UT T00064-SS1
Don. 2014

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

Laboratorio de Tecnologías de Información,
CINVESTAV-Tamaulipas

Construcción de Covering Arrays Utilizando Vectores Inicializadores

Tesis que presenta:

Aldo Josimar González Gómez

Para obtener el grado de:

Maestro en Ciencias en Computación

Director de la Tesis:
Dr. José Torres Jiménez

Cd. Victoria, Tamaulipas, México. Diciembre, 2013

La tesis presentada por Aldo Josimar González Gómez fue aprobada por:

Dr. Nelson Rangel Valdez

Dr. José Juan García Hernández

Dr. José Torres Jiménez, Director

Cd. Victoria, Tamaulipas, México., 6 de Diciembre de 2013

"UNIX básicamente es un sistema operativo muy simple, pero se tiene que ser un genio para entender su simplicidad"
Dennis Ritchie (1941-2011)

Agradecimientos

- Agradezco a mis compañeros de laboratorio por todos los buenos momentos que vivimos.
- A todos los doctores del CINVESTAV Tamaulipas por compartir su conocimiento conmigo.
- Al personal administrativo del CINVESTAV Tamaulipas por su atención a las necesidades que surgieron durante mi estancia en la institución.
- Al CONACyT por la beca que me proporcionaron durante mi estancia en el CINVESTAV Tamaulipas.
- A mi asesor el Dr. José Torres Jiménez por su paciencia y sus ideas, sin él este trabajo no hubiera sido posible.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Resumen

Construcción de Covering Arrays Utilizando Vectores Inicializadores

por

Aldo Josimar González Gómez

Laboratorio de Tecnologías de Información, CINVESTAV-Tamaulipas
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2013
Dr. José Torres Jiménez, Director

Actualmente los sistemas computacionales forman una parte importante en la vida cotidiana de las personas, por esta razón es muy importante aplicar un conjunto de pruebas de software al sistema computacional. Las pruebas de software pueden diseñarse utilizando objetos combinatorios como los *Covering Arrays*. Un *Covering Array* (CA) es una matriz de tamaño $N \times k$ donde en cada submatriz de tamaño $N \times t$ existe al menos un renglón por cada tupla del conjunto $\mathbb{Z}_v^t$, donde N es el número de renglones, k es el número de columnas, t es la fuerza de cobertura y v es el número de símbolos, un *Covering Array* se denota $CA(N; t, k, v)$. Existen muchos métodos para construir CAs y ellos pueden clasificarse en métodos algebraicos, métodos heurísticos, métodos voraces, métodos exactos y métodos de manipulación. En esta tesis se presentará un método generalizado para construir CA usando Vectores Inicializadores (VI) usando las operaciones de *Rotación* y *Traslación* de fuerza dos hasta seis. La operación de *Rotación* sobre el VI de tamaño k construye $\mathcal{R}$ vectores donde $1 \leq \mathcal{R} \leq k$ rotando los elementos del VI. La operación de *Traslación* sobre el VI de tamaño k construye $\mathcal{T}$ vectores donde $1 \leq \mathcal{T} \leq v$ rotando los símbolos del VI. Un CA es el producto de las operaciones de *Rotación* y *Traslación* al VI de tamaño k, aplicando la operación de *Traslación* $\mathcal{T}$ veces y por cada *Traslación* aplicando la operación de *Rotación* $\mathcal{R}$ veces, construyendo un CA de tamaño $\mathcal{TR} \times k$. Los símbolos se pueden dividir en dos subconjuntos llamados símbolos fijos $\mathcal{F}$ y símbolos no fijos. Los símbolos $\mathcal{F}$ no son afectados por la operación de *Traslación* y esto propicia que el número de copias necesarias sea menor si $|\mathcal{F}| > 0$, el CA resultante será de tamaño $(v - |\mathcal{F}|)k$ renglones y k columnas. Dado que los símbolos fijos $\mathcal{F}$ no son trasladados las interacciones entre

ellos no son cubiertas por la matriz de tamaño $(v - |\mathcal{F}|)k$, es necesario agregar un CA de alfabeto $|\mathcal{F}|$ y fuerza t con k columnas. Utilizando este método se mejoraron dos cotas, se igualaron 31 y solo dos casos no se pudo igualar al mejor reportado.

Abstract

Covering Array Construction using Starter Vectors

by

Aldo Josimar González Gómez

Information Technology Laboratory, CINVESTAV-Tamaulipas

Center for Research and Advanced Studies from the National Polytechnic Institute, 2013

Dr. José Torres Jiménez, Advisor

Nowadays the computational systems have become an important part of everyday life of the people, for this reason it is important to apply a test suite over the computational system. The software test can be designed using combinatorial objects like *Covering Arrays*. A *Covering Array* (CA) is a matrix of size $N \times k$ where in each submatrix of size $N \times t$ exists at least one row of each tuples of set the $\mathbb{Z}_v^t$, where N is the number of rows, k is the number of columns, t is the strength of coverage and v is the number of symbols and is denoted by $CA(N; t, k, v)$. To construct CAs exists many methods and they can be classified in algebraic methods, heuristic methods, greedy methods, exact methods and manipulation methods. In the present master thesis it will be presented a generalized method to construct CAs using a starter vectors (SV) and the operations of *Rotation* and *Translation* for strength two to six. The Rotation operation over a SV of size k builds $\mathcal{R}$ vectors, $1 \leq \mathcal{R} \leq k$, rotating elements of the SV. The Translation operation over a SV of size k builds $\mathcal{T}$ vectors, $1 \leq \mathcal{T} \leq v$, rotating symbols of the SV. A CA is the product of the operations of Rotation and Translation to SV of size k, applying the Translation operation $\mathcal{T}$ times and for each Translation apply the Rotation $\mathcal{R}$ times, building a matrix of $\mathcal{T} \times \mathcal{R}$ rows and k columns. The symbols can be divided into two subsets called fixed symbols $\mathcal{F}$ and non fixed symbols. The $\mathcal{F}$ symbols are unaffected for the Translation operation therefore the number of needed copies is less when $|\mathcal{F}| > 0$, the CA obtained will be of size $(v - |\mathcal{F}|)\, k$ rows and k columns. Because the $\mathcal{F}$ symbols are not translated, the interaction between them is not covered for the matrix of size $(v - |\mathcal{F}|)k$, is necessary to add a CA of $v = |\mathcal{F}|$. Using this generalized method could improve two bounds, 31 bounds were equalized and only two bounds could not equal that best reported bound.

Nomenclatura

Acrónimos

OA	*Orthogonal Array*
CA	*Covering Array*
MCA	*Mixed Covering Array*
CAN	*Covering Array Number*
CACP	*Covering Array Construction Problem*
VI	*Vector Inicializador*
VC	Vector Canónico
RL	Recálculo Local

Notación

$OA_\lambda(N;k,t,v)$	*Orthogonal Array* de tamaño $N \times k$, alfabeto v, fuerza t e índice λ.
$CA(N;t,k,v)$	*Covering Array* de tamaño $N \times k$, alfabeto v y fuerza t.
$MCA(N;t,k,v_0,\ldots,v_{k-1})$	*Mixed Covering Array* de tamaño $N \times k$, fuerza t y un alfabeto diferente por cada columna.
$\mathbb{Z}_v$	Conjunto de enteros de 0 hasta $v-1$.
$\mathbb{Z}_v^t$	Producto cartesiano de t conjuntos iguales a $\mathbb{Z}_v$.
N	Número de renglones de un OA, un CA o un MCA.
k	Número de columnas de un OA, un CA o un MCA.
t	Valor de la fuerza de un OA, un CA o un MCA.
v	Valor del alfabeto de un OA y un CA.

1

Introducción

En este capítulo se presenta de manera breve la importancia de las pruebas de software. El capítulo se divide en once secciones, en la primer sección se presentará una breve introducción sobre la importancia del software en la vida de las personas. En la segunda sección se hablará del proceso de las pruebas de software, así como las reglas que siguen. La tercer sección contendrá el diseño de las pruebas de software, mencionando dos tipos de pruebas de software: las pruebas de caja blanca y las pruebas de caja negra. De cada tipo de prueba se mencionarán las diferentes técnicas para realizar el diseño de cada tipo de prueba. En la cuarta sección se profundizará en una técnica del tipo de prueba de caja negra, las pruebas de *Orthogonal Array*, explicando una manera para diseñar las pruebas utilizando objetos combinatorios. En la quinta sección se definirán los objetos combinatorios *Orthogonal Array*, *Covering Array* y *Mixed Covering Array*, también se explicará el problema de construcción de *Covering Array* y se hablará de la clasificación de los diferentes métodos existentes para construir *Covering Arrays*. La sexta sección contendrá el planteamiento del problema, explicando de manera breve los trabajos en los que se inspiró este trabajo de tesis. La séptima sección

contendrá el objetivo general del trabajo de tesis. La octava sección presenta los objetivos específicos, la novena sección contendrá la metodología de investigación que se siguió y la décima sección la organización del resto del documento de tesis. Al final se muestra un resumen de este capítulo.

