718b614e	24
22ba85832f3d14f7	24
82b3b437c61475f8	24
ac6aae2191dc2e79	24

Tabla 5.2: Número de variables 6; Número de vecinos: 223; Número de ciclos: 4

$\mathcal{G}(n)$	
f_{16}	no linealidad
1845c271c1b2f7d7	22
36b2738d6f871324	24
61c38b7b0727954e	24
62f24209dbfaac5c	24
1e19e02cd362e5de	22
cefc74889cb2f680	24
97a3ab850cc3fc34	24
8322bcb75a9f2135	22
21f1683994ad3d6e	24
a037cf4ee17683d0	24
98d6d018853fdc7c	26
1425b0adf9d888f7	24
6f401aa77078377c	24
d7f87e4445d9e880	24
e17fb1e049447c2e	24
a4f6e33bae890ad0	24
19d04ce66a713d67	24
24ad300f05dddbe9	22
fd16d9cd38514b24	24
5ac17417c9e3454f	24

Heurística	
f_{16}	no linealidad
d15d98033fe64574	20
c153b0033ff2357c	20
cb5d88033fee452c	20
c2db38033fe324bc	20
d25910033ff765b4	22
d8d1a0033ffa74e4	24
c95bb0033ff2256c	20
c15ba8033fea257c	22
ca5d10033ff745ac	20
c8dbb8033fe224ec	22
d1dd08033fef4474	18
d8dda0033ffa44e4	20
cbdb18033fe7242c	18
ca5f90033ff605ac	20
d0dd38033fe344f4	20
c2db10033ff724bc	20
d2dbb8033fe224b4	20
db5138033fe37524	20
d3dd90033ff64434	18
c05128033feb75fc	18

Tabla 5.3: Número de variables : 6; Número de vecinos: 213; Número de ciclos: 3

$\mathcal{G}(n)$	
f_{16}	no linealidad
eb185bf694cd8215	22
c287a3d762fae25	20
e6a573717920da98	22
9bbc745d20a4aba3	22
ca6fc32158d86bf	24
4b9533d59a2c5678	20
a24fd4742720abe7	24
c55c64e9eba1983c	24
b6d05c34d4b115d7	22
c762f86182e63b53	24
e73e7c5500f2924d	22
658e41dbaea8726c	22
625fe4bea87f2104	24
68e69407aeb285af	24
9762ea5b90ad159c	24
28ded424a1b789cf	24
147b89879477e4ac	22
9fbcdf1c1425240f	22
b41f41f64b18c47e	22
cf1ea18e652a1676	24

Heurística	
f_{16}	no linealidad
dad7a0033ffa14a4	18
db5188033fee7524	22
c8db90033ff624ec	22
c9d198033fe6746c	18
d15318033fe73574	24
c8d5a0033ffa54ec	20
c85118033fe775ec	20
ca5f98033fe605ac	22
d35b18033fe72534	20
d95d98033fe64564	18
d0d900033fff64f4	20
c15f18033fe7057c	18
c05d10033ff745fc	20
d253a8033fea35b4	22
c1d308033fef347c	18
d9d3a0033ffa3464	20
cb5330033ff3352c	22
c0d798033fe614fc	22
d85718033fe715e4	22
db5328033feb3524	20

Tabla 5.4: Número de variables : 6; Número de vecinos: 213; Número de ciclos: 1

$\mathcal{G}(n)$	
f_{16}	no linealidad
3b9b93f192ff052d4b5f00b670362623	44
72a8b1f3a26e1f31730789ba3072eab1	46
3b82b5f382678762b91e19be3052be23	48
7ab211a1a27f9534d35601ba7a77b2a1	46
7a90dff1b2f29d72b146b0b27004f6a1	42
3ab99fe3b273df3cc30431b2380662a3	44
7ba83fa3b2e7853e835e18b23a03ea21	42
729ad5fb9267c7217b1c19b62054a6b1	46
73905df982e78737131e18be6045f631	50
32b017f9a27f5771711501ba6017f2b3	42
72819de9a2628760f91eb9ba68467eb1	44
339037e992fbcf7d410c20b66813f633	44
3a91f7f182e3cd3aa34c38be701076a3	46
72a897ab82ea0d217b4fa8be2a16eab1	46
73baddf3a2724730f31db1ba3044a231	46
7aa879e1b26e8f64d90e89b27861eaa1	48
3a9997b3b2778520fb5e11b2321666a3	44
3b8377e18263d779611439be78113e23	46
7a9a99a3a2f2df7aa104b0ba3a66a6a1	42
7bb21ba9a2674779611d19ba6a27b221	46