1.1 Introducción

En la actualidad, los sistemas computacionales forman parte de la vida cotidiana de las personas [1], éstos pueden ser encontrados en diferentes dominios de aplicación, tales como, empresariales, médicas, astronómicas, mecánicas, etc.

Puesto que los sistemas computacionales forman parte de la vida cotidiana de las personas, muchas personas han tenido dificultades por un mal funcionamiento de un sistema computacional [2]. Las dificultades ocasionadas por un mal funcionamiento de algún sistema computacional dependen del dominio de aplicación, por ejemplo, en el dominio de aplicaciones empresariales puede ocasionar pérdidas económicas y en el dominio de aplicaciones médicas puede ocasionar pérdidas de vidas humanas [3, 4].

Puesto que las personas interactúan con los sistemas computacionales, es muy importante que el sistema computacional funcione de acuerdo a sus especificaciones. Para validar que el sistema computacional funcione de acuerdo a sus especificaciones es necesario aplicar un conjunto de pruebas a cada uno de sus componentes de software. En la siguiente sección se hablará de las pruebas de software junto con los objetivos que éstas tienen.

1.2 Pruebas de Software

Puesto que un sistema computacional debe funcionar de acuerdo a sus especificaciones, es muy importante aplicar un conjunto de pruebas a cada uno de los componentes de un sistema computacional. Las pruebas de software siguen tres reglas que pueden servir como objetivos [5]:

a) El proceso de pruebas consiste en ejecutar un programa con la intención de encontrar un error.

b) Un buen caso de prueba es aquel que tiene una gran probabilidad de encontrar un error no descubierto.

c) Una prueba es útil si encuentra un error que aún no se ha descubierto.

Tomando en cuenta las reglas anteriores, el proceso para las pruebas de software se puede realizar en los siguientes tres pasos:

1. Diseñar los casos de prueba.
2. Ejecutar los casos de prueba.
3. Si se encontró al menos un error hay que corregir el error en el software y regresar al paso dos, en caso contrario se termina el proceso de pruebas de software.

La Figura 1.1 muestra el proceso de prueba del software. En la siguiente sección se hablará del diseño de casos de las pruebas de software.

1.3 Diseño de Casos de Prueba

Para poder aplicar las pruebas a un componente de software, es necesario diseñar las pruebas a aplicar. El diseño de casos de prueba se puede realizar de diferentes maneras, dos de esas maneras son [6]:

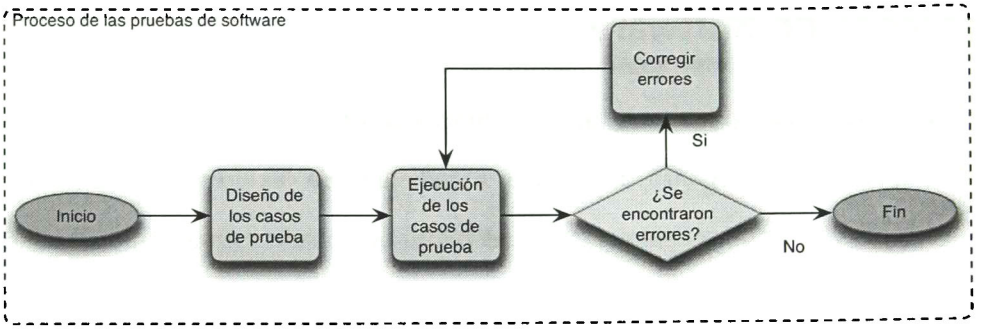


Figura 1.1: El proceso de las pruebas de software consta de tres pasos, en el primer paso se diseñan los casos de prueba, en el segundo paso se ejecutan los casos de prueba diseñados, si se encuentra al menos un error en el tercer paso se corrigen los errores y se regresa al paso dos.

a) Pruebas de caja negra.

b) Pruebas de caja blanca.

Las pruebas de caja negra se diseñan para verificar que el software funcione de acuerdo a sus especificaciones y las pruebas de caja blanca se diseñan considerando el funcionamiento interno del componente de software. Cada uno de estos tipos de prueba serán descritos en las siguientes subsecciones junto con las técnicas existentes para realizar cada tipo de prueba.

1.3.1 Pruebas de Caja Blanca

Las pruebas de caja blanca (Figura 1.2), también conocidas como pruebas de caja de cristal, se enfocan en probar las estructuras internas (instrucciones de condición, ciclos, etc.) de cada componente de software [7], probando los caminos lógicos a través del software. Puesto que este tipo de prueba de software verifica el funcionamiento interno del componente de software, al cambiar cualquier estructura interna estas pruebas deben ser modificadas de acuerdo a la estructura que se modificó para así verificar dicha estructura.

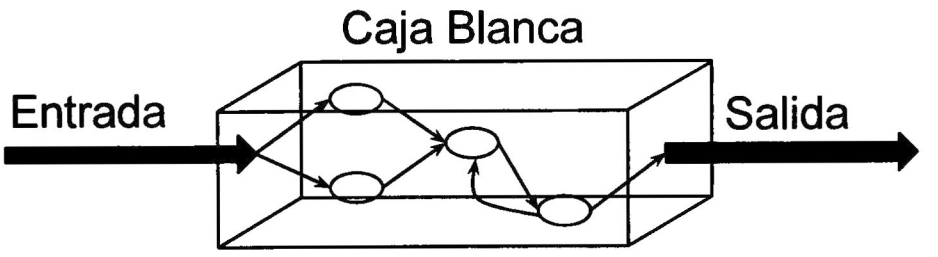


Figura 1.2: Pruebas de caja blanca. Este tipo de pruebas verifica el funcionamiento interno del componente de software.

Existen diversas técnicas para realizar las pruebas de caja blanca, las cuales se pueden clasificar en pruebas de *camino básico* y pruebas de *control de estructura*. Cada clasificación tiene una diversidad de técnicas para diseñar las pruebas de caja blanca enfocándose en probar las estructuras internas de manera diferente.

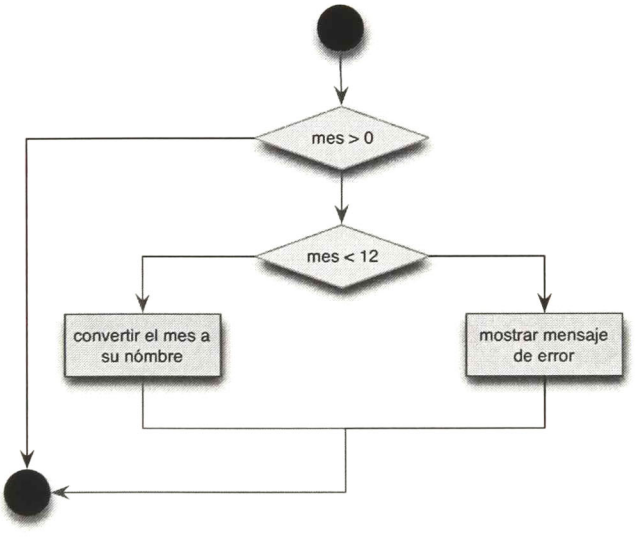


Figura 1.3: Diagrama que transforma un número que está entre 1 y 12 y lo transforma al nombre del mes que le corresponde, el camino básico se resalta en color rojo.

- **Prueba del camino básico** [5]. Las pruebas de camino básico utilizan los diagramas de flujo[1]

[1] un diagrama de flujo es la representación gráfica del proceso

para diseñar estos casos de prueba, el camino básico es aquel camino que ejecuta la mayoría de las instrucciones del proceso a probar, en la Figura 1.3 se muestra un ejemplo de un diagrama de flujo que transforma un número que está entre 1 y 12 al nombre del mes, el camino básico (resaltado en color rojo) es cuando el mes está entre el rango de 1 y 12 y transforma el número al mes correspondiente. Existen diferentes maneras para realizar esta técnica las cuales se mencionan a continuación, complejidad cíclica y matrices del diagrama.

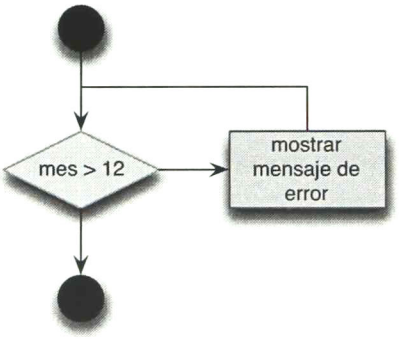


Figura 1.4: Ejemplo de un diagrama que verifica que la variable *mes* sea menor a 12, la prueba de control de estructura verifica que muestre el mensaje de error cada vez que *mes* sea mayor a 12.

- **Prueba de control de estructura** [5]. Al igual que las pruebas de camino básico, las pruebas de control de estructura utilizan los diagramas de flujo para identificar las estructuras internas que modifican el flujo de ejecución de las instrucciones de un programa, estas estructuras internas son: condiciones y ciclos, la Figura 1.4 muestra un fragmento de un diagrama de flujo que verifica que la variable *mes* sea menor a 12, la prueba de control de estructura verifica que muestre el mensaje de error cuando *mes* sea mayor a 12; y cuando *mes* sea menor a 12 no muestre el mensaje de error. Las técnicas para realizar este tipo de prueba son: pruebas de condición, pruebas de flujo de datos y pruebas de ciclos.

Figura 1.5: Pruebas de caja negra. Las pruebas de caja negra se enfocan en probar las especificaciones del software.

1.3.2 Pruebas de Caja Negra

Las pruebas de caja negra (Figura 1.5) se enfocan en que el software funcione de acuerdo a sus especificaciones, buscando errores de: especificaciones incorrectas o especificaciones faltantes, errores de interfase, errores de comportamiento o desempeño y errores de inicialización y terminación [5]. Puesto que estas pruebas validan las especificaciones del software al cambiar cualquier especificación estas pruebas deben de ser modificadas para validar la nueva especificación.

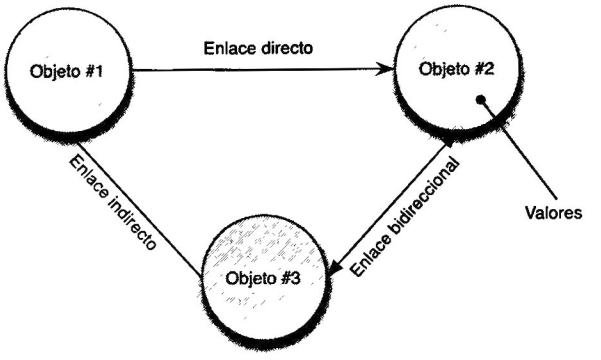


Figura 1.6: Ejemplo del modelo gráfico de pruebas, los nodos representan los objetos del software y los arcos las relaciones que hay entre ellos.

Para realizar las pruebas de caja negra existen diversas técnicas las cuales se nombrarán y se explicarán a continuación.

- **Basados en modelos gráficos** [5]. Las pruebas basadas en modelos gráficos utilizan un diseño gráfico del componente de software para verificar las relaciones existentes entre los componentes del software y así encontrar errores, la Figura 1.6 muestra un ejemplo de las relaciones entre tres objetos. Existen diversos métodos que utilizan modelos gráficos, algunos de ellos son: modelado del flujo de transacción (los nodos representan pasos en alguna transacción), modelado de estado finito (los nodos representan diferentes puntos de control del software), modelado de flujo de datos (los nodos son objetos del programa y los arcos son las transformaciones que ocurren al cambiar de un objeto a otro) y modelado relacionado con el tiempo (Los nodos son objetos del programa y los uniones son conexiones secuenciales entre los objetos).

Figura 1.7: Ejemplo de la partición equivalente, en el ejemplo se muestran las clases de equivalencia: valores positivos válidos, valores positivos no válidos y valores negativos no válidos, del parámetro mes.

- **Partición equivalente** [5]. La técnica de partición equivalente divide los datos de entrada en clases de equivalencia, en donde una clase de equivalencia representa un conjunto de estados válidos o no válidos para los valores de entrada, en la Figura 1.7 se muestra los valores de entrada de la variable *mes*, dividiendo las entradas en tres estados, valores válidos positivos,

valores no válidos positivos, valores no válidos negativos. Al separar los datos de entrada en clases de equivalencia se seleccionan aquellos datos que prueben el mayor número de estados del componente de software.

- **Análisis de valores límite** [5]. El análisis de valores límite complementa la partición equivalente, seleccionando los valores que se encuentran en los límites de las clases de equivalencia para probar el componente de software. Por ejemplo, si para el parámetro *mes* el rango de valores de la clase está delimitado por 1 y 12, las pruebas de valores usarán los valores 1 y 12.

- **Pruebas de Orthogonal Array** [5]. Las pruebas de *Orthogonal Array* son una alternativa para una prueba exhaustiva[2] proporcionando en un menor número de casos de prueba todas las combinaciones de un cierto tamaño de los parámetros de entrada. Por ejemplo, si queremos probar una función que cuenta con cuatro entradas y cada entrada tiene tres opciones diferentes, utilizando una prueba exhaustiva se requerirían $3^4 = 81$ casos, mientras que con la prueba de *Orthogonal Array* si queremos verificar todas las parejas de los valores de entrada solo se requerirían nueve casos.

1.4 Pruebas de Orthogonal Array

Las pruebas de *Orthogonal Array* (también llamadas pruebas de interacción) son una opción para las pruebas exhaustivas, puesto que, las pruebas exhaustivas verifican todas las combinaciones de las opciones de todos los parámetros de las entradas, mientras que las pruebas de interacción garantizan que todas las combinaciones de un tamaño menor están cubiertas en un número reducido de casos de prueba. El tamaño de combinaciones se denota por t, y éste indica que todas las combinaciones de tamaño t de los valores de las entradas están cubiertas en los casos de prueba.

[2]Verifican todas las combinaciones de los parámetros de entrada

Tabla 1.1: Estados de los cuatro parámetros de la función *enviar*.

P_1	P_2	P_3	P_4
enviar ahora	enviar ahora	enviar ahora	enviar ahora
en media hora	enviar en media hora	enviar en media hora	enviar en media hora
enviar en dos horas	enviar en dos horas	enviar en dos horas	enviar en dos horas

Para ejemplificar esta prueba, consideramos que la función *enviar* tiene cuatro parámetros de entrada P_1, P_2, P_3 y P_4, cada parámetro tiene tres opciones (Tabla 1.1), a cada opción le asignamos un valor numérico de la siguiente manera:

0 = enviar ahora.

1 = enviar en media hora.

2 = enviar en dos horas.

Una prueba exhaustiva probaría 81 casos, mientras que la prueba de *Orthogonal Array* solo probarían 9 casos. En la Tabla 1.2 se muestran los casos que se requerirían para esta prueba (para construir los casos se pueden utilizar diversos procedimientos, uno de ellos es el que se encuentra reportado en [8]).

Para crear los casos de las pruebas de *Orthogonal Array*, se pueden utilizar diversos métodos, uno de ellos es utilizando objetos combinatorios.

En [9, 10, 11, 12, 13, 14, 15] se explica la importancia de utilizar objetos combinatorios para representar conjuntos de casos de pruebas de software, al igual que sus ventajas sobre las pruebas exhaustivas, en los trabajos [12, 16, 15] se utilizan los objetos combinatorios para representar conjuntos de pruebas en hardware y en [17] se usan los objetos combinatorios aplicados a la agricultura, medicina y manufactura.

Tabla 1.2: Casos de prueba de *Orthogonal Array* para los parámetros P_1, P_2, P_3 y P_4; y cada parámetro tiene tres opciones diferentes.

Caso de prueba	Parámetros			
	P_1	P_2	P_3	P_4
1	0	0	0	0
2	0	1	1	1
3	0	2	2	2
4	1	0	1	2
5	1	1	2	0
6	1	2	0	1
7	2	0	2	1
8	2	1	0	2
9	2	2	1	0

La importancia de utilizar objetos combinatorios para diseñar las pruebas de *Orthogonal Array* se muestra en el trabajo realizado por Kuhn *et al.* [18]. Khun *et al.* analizan las fallas encontradas en cuatro tipos de sistemas computacionales utilizando un objeto combinatorio llamado *Covering Array*. Los sistemas computacionales que se analizaron fueron: *NASA Goddard Space Flight Center* (NASA GSFC), Dispositivos Médicos, el navegador web Mozilla y el servidor Apache. Estas fallas fueron encontradas durante el desarrollo de cada sistema. La Tabla 1.3 muestra en porcentajes las fallas encontradas en relación al grado de interacción.