Heurística	
f_{16}	no linealidad
c17c6eface8063e824ce15391338bf78	48
a8734cd8ccbfb2a87652532443f376a2	50
db60fceca3c38bf49070736d2d708634	52
b9626ceab2cdeb43455c9fb201781f14	48
70e85ce49c8eabf43ac707ea0934b770	50
70690cd01adf22c28bd165789bb2e77b	50
9364efd3c68f43029565cd36217596f0	50
71706fa83adaab04a4c669b40738d7f3	50
b4e81ca00eb19a90aace653d3ff2d794	50
3ee41eaf798b16b8b05843aa7935d650	50
7768bcf6239a0688c2f14f90c1b366f6	50
f1603fc08a8987e1e44ce322e1f1b7b7	50
3de01dec3dcda7042a6665620b3d7716	52
e1782f8e48a98e7132d401f065f75f3c	48
33651cfe11cb37432554876065ff960e	52
f3648eec40cb3b656155f3a8b1353e21	50
50f41eee18f9d64ae4dc4d287b703670	50
1e8dde541afa668356091cd27f3f6c8	50
d6751ec42ab943811ad5b56f4332b770	52
f87e1fe8e2e3a25964c40f120bb9867c	50

Tabla 5.5: Número de variables : 7; Número de vecinos: 231; Número de ciclos : 5

$$f(n) = 1.30381\ n^3 - 14.8007\ n^2 + 50.8046\ n - 44.2. \tag{5.1}$$

$$k = 0.550137\ n^2 - 3.81909\ n + 7.4011. \tag{5.2}$$

Tabla 5.6: Parámetros iniciales

$$f(n) = 2.87229\ n^2 + 50.7208\ n - 264.03. \tag{5.3}$$

$$k = -0.0292208\ n^2 + 6.16775\ n + 27.7727. \tag{5.4}$$

Tabla 5.7: Nuevos parámetros

lidad del vecindario y el número de iteraciones, se estimó que los nuevos parámetros para el recocido simulado con la heurística son los que se muestran en la tabla 5.7, donde $f(n)$ representa la cardinalidad del vecindario y k el número de iteraciones.

Al reducir el número de iteraciones y la cardinalidad del vecindario es obvio que el tiempo de ejecución se reduzca. Pruebas experimentales midiendo el tiempo promedio del algoritmo de recocido simulado con y sin la heurística son graficadas en la figura 5.10.

Indudablemente con el algoritmo de selección es posible reducir el tiempo de cómputo para localizar funciones booleanas balanceadas con alta no-linealidad.

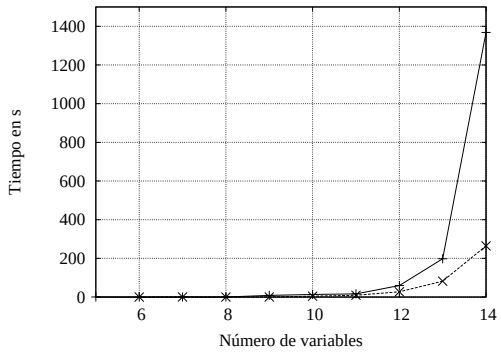


Figura 5.10: Selección de vecinos

Capítulo 6

Conclusiones

El principal problema de optimizar una o más propiedades criptográficas de las funciones (sean balanceadas o no) es la cardinalidad del espacio de funciones booleanas, el cual crece de manera doblemente exponencial. Dado que la cantidad de funciones con propiedades importantes es reducido, la probabilidad de localizar funciones con propiedades criptográficas importantes disminuye cuando el número de variables de la función booleana crece.

Aunado al problema de la cardinalidad, está el problema de representar una función. Existen diferentes formas de expresar funciones booleanas como es la *Forma Normal Algebraica* (FNA). Sabemos que toda f posee una única forma reducida FNA y resulta muy práctica esta representación. Pero no es posible utilizar la FNA en algunos métodos de búsqueda como el de búsqueda local y recocido simulado que se desarrolló, por lo que es necesario utilizar representaciones como la de vector de verdad. El utilizar esta representación implica más problemas, tales como el espacio necesario para representarlo ya sea a través de arreglos de caracteres o si se utiliza una representación numérica. Esto implica utilizar un conjunto de funciones (como las de GMP) que permitan expresar números grandes de forma eficiente o funciones que utilicen otros medios de almacenamiento para poder manejar cadenas de bits grandes cuyo crecimiento se comporta de manera exponencialmente respecto al número de variables.