Tabla 1.3: Porcentaje del total de fallas encontradas en relación al grado de interacción (t).

t	Dispositivos Médicos	Mozilla	NASA GSFC	Apache
1	66 %	29 %	42 %	68 %
2	97 %	76 %	70 %	93 %
3	99 %	95 %	89 %	98 %
4	100 %	97 %	96 %	100 %
5	100 %	99 %	96 %	100 %
6	100 %	100 %	100 %	100 %

Al observar la Tabla 1.3 se aprecia que al incrementar el grado de interacción, el número de errores descubiertos se va incrementando y en el caso de todos los sistemas computacionales probados, con

$t = 6$ se descubrió el 100 % de los errores. Concluyendo que entre mayor sea el grado de interacción (mayor valor de t) la cantidad de errores descubiertos será mayor (Figura 1.8).

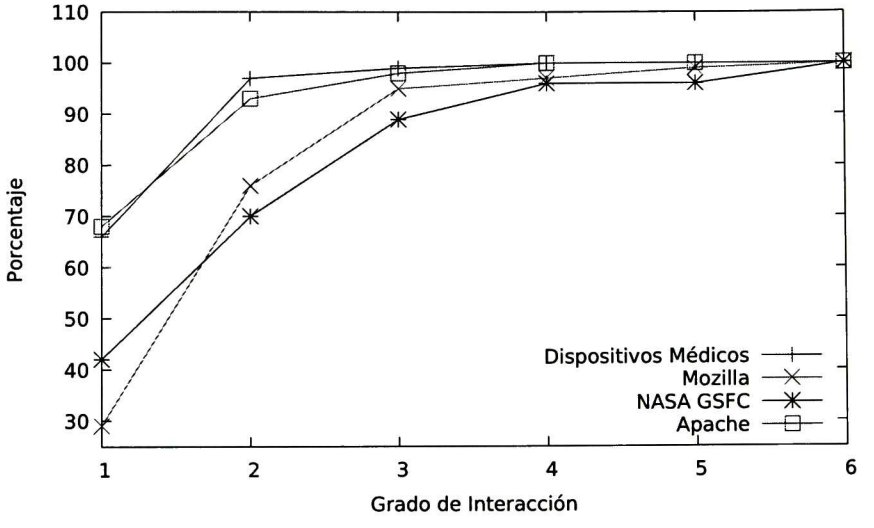


Figura 1.8: Gráfica de las fallas detectadas en cada software en relación a la interacción t.

Los diferentes objetos combinatorios utilizados para construir las pruebas de *Orthogonal Array* se detallarán en la siguiente sección.

1.5 Objetos Combinatorios

Algunos objetos combinatorios han sido usados para crear los casos de prueba de software, uno de estos objetos combinatorios es el *Orthogonal Array*, el cual se define de la siguiente manera.

Definición 1 (Orthogonal Array).

Un *Orthogonal Array* (OA) es una matriz de tamaño $N \times k$, en donde en cada submatriz de tamaño $N \times t$ existen exactamente λ renglones que cubren cada tupla del conjunto $\mathbb{Z}_v^t$ y se denota por $OA_\lambda(N; k, t, v)$ [17].

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 2 & 0 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 0 & 1 \\ 2 & 0 & 1 & 1 \\ 0 & 2 & 1 & 2 \\ 1 & 0 & 2 & 2 \\ 2 & 1 & 0 & 2 \end{pmatrix}$$

Figura 1.9: $OA_1(9; 4, 2, 3)$. *Orthogonal Array* de 9 renglones, 4 columnas, de alfabeto 3 y fuerza 2. En cada submatriz de tamaño 9×2 existe exactamente un renglón para cada tupla del conjunto $\mathbb{Z}_3^2$.

Puesto que el OA debe contener exactamente λ renglones para cada t-tupla del conjunto $\mathbb{Z}_v^t$, el parámetro λ limita la existencia de los OA para cualquier valor de k, t y v, por ejemplo, para los valores $k = 5$, $v = 3$, $t = 2$ y $\lambda = 1$ no existe un OA que tenga exactamente un renglón para cada t-tupla del conjunto $\mathbb{Z}_3^2$, pero si existe un OA para $k = 4$, $v = 3$, $t = 2$ y $\lambda = 1$(Figura 1.9).

Otro objeto combinatorio es el *Covering Array* y éste se define de la siguiente manera.

Definición 2 (Covering Array).

Un *Covering Array* (CA) es una matriz de tamaño $N \times k$ en donde en cada submatriz de tamaño $N \times t$ existe al menos un renglón para cada tupla del conjunto $\mathbb{Z}_v^t$. El parámetro t es llamado fuerza y el parámetro v es llamado alfabeto y se denota por $CA(N; t, k, v)$ [19].

Un CA es una versión relajada de un OA, puesto que, en cada submatriz de tamaño $N \times t$ del CA existe al menos un renglón para cada t-tupla del conjunto $\mathbb{Z}_v^t$, mientras que en el OA, en cada submatriz de tamaño $N \times t$ deben existir exactamente λ renglones para cada t-tupla del conjunto

$\mathbb{Z}_v^t$, por ejemplo, para los valores $k = 5$, $v = 3$ y $t = 2$ existe un CA de 11 renglones (Figura 1.10) y para los mismos valores con $\lambda = 1$ no existe un OA.

$$\begin{pmatrix} 2 & 0 & 1 & 1 & 0 \\ 0 & 1 & 2 & 1 & 1 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 2 & 2 \\ 0 & 1 & 0 & 2 & 0 \\ 2 & 2 & 2 & 2 & 1 \\ 1 & 2 & 0 & 1 & 2 \\ 1 & 0 & 1 & 2 & 1 \\ 1 & 1 & 2 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 \\ 2 & 1 & 1 & 0 & 2 \end{pmatrix}$$

Figura 1.10: $CA(11; 2, 5, 3)$. Este CA tiene 11 renglones, 5 columnas, es de fuerza 2 y de alfabeto 3. En cada una de sus submatrices de tamaño 11×2 existen todas las combinaciones de los tres valores diferentes, $\mathbb{Z}_3^2$.

A un CA de N renglones, k columnas, fuerza t y alfabeto v se le pueden aplicar las siguientes operaciones:

a) Permutación de renglones.

b) Permutación de columnas.

c) Permutación de elementos en cualquier subconjunto de columnas.

Al aplicar cualquiera de estas operaciones (o una combinación de ellas) a un CA se obtiene un CA isomórfico. Un CA isomórfico se define de la siguiente manera:

DEFINICIÓN 3 (CAs isomórficos).

Dos CA $\mathcal{A}$ y $\mathcal{B}$ son isomórficos entre sí, si a partir de $\mathcal{A}$ se puede obtener $\mathcal{B}$ a través de alguna combinación de permutaciones de renglones, permutaciones de columnas y permutaciones de elementos en cualquier subconjunto de columnas [20].

Si a un $CA(N;t,k,v)$ se le aplica la operación de permutación de renglones existe un total de $N!$ CAs isomórficos, si se aplica la operación de permutación de columnas existe un total de $k!$ CAs isomórficos y si se aplica la operación de permutación de elementos en cualquier subconjunto de columnas existe un total de $v!$ por cada columna, dando un total de $(v!)^k$ de CAs isomórficos. La operación de permutación de elementos en cualquier subconjunto de columnas se refiere al reetiquetado de los elementos. Si aplicamos las tres operaciones a un CAs existen un total de $N!k!(v!)^k$ CAs isomórficos.

Por ejemplo, si denotamos la permutación de renglones por ϵ, la permutación de columnas la denotamos por ι y la permutación de elementos lo denotamos por ς y si tenemos un $CA(6;2,5,2)$, con $\epsilon = (0,1,2,3,4,5)$, $\iota = (0,1,2,3,4)$ y $\varsigma = (0,0,0,0,0)$; y cambiamos cada permutación por las siguientes: $\epsilon' = (3,1,2,5,4,0)$, $\iota' = (4,2,0,3,1)$ y $\varsigma' = (0,1,0,1,0)$, nos da un CA isomórfico. La Figura 1.11 muestra los dos CAs isomórficos anteriores.

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \begin{matrix} \epsilon{=}(0,1,2,3,4,5) \\ \iota{=}(0,1,2,3,4) \\ \varsigma{=}(0,0,0,0,0) \end{matrix} \qquad \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{pmatrix} \quad \begin{matrix} \epsilon'{=}(3,1,2,5,4,0) \\ \iota'{=}(4,2,0,3,1) \\ \varsigma'{=}(0,1,0,1,0) \end{matrix}$$

Figura 1.11: $CA(6;2,5,2)$, Ejemplo de CAs isomórficos permutando renglones, permutando columnas y permutando elementos.

El CA es utilizado cuando se requiere que todas las columnas contengan el mismo conjunto $\mathbb{Z}_v$, pero cuando se requiere que las columnas tengan diferentes $\mathbb{Z}_v$ se utiliza otro objeto combinatorio llamado *Mixed Covering Array*, el cual se define de la siguiente manera:

Definición 4 (Mixed Covering Array).

Un *Mixed Covering Array* (MCA) es una matriz de tamaño $N \times k$ donde cada columna puede tener un $\mathbb{Z}_v$ diferente, denotado por un vector $\mathcal{V}$, donde un elemento particular se denota por v_j, $0 \leq j \leq k-1$. En cada submatriz de tamaño $N \times t$ cubre al menos una vez todas las t-tupas del producto cartesiano de t elementos de v_j . El vector $\mathcal{V}$ es el vector de alfabetos y t es la fuerza y se denota $MCA(N; t, k, \mathcal{V})$ [21].

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
3 & 1 & 0 & 1 & 2 & 0 & 1 & 0 & 0 \\
2 & 2 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 3 & 1 & 1 & 0 & 1 & 2 & 2 & 0 \\
0 & 2 & 2 & 1 & 3 & 2 & 2 & 0 & 0 \\
3 & 0 & 2 & 0 & 0 & 2 & 0 & 1 & 1 \\
2 & 1 & 1 & 0 & 3 & 2 & 2 & 0 & 1 \\
3 & 3 & 3 & 3 & 1 & 2 & 2 & 0 & 0 \\
2 & 0 & 3 & 1 & 3 & 0 & 1 & 2 & 2 \\
0 & 1 & 3 & 2 & 0 & 1 & 0 & 1 & 1 \\
2 & 2 & 0 & 3 & 0 & 0 & 1 & 2 & 2 \\
2 & 3 & 2 & 2 & 2 & 0 & 2 & 1 & 0 \\
0 & 3 & 1 & 0 & 1 & 0 & 1 & 1 & 2 \\
1 & 1 & 2 & 3 & 1 & 1 & 1 & 2 & 2 \\
1 & 2 & 3 & 0 & 2 & 2 & 1 & 2 & 1 \\
3 & 2 & 1 & 2 & 3 & 1 & 1 & 2 & 2 \\
0 & 0 & 1 & 3 & 2 & 1 & 0 & 2 & 2 \\
1 & 0 & 0 & 2 & 1 & 2 & 2 & 0 & 2 \\
1 & 3 & 0 & 3 & 3 & 0 & 0 & 1 & 1
\end{pmatrix}$$

Figura 1.12: Ejemplo de un $MCA(19; 2, 9, 4, 4, 4, 4, 4, 3, 3, 3, 3)$. En este MCA las primeras 5 columnas son de alfabeto 4 y las siguientes cuatro son de alfabeto 3, tiene 19 renglones y es de fuerza 2.

Un MCA es una versión relajada de un CA, puesto que, el CA en cada columna el conjunto $\mathbb{Z}_v$ es el mismo, mientras que en el MCA el conjunto $\mathbb{Z}_v$ puede ser diferente en cada columna. En la Figura 1.12 se muestra el $MCA(19; 2, 9, 4, 4, 4, 4, 4, 3, 3, 3, 3)$, en el cual el vector $\mathcal{V}$ indica que en las primeras cinco columnas el alfabeto es 4 y en las últimas cuatro columnas el alfabeto es 3.

Al cambiar un elemento de alguna entrada del CA o MCA sin que se afecte el grado de cobertura del CA o MCA, este elemento es un elemento comodín. Un elemento comodín se define de la siguiente manera:

Definición 5 (Elemento Comodín).

Un elemento comodín es aquel elemento que puede tomar cualquier valor del alfabeto sin que afecte el grado de cobertura en el un CA o un MCA.

Los CA y MCA pueden no tener entradas con elemento comodines. En esta tesis los elementos comodines se denotan por el valor del alfabeto de la columna en la que se encuentra. La Figura 1.13 muestra un ejemplo de un CA con dos elementos comodines en las últimas dos columnas del primer renglón.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & \mathbf{2} & \mathbf{2} \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 1.13: $CA(7; 2, 11, 2)$. CA con dos elementos comodines en las últimas dos columnas del primer renglón.

Cuando se tienen dos conjuntos de pruebas con la misma cantidad de parámetros, el mismo grado de interacción entre sus parámetros y la misma cantidad de posibles valores pero uno de ellos tiene mas casos de prueba y el otro tiene menos casos de prueba, al aplicar el de menor número de casos, se tiene la misma cobertura entre los parámetros pero con un esfuerzo menor. Por esta razón es importante obtener CAs con el valor mínimo de N. El valor mínimo de renglones posibles en un CA se denomina *Covering Array Number* (CAN) y se denota por $CAN(t, k, v)$ (1.1).

$$CAN(t, k, v) = min\{(N \mid \exists\, CA(N; t, k, v)\} \tag{1.1}$$

Dados los valores de t, k y v, el problema de construir un $CA(N; t, k, v)$ con $N = CAN(t, k, v)$ renglones es llamado el *Problema de Construcción de Covering Array* ($CACP$ por sus siglas en inglés "*Covering Array Construction Problem*"). Actualmente no se ha demostrado si el $CACP$ pertenece a la clase de problemas NP-Completos [22], sin embargo, algunos problemas relacionados con el $CACP$ sí pertenecen a los problemas NP-Completos, por ejemplo, si denotamos por $\mathcal{P}$ el conjunto de t-tuplas no cubiertas y tenemos p como un número de t-tuplas por cubrir, el determinar si existe un renglón que cubra p t-tuplas de $\mathcal{P}$ es NP-Completo [23].

Existe una gran cantidad de métodos para construir CAs, los cuales se pueden clasificar en métodos exactos, métodos algebraicos, métodos heurísticos, métodos voraces y métodos de manipulación [24, 25, 26, 27]. En los métodos algebraicos destacan dos métodos que resuelven el $CACP$ en tiempo polinomial, pero estos métodos están restringidos a un conjunto específico de valores para los parámetros t, k y v [8, 28, 29, 30]. En esta tesis se propone un método algebraico que usa vectores inicializadores (DEFINICIÓN 6) para construir CAs para $t = 2, \ldots, 6$.

1.6 Planteamiento del Problema

La construcción de CAs utilizando un vector inicializador (VI) inició en 1999 con Chateauneuf *et al.*, utilizando solo las operaciones de *Rotación* y *Traslación*, después, en el 2008 Meagher y Stevens ajustaron el método para construir CAs de fuerza 2 utilizando un símbolo fijo, en el 2010 Lobb utilizó utilizó tres símbolos fijos y en el 2012 Lobb *et al.* generalizaron el manejo de los símbolos a cualquier cantidad (Tabla 1.4).

Tabla 1.4: Tabla de los trabajos que usan un VI para construir CA.

Nombre	Autor(es)	Año	Usa símbolos fijos	
Covering arrays of strength three	Chateauneuf *et al.*	1999	No	[31]
On the state of strength 3 covering arrays	Chateauneuf y Kreher	2002	No	[32]
Group construction of covering arrays	Meagher y Stevens	2005	Sí	[33]
Strength two covering arrays: existence tables and projection	Colbourn	2008	Sí	[34]
Covering arrays from cyclotomy	Colbourn	2010	No	[35]
Hybrid strength two Covering Array constrctions: using cover starters to create covering arrays	Lobb	2010	Sí	[36]
Cover starters for covering arrays of strength two	Lobb *et al.*	2012	Sí	[37]

Este trabajo de tesis es una generalización de la construcción de un VI de Lobb *et al.* [37] y de Colbourn [35].

Definición 6 (Vector Inicializador).

Un vector inicializador (VI) es un vector de tamaño k y puede contener símbolos del conjunto $\mathbb{Z}_v$ en cada posición del VI, el espacio de búsqueda que se tiene es v^k.

La manera en que se construye el CA utilizando el VI, es aplicando las operaciones de *Rotación* (Definición 7) y *Traslación* (Definición 8) al VI.

Definición 7 (Rotación).