La gráfica de funciones balanceadas $\mathcal{G}(n)$ provee una estructura específica y fácil de construir. Entre sus principales características es el crecimiento exponencial que tiene el conjunto de vértices y el número de vecinos que tiene cada nodo, pero la propiedad más importante que se encontró fue que el diámetro de la gráfi-

ca es igual a 2 y en procesos de búsqueda esto resulta muy atractivo, ya que es posible alcanzar la solución óptima desde cualquier vértice en la gráfica.

Esta propiedad puede ser explotada para generar un mecanismo de selección de funciones booleanas con ciertas características de forma precisa, reduciendo el número de funciones que deben ser consideradas sin generar toda la gráfica y en consecuencia el tiempo de cómputo para calcularlas.

El espacio de funciones booleanas es también un álgebra de Boole, por lo que la suma $\oplus$ es de orden 2, es decir, cada elemento es su propio inverso. Esto conlleva que $\mathcal{G}(n)$ posea, entre otras propiedades, un diámetro igual a 2, y además sea cerrada bajo la operación de complemento. La gráfica $\mathcal{G}(n)$ permite así reducir espacios de almacenamiento y realizar búsquedas eficientes sobre ella.

Entre los diferentes métodos para localizar funciones con propiedades criptográficas importantes, los métodos heurísticos son una buena opción ya que éstos tienen características atractivas como sencillez y su capacidad de generar una gran variedad de funciones booleanas.

Los algoritmos de búsqueda local y recocido simulado arrojan resultados aceptables cuando el espacio de búsqueda tiene una cardinalidad que no sea muy grande. Los resultados obtenidos con estos algoritmos son aceptables para valores de n pequeños, pero cuando el valor de n aumenta la cardinalidad del espacio de búsqueda crece exponencialmente y la probabilidad de localizar óptimos globales se reduce. Es por esta razón que el contar con una estructura de vecindario precisa es necesaria para mejorar los resultados obtenidos. Sin embargo, los resultados obtenidos muestran que el utilizar la gráfica como vecindario no es suficiente y el contar con un método que guíe la búsqueda se vuelve necesario para mejorar los resultados de los algoritmos.

Los resultados del recocido simulado están relacionados con los parámetros del sistema: la temperatura inicial, la constante de enfriamiento, criterio de enfriamiento y criterio de paro. Sin embargo, diferentes valores de estos parámetros generan resultados diferentes, lo que hace difícil proponer parámetros ideales para obtener un mejor desempeño del algoritmo de recocido simulado dado que no existe un método exacto para calcular estos parámetros.

Los resultado obtenidos con el algoritmo de recocido simulado en la localiza-

ción de funciones booleanas balanceadas escapan a los óptimos obtenidos con la selección aleatoria y búsqueda local, pero la cardinalidad del espacio de búsqueda local afectan el desempeño de éstos y aún cuando se utilizó la gráfica de funciones balanceadas para tener una estructura de vecindario, no fue suficiente para mejorar los resultados obtenidos y un método de selección de funciones balanceadas se hizo necesario.

Una selección adecuada de funciones balanceadas permite mejorar el desempeño de los algoritmos de búsqueda. Utilizando la representación basada en patrones y la teoría de cúmulos fue posible generar un patrón del comportamiento de la no-linealidad en funciones balanceadas de n variables. Sin embargo, el proceso de localizar dichos patrones se complica cuando el número de variables de las funciones balanceadas aumenta. El algoritmo desarrollado reduce el número de funciones que deben ser utilizadas para generar el patrón de la no-linealidad al explotar la propiedad del diámetro de $\mathcal{G}(n)$ y utilizando como entrada dos funciones balanceadas .

Un mecanismo de selección de funciones basado en patrones debe ser preciso para poder guiar a los algoritmos de búsqueda hacia los óptimos globales. El algoritmo de selección que se presentó permite guiar la búsqueda y reducir el costo computacional para localizar valores óptimos al reducir la cardinalidad del vecindario y el número de ciclos que se ejecuta el algoritmo Metrópolis.