Dado un vector $\mathcal{S}$ de tamaño k donde un elemento de $\mathcal{S}$ se designa por $\mathcal{S}_i$, $0 \leq i \leq k-1$, la operación de *Rotación* produce en vector $\mathcal{S}'_i = \mathcal{S}_{(i+\sigma) \bmod k}$ donde $0 \leq \sigma \leq k-1$.

Definición 8 (Traslación).

Dado un vector $\mathcal{S}$ de tamaño k donde un valor particular de $\mathcal{S}$ se designa por $\mathcal{S}_i$, $0 \leq i \leq k-1$, la operación de *Traslación* sobre el vector $\mathcal{S}$ produce un vector $\mathcal{S}'_i = (\mathcal{S}_i + \delta) \bmod v$ donde $0 \leq \delta \leq v-1$.

Tabla 1.5: Construcción de un CA a través de las operaciones de *Rotación* y *Traslación* para los valores $k = 5$, $v = 3$ y $t = 2$, asignando a δ los valores de 0 hasta 2 y a σ los valores de 0 hasta 4 al $VI = \{0, 0, 0, 0, 1\}$; construyendo un $CA(15; 2, 5, 3)$.

Vectores					δ	σ
0	0	0	0	1	0	0
1	0	0	0	0	0	1
0	1	0	0	0	0	2
0	0	1	0	0	0	3
0	0	0	1	0	0	4
1	1	1	1	2	1	0
2	1	1	1	1	1	1
1	2	1	1	1	1	2
1	1	2	1	1	1	3
1	1	1	2	1	1	4
2	2	2	2	0	2	0
0	2	2	2	2	2	1
2	0	2	2	2	2	2
2	2	0	2	2	2	3
2	2	2	0	2	2	4

Lobb *et al.* [37] construye un CA utilizando un VI y a éste le aplica la operación de *Traslación* v veces asignando a δ los valores desde 0 hasta $v-1$ y por cada operación de *Traslación* se aplica la *Rotación* k veces asignando σ los valores desde 0 hasta $k-1$, produciendo un CA de tamaño $vk \times k$. En la Tabla 1.5 se muestra el proceso de construcción del CA utilizando las operaciones de *Rotación* y *Traslación* para los valores de $k=5$, $v=3$ y $t=2$ con $\delta = 0,\ldots,2$ y $\sigma = 0,\ldots,4$ al $VI = \{0,0,0,0,1\}$ construyendo un CA de 15 renglones.

Para construir CAs con un menor número de renglones, Lobb *et al.* dividen el conjunto $\mathbb{Z}_v$ en dos subconjuntos, el subconjunto de símbolos fijos $\mathcal{F}$ y el subconjunto de símbolos no fijos. Donde $|\mathcal{F}|$ denota la cardinalidad del subconjunto $\mathcal{F}$. Los símbolos $\mathcal{F}$ se definen de la siguiente manera:

Definición 9 (Símbolos fijos).

Los símbolos fijo ($\mathcal{F}$) son un subconjunto del alfabeto, los cuales van desde $\{0,\ldots,|\mathcal{F}|-1\}$ y no son afectados por la *Traslación*.

Al utilizar los símbolos fijos, la operación de *Traslación* y esto implica $0 \leq \delta \leq v - |\mathcal{F}|$, y por cada *Traslación* se aplica la operación de *Rotación* k veces ($0 \leq \sigma \leq k-1$), dando como resultado una matriz de tamaño $(v-|\mathcal{F}|)k \times k$. Para que la matriz resultante sea un CA se le yuxtapone un CA con $v = |\mathcal{F}|$ para cubrir todas las combinaciones. En la Figura 1.14 se muestra un ejemplo de como se construye un CA utilizando $|\mathcal{F}|=1$, $k=5$, $v=3$, $t=2$ y el vector $\{0,1,1,1,2\}$.

Para que la matriz resultante de las operaciones de *Rotación* y *Traslación* sea un CA, dividen en grupos las combinaciones de las posiciones del VI. Un grupo se define de la siguiente manera:

$$\left(\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & 1 & 2 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 & 1 \\ 1 & 1 & 2 & 0 & 1 \\ 1 & 1 & 1 & 2 & 0 \\ \hline 1 & 2 & 2 & 2 & 0 \\ 0 & 1 & 2 & 2 & 2 \\ 2 & 0 & 1 & 2 & 2 \\ 2 & 2 & 0 & 1 & 2 \\ 2 & 2 & 2 & 0 & 1 \end{array}\right)$$

Figura 1.14: $CA(11; 2, 5, 3)$, CA construido con $\mathcal{F} = 1$, $k = 5$, $v = 3$, $t = 2$ y el vector $\{0, 1, 1, 1, 2\}$

Definición 10 (Grupo).

Un grupo $\mathcal{G}$ es una pareja $(\mathbb{C}, \tau)$, donde $\mathbb{C}$ es un conjunto no vacío y τ es una operación binaria tal que $\mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C}$.

La manera en que se agrupan las combinaciones de las posiciones del VI se realiza de tal manera que el grupo sea un grupo cíclico. Un grupo cíclico se define de la siguiente manera:

Definición 11 (Grupo Cíclico).

Un grupo $\mathbb{G}$ es cíclico si existe $g \in \mathbb{G}$ tal que $g^n \in \mathbb{G}$, donde n es un número entero.

La característica del grupo cíclico es que, al elevar a alguna potencia cualquier elemento del grupo, se pueden obtener los demás elementos del grupo. Puesto que g es un vector, al elevar el vector a una potencia se realiza un desplazamiento módulo k. Tomando un elemento de todos los grupos cíclicos del VI se verifica si la matriz resultante es un CA. A este conjunto de elementos se les llama órbitas. Una órbita se define de la siguiente manera:

$$\left(\begin{array}{ccccc}
0 & 0 & 0 & \mathbf{0} & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{0} \\
\mathbf{0} & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{0} & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{0} & \mathbf{1} & 0 \\
\hline
1 & 1 & 1 & \mathbf{1} & \mathbf{2} \\
\mathbf{2} & 1 & 1 & 1 & \mathbf{1} \\
\mathbf{1} & \mathbf{2} & 1 & 1 & 1 \\
1 & \mathbf{1} & \mathbf{2} & 1 & 1 \\
1 & 1 & \mathbf{1} & \mathbf{2} & 1 \\
\hline
2 & 2 & 2 & \mathbf{2} & \mathbf{0} \\
\mathbf{0} & 2 & 2 & 2 & \mathbf{2} \\
\mathbf{2} & \mathbf{0} & 2 & 2 & 2 \\
2 & \mathbf{2} & \mathbf{0} & 2 & 2 \\
2 & 2 & \mathbf{2} & \mathbf{0} & 2
\end{array}\right)$$

Figura 1.15: $CA(15; 2, 5, 3)$. CA construido a través de las operaciones de *Rotación* y *Traslación* al vector $\{0, 0, 0, 0, 1\}$, en el cual, se resaltan las tuplas $(0, 1)$ del primer bloque, del segundo bloque la tupla $(1, 2)$ y la tupla $(2, 0)$ del tercer bloque las cuales se obtienen usando la operación de *Traslación* y se muestra como se van desplazando las tuplas a través de la operación de *Rotación*.

Definición 12 (Órbita).

Una órbita g es un vector de tamaño t el cual es el elemento $g \in \mathbb{G}$ donde $\mathbb{G}$ es un grupo cíclico. [36]

Las órbitas aseguran que las tuplas que se obtienen de la *Traslación* están cubiertas en todas las órbitas de esa misma distancia a través de las operación de *Rotación*. Para ejemplificar lo anterior si tenemos los valores $k = 5$, $v = 3$ y $t = 2$ y el vector $\{0, 0, 0, 0, 1\}$ y tomamos la tupla $(0, 1)$ de las posiciones 3,4, al aplicar la operación de *Rotación* cinco veces la tupla $(0, 1)$ se encuentra en las columnas 3,4 del primer renglón, en el segundo renglón la tupla se encuentra en las columnas 4,0, en el tercer renglón la tupla se encuentra en las columnas 0,1, el cuarto renglón contiene la tupla en las columnas 1,2 y el quinto renglón la tupla se encuentra en las columnas 2,3 y las tuplas obtenidas por

la operación de *Traslación* de los siguientes bloques se encuentran en las mismas posiciones (Figura 1.15).

Colbourn [35] construye el VI de tamaño k a través de los campos finitos de Galois, destacando que el valor de k debe de ser una potencia de un número primo. A partir del VI se pueden construir dos instancias de CAs.

- $CA(k; t, k, v)$. En esta instancia se construye un CA con k renglones aplicando la operación de *Rotación* al VI k veces, si en los k renglones se cubren todas las t-tuplas en todas las submatrices de tamaño $k \times t$ se construye este caso.

- $CA(vk; t, k, v)$. En esta instancia se construye un CA con vk renglones aplicando la operación de *Traslación* v veces y por cada *Traslación* se aplica la operación de *Rotación* k veces, si en los vk renglones se cubren todas las t-tuplas en todas las submatrices de tamaño $k \times t$ se puede construir este caso.

Tabla 1.6: $CA(7; 2, 7, 2)$. CA construido utilizando el $VI = \{0, 0, 0, 1, 0, 1, 1\}$.

VI	0	0	0	1	0	1	1
	0	0	0	1	0	1	1
	0	0	1	0	1	1	0
	0	1	0	1	1	0	0
Rotaciones	1	0	1	1	0	0	0
	0	1	1	0	0	0	1
	1	1	0	0	0	1	0
	1	0	0	0	1	0	1

En la Tabla 1.6 se muestra un CA de $k = 7$, $v = 2$ y $t = 3$ construido utilizando el vector $\{0, 0, 0, 1, 0, 1, 1\}$ de 7 renglones siguiendo el primer caso de la construcción de Colbourn.

1.7 Objetivo General

Generalizar la construcción de CAs a través del uso de VIs para $t = 2$ hasta $t = 6$.

1.8 Objetivos Específicos

a) Construir CAs con igual o menor cardinalidad de renglones que los mejores reportados en el estado del arte utilizando la generalización propuesta.

b) Desarrollar un algoritmo exacto para la construcción del VI que iguale o mejore algunas cotas de los CAs reportados en el estado del arte.

1.9 Metodología de Investigación

Para resolver el problema se seguirán los siguientes pasos:

- Se realizará una revisión del estado del arte.
- Se diseñará e implementará un algoritmo para verificar que los vectores son VIs utilizando las órbitas.
- Utilizando el algoritmo de verificación, se diseñará e implementará un algoritmo de búsqueda exacto para construir los VIs.
- Se realizará una experimentación computacional y se compararán los resultados obtenidos con los mejores reportados en el estado del arte.

1.10 Organización del Documento de Tesis

El Documento de tesis está estructurado en cinco capítulos; a continuación se mencionará de manera breve el contenido de los siguientes cuatro capítulos:

- El Capítulo 2 contendrá de manera breve algunos de los métodos mas importantes para la construcción de CAs, clasificándolos en métodos algebraicos, métodos exactos, métodos voraces, métodos heurísticos y métodos de manipulación.

- El Capítulo 3 describe la metodología propuesta detallando cada módulo desarrollado para construir los VIs.

- El Capítulo 4 mostrará los resultados obtenidos utilizando la metodología propuesta y se compararán con las mejores cotas reportadas.

- El Capítulo 5 describirá las conclusiones que se obtuvieron al realizar este trabajo de tesis y el trabajo futuro que se deriva de este trabajo de tesis.

1.11 Resumen del Capítulo

En este capítulo se presentó la importancia de las pruebas de software y la manera de diseñar los casos de pruebas clasificándolas en pruebas de caja blanca y pruebas de caja negra, detallando las técnicas existentes para cada tipo de prueba de software. Para las pruebas de caja blanca se mostraron las técnicas de pruebas del camino básico y pruebas de control de estructura. De cada tipo de prueba se explicaron las técnicas con las que se puede realizar dicha prueba. Para las pruebas de caja negra se detallaron las técnicas de pruebas basados en grafos, partición equivalente, análisis de valores límite y las pruebas de *Orthogonal Array*.

Se profundizó en las pruebas de *Orthogonal Array* mostrando una opción para crear estas pruebas a través de los objetos combinatorios, definiendo de manera formal los objetos combinatorios: OA, CA y MCA. De estos objetos combinatorios se explicaron las diferencias entre los OA y CA y las diferencias entre los CA y MCA. Se definió lo que es un elemento comodín y cómo es que se determina sí un elemento es un elemento comodín. También se habló del problema que se resuelve en esta tesis a través del uso de un VI que permite construir un CA aplicando las operaciones de *Rotación* y *Traslación* al VI y como se utilizan las órbitas para verificar el grado de cobertura del VI, se explicaron los trabajos que usan un VI para construir un CA. Se mencionaron los objetivos que se tienen para este trabajo de tesis y la metodología de investigación que se seguirá.

Dentro de los objetos combinatorios se explicó el problema de $CACP$ y se mencionaron las clasificaciones de los métodos existentes para construir CAs. El siguiente capítulo habla de estos métodos clasificándolos en: métodos algebraicos, métodos exactos, métodos voraces, métodos heurísticos y manipulación de CAs.

2

Estado del arte

En el capítulo anterior se habló de la importancia de las pruebas de software, junto con algunas técnicas para diseñar los casos de prueba y se profundizó en la técnica de pruebas de *Orthogonal Array*. Se explicó una manera para crear estos casos de prueba utilizando objetos combinatorios. De los diferentes objetos combinatorios se habló del CA y los métodos existentes para construir CAs. Puesto que existe una gran cantidad de métodos para construir CAs, éstos se pueden clasificar en métodos algebraicos, métodos exactos, métodos voraces, métodos heurísticos y métodos de manipulación. En este capítulo se presenta una breve revisión del estado del arte sobre algunos de estos métodos para construir CAs de acuerdo a las clasificaciones previamente mencionadas. El capítulo se divide en siete secciones, en la primer sección se mencionarán algunos de los métodos algebraicos existentes para construir CAs. En la segunda sección estarán algunos de los métodos de construcción exactos, la tercera sección contendrá los métodos voraces, la cuarta sección contendrá los métodos heurísticos y en la quinta sección se mostrarán algunos de los métodos de manipulación. Al final se presenta un resumen del capítulo.

2.1 Métodos Algebraicos

Los métodos de construcción algebraicos siguen un proceso para construir CAs utilizando fórmulas matemáticas y generalmente requieren de otros objetos matemáticos como vectores, campos finitos de Galois u otros CAs.

2.1.1 Construcción de Bush

En 1952 Bush [8] presenta una construcción de OA con $\lambda = 1$, $OA_1(v^t; v+1, v, t)$ utilizando los Campos Finitos de Galois (GF), cuando v es una potencia de un número primo ($v = p^\alpha$) donde p es un número primo y $\alpha \geq 1$. Los OA de $\lambda = 1$ son equivalentes a CA óptimos, $CA(p^{\alpha t}; t, p^\alpha + 1, p^\alpha)$.

Los GF (p^α) son un conjunto de números finitos módulo p. Cada elemento del GF (p^α) es representado por un polinomio de grado $\alpha - 1$. Cuando $\alpha = 1$, los elementos del GF son polinomios de un término, el cual es un entero módulo p. Por ejemplo, los elementos del GF (3) son $\{0, 1, 2\}$. Cuando $\alpha > 1$ los elementos del GF se definen de acuerdo a la siguiente fórmula:

$$P_j(x) = \sum_{i=0}^{\alpha-1} a_i x^i \quad \text{para} \quad j = 0, \ldots, v \tag{2.1}$$

Con los elementos de GF (p^α) se pueden realizar operaciones como la suma o la multiplicación [38]. La suma de dos elementos del GF (p^α) se realiza haciendo una suma modulo p. La multiplicación entre dos elementos del GF (p^α) puede resultar en un polinomio de grado mayor a $\alpha - 1$, cuando ocurre esto, se requiere de un polinomio primitivo del GF (p^α) de grado menor o igual a $\alpha - 1$ para reducir los términos que sean de mayor grado a $\alpha - 1$. Una opción para realizar la multiplicación es a través de las tablas de logaritmos y antilogaritmos [39], en esta opción también se requiere de un elemento primitivo para crear las tablas. El problema de cualquiera de los dos métodos es encontrar el elemento primitivo para cualquier valor de p y α.

Torres-Jiménez *et al.* [39] presentaron en el 2010 un algoritmo para encontrar el elemento primitivo de menor grado y a su vez construir las tablas de logaritmos y antilogaritmos de un GF. La manera en que encuentran el elemento primitivo de menor grado, es mediante la construcción de las tablas de logaritmos y antilogaritmos probando cada elemento de los GF (v).

$$\sum_{j=0}^{t-1} \delta_j X^j \tag{2.2}$$

Una vez construidos los GF (v) con la tabla de logaritmos, la construcción de Bush se describe de la siguiente manera: se crea una matriz $\mathbb{D}$ de tamaño $v^t \times (v+1)$, donde los renglones se etiquetan con los v^t polinomios diferentes $\ell(X) = \delta_{t-1}X^{t-1} + \ldots + \delta_1 X + \delta_0$ donde $(\delta_0, \delta_1, \ldots \delta_{t-1}) \in v^t$ y las primeras v columnas se etiquetan con cada elemento de $\delta \in GF(p)$ y la última columna se etiqueta con ∞. La matriz se llena de acuerdo al siguiente criterio: a las columnas que están etiquetadas con los valores $0, \ldots, p-1$, se les asigna el valor obtenido de acuerdo a la ecuación (2.2) y a la columna con la etiqueta ∞ se le asigna el coeficiente δ_{t-1}.

Tabla 2.1: Tabla de logaritmos y antilogaritmos construida con el elemento primitivo 2 para GF(5).

GF(5)	Logaritmos	Antilogaritmos
1	4	2
2	1	4
3	3	3
4	2	1

Para ejemplificar al construcción anterior, lo haremos con un $CA(25; 2, 6, 5)$ con los valores $p = 5$, $\alpha = 1$, y $t = 2$, primero se construye una matriz $\mathbb{D}$ de tamaño $25 \times (5+1)$, luego se crea la matriz de logaritmos y antilogaritmos con el elemento primitivo 2 (Tabla 2.1). Una vez que se tiene la tabla

de logaritmos y antilogaritmos, a la matriz $\mathbb{D}$, se le etiqueta cada renglón con los polinomios desde el polinomio 0 hasta el polinomio $4x + 4$ y se etiquetan las primeras v columnas con los elementos del GF(5) $= \{0, 1, 2, 3, 4\}$ y la última columna de denota por ∞, al utilizar la tabla de logaritmos y antilogaritmos, se construye la matriz de 25×5, para la última columna, se toma el primer coeficiente del polinomio de la etiqueta de cada renglón y se pone en la columna ∞, dando como resultado un $CA(25; 2, 6, 5)$ (Tabla 2.2).

Tabla 2.2: $CA(25; 2, 6, 5)$, CA construido a través del método de Bush. En la primer columna se muestra la etiqueta de los renglones y en el primer renglón se muestra la etiqueta de las columnas.

Etiquetas			0	1	2	3	4	∞
		0	0	0	0	0	0	0
		1	1	1	1	1	1	0
		2	2	2	2	2	2	0
		3	3	3	3	3	3	0
		4	4	4	4	4	4	0
x			0	1	2	3	4	1
x	$+$	1	1	2	3	4	0	1
x	$+$	2	2	3	4	0	1	1
x	$+$	3	3	4	0	1	2	1
x	$+$	4	4	0	1	2	3	1
$2x$			0	2	4	1	3	2
$2x$	$+$	1	1	3	0	2	4	2
$2x$	$+$	2	2	4	1	3	0	2
$2x$	$+$	3	3	0	2	4	1	2
$2x$	$+$	4	4	1	3	0	2	2
$3x$			0	3	1	4	2	3
$3x$	$+$	1	1	4	2	0	3	3
$3x$	$+$	2	2	0	3	1	4	3
$3x$	$+$	3	3	1	4	2	0	3
$3x$	$+$	4	4	2	0	3	1	3
$4x$			0	4	3	2	1	4
$4x$	$+$	1	1	0	4	3	2	4
$4x$	$+$	2	2	1	0	4	3	4
$4x$	$+$	3	3	2	1	0	4	4
$4x$	$+$	4	4	3	2	1	0	4

En esta construcción Bush también describe un caso especial, cuando $v = 2^{\alpha}$ y $t = 3$, en este caso se le añade una columna más y a esta columna se le agrega el valor de δ_1.

2.1.2 Construcción de CA de Fuerza dos y Alfabeto Binario

En 1993 Sloane [30] describe como fue solucionado el caso para CA con $t = 2$ y $v = 2$ $CA(N; 2, k, 2)$. Él describe que primero fue solucionado por Rényi [40] cuando la k es par, el problema para cualquier k fue resuelto por Rényi [40] e independientemente por Katona [28] y Kleitman and Spencer [29].

La construcción empieza determinando el valor de k para un valor de N dado, tal que satisfaga la ecuación (2.3).

$$k \leq \binom{N-1}{\lceil \frac{N}{2} \rceil} \tag{2.3}$$

Una vez que se determina el valor de k para el valor de N dado, se crea una matriz de N renglones y k columnas, llenando el primer renglón con ceros. La matriz restante de tamaño $(N-1) \times k$ se construye con las $\binom{N-1}{\lceil \frac{N}{2} \rceil}$ combinaciones posibles con $\left\lceil \frac{N}{2} \right\rceil$ unos y $(N-1) - \left\lceil \frac{N}{2} \right\rceil$ ceros, donde cada combinación es una columna de la matriz.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Figura 2.1: $CA(6; 2, 10, 2)$, CA de $v = 2$ y $t = 2$ utilizando la construcción de Rényi.

Ejemplificando lo anterior, se construirá un CA con $N = 6$, de acuerdo a la ecuación (2.3) el valor de k es 10, se construye la matriz de tamaño 6×10, se llena el primer renglón con 10 ceros.

La matriz restante de tamaño 5×10 se llena con las diez combinaciones de columnas con tres 1 y dos 0. Construyéndose así el $CA(6; 2, 10, 2)$, la Figura 2.1 muestra el CA resultante.

2.1.3 Vectores de Pesos Constantes

En 1983, Tang y Woo [41] utilizaron vectores de pesos contantes para diseñar conjuntos de prueba exhaustivas para aplicar las pruebas en circuitos. Estos conjuntos de pruebas son equivalentes a un CA. Ellos definen un vector de peso constante de la siguiente manera:

Definición 13 (Vectores de Pesos Constantes).

Sea un vector ϕ de tamaño k con elementos del conjunto $\mathbb{Z}_v$, donde un elemento en particular se designa por ϕ_i, $0 \leq i \leq k-1$, el peso de ϕ denotado por w, es la suma de todos los ϕ_i. El peso del vector es representado por

$$\omega = \sum_{i=0}^{k-1} \phi_i$$

Ellos derivaron que para los valores $k > t$ y $v > 2$, se requiere un conjunto de vectores $\mathbf{T}$ que contenga todos los vectores de tamaño k, con v elementos y con peso ω tal que $w \equiv c \bmod s$, para una constante c, donde $s = (k-t)(v-1)+1$, $0 \leq c \leq k-t$ y $0 \leq \omega \leq k(v-1)$.

De esta manera, existen un total de $k - t + 1$ soluciones. Entonces se debe de buscar cuál de esas soluciones obtiene del conjunto de vectores con menor cardinalidad, es decir, el menor tamaño de N.

Para ejemplificar la construcción anterior, se tienen los valores de $k = 4$, $v = 3$ y $t = 2$; se obtiene el valor de s de acuerdo a $s = (k-t)(v-1)+1 = 5$, cuando $c = 1$, ω toma valores en $\{1, 6\}$ por $\omega \equiv 1 \bmod 5$. Dando como resultado un $CA(14; 2, 4, 3)$ (Tabla 2.3).

Tabla 2.3: $CA(14;2,4,3)$, CA formado por los conjuntos de vectores de peso 1 y 6.

Peso	Vectores			
1	1	0	0	0
	0	1	0	0
	0	0	1	0
	0	0	0	1
6	0	2	2	2
	2	0	2	2
	2	2	0	2
	2	2	2	0
	2	2	1	1
	2	1	2	1
	1	2	2	1
	2	1	1	2
	1	2	1	2
	1	1	2	2

2.1.4 Construcción a Través de Coeficientes Trinomiales

En 2010, Martinez-Peña *et al.* [42, 43] propusieron una construcción para CA de alfabeto tres $CA(N;t,k,3)$ a través de coeficientes trinomiales. Los coeficientes trinomiales los definen de la siguiente manera:

Definición 14 (Coeficientes Trinomiales).

Dado a,b,c satisfaciendo $(0 \leq a,b,c \leq k)$ y $(a+b+c=k)$, un coeficiente trinomial (CT), denotado por $\binom{k}{a,b,c}$, es el coeficiente obtenido por la ecuación

$$\binom{k}{a,b,c} = \frac{(a+b+c)!}{a!b!c!} \tag{2.4}$$

Los CTs representan un subconjunto particular de renglones que puede pertenecer a un CA de $v=3$. Para $0 \leq