6.1. Trabajo a Futuro

En este trabajo se presentó una estructura de gráfica para representar el espacio de funciones balanceadas y sus propiedades generales, de las cuales sólo se explotaron pocas y faltan más por encontrar. Para esto, es posible utilizar los resultados de la teoría de gráficas para enumerar nuevas propiedades y características que permitan utilizar esta gráfica en diferentes métodos de búsqueda y selección de puntos.

Aunque sólo se utilizaron los algoritmo de búsqueda local y recocido simulado con el algoritmo de selección de puntos, es posible utilizar el método de selección de puntos con otros algoritmos de búsqueda como hill-climbing. Esto permitirá medir la efectividad de los patrones utilizados.

El actual mecanismo de selección de puntos sólo toma en cuenta la no-linealidad de las funciones para generar un conjunto de puntos que servirá de entrada para el algoritmo de k-means. Sin embargo, este criterio de selección puede ampliarse y utilizar otras propiedades como la auto-correlación y extender el mecanismo de búsqueda.

El mecanismo de extrapolación mostrado en el capítulo 4 no es preciso ya que éste disminuye la efectividad del patrón al indeterminar el doble de bits del patrón original. El desarrollo de un mecanismo preciso queda abierto. Dicho mecanismo debe tener la característica de ser rápido pero sin indeterminar una gran cantidad de bits.

Bibliografía

[1] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Octubre 1996.

[2] William Millan, Joanne Fuller, and Ed Dawson. New concepts in evolutionary search for boolean functions in cryptology. *Computational Intelligence*, 20(3):463–474, 2004.

[3] GNU. The gnu multiple precision arithmetic library.

[4] Guillermo Morales Luna. A glimpse to non-linear balanced boolean functions. *The 9th International Conference on Distributed Multimedia Systems*, pages 679–684, Septiembre 2003.

[5] John A Clark, Jeremy L Jacob, Subhamoy Maitra, and Pantelimon Stanica. Almost boolean functions: the design of boolean functions by spectral inversion. *Computational Intelligence*, 20(3):450–462, 1994.

[6] Jennifer Seberry, Xian-Mo Zhang, and Yuliang Zheng. Nonlinearity and propagation characteristics of balanced boolean functions. *Inf. Comput.*, 119(1):1–13, 1995.

[7] Piotr Porwik. The spectral test of the boolean functions nonlinearity. *International Journal of Applied Mathematics and Computer Science*, 13(4):567–575, 2003.

[8] Willi Meier and Othmar Staffelbach. Nonlinearity criteria for cryptographic functions. In *EUROCRYPT '89: Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 549–562, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[9] Subhamoy Maitra. On nonlinearity and autocorrelation properties of correlation immune boolean functions. *Journal of Information Science and Engineering*, 20:305–323, 2004.

[10] Mitchell A. Thornton and D. Michael Miller. Walsh spectrum computations using cayley color graphs. In *IEEE International Midwest Symposium on Circuits and Systems*, volume 1, pages 110–113, August 2001.

[11] Soumen Maity and Thomas Johansson. Construction of cryptographically important boolean functions. *International Conference on Cryptology in India*, 2551:234–245, 2002.

[12] O. S. Rothaus. On bent functions. *J. Comb. Theory, Ser. A*, 20(3):300–305, 1976.

[13] Palash Sarkar and Subhamoy Maitra. Construction of nonlinear boolean functions with important cryptographic properties. In *EUROCRYPT*, pages 485–506, 2000.

[14] Jennifer Seberry, Xian-Mo Zhang, and Yuliang Zheng. Nonlinearly balanced boolean functions and their propagation characteristics (extended abstract). *Advances in Cryptology - CRYPTO '93*, 773:49–60, 1993.

[15] Ren Kui, Park Jaemin, and Kim Kwangjo. On the construction of cryptographically strong boolean functions with desirable trade-off. *Journal of Zhejiang University SCIENCE (ISSN 1009-3095), 2004 (Vol. 5, No. 11)*, 6A(19):358–364, 2005.

[16] John A. Clark, Jeremy L. Jacob, Susan Stepney, Subhamoy Maitra, and William Millan. Evolving boolean functions satisfying multiple criteria. *Progress in Cryptology - INDOCRYPT 2002*, 2551:246–259, December 2002.

[17] Emile Aarts and Jan Korst. *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Inc., New York, NY, USA, 1989.

[18] Yiu Ming Cheung. k-means: A new generalized k-means clustering algorithm. 24:2883–2893, 2003.

[19] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 658:281–297, 1993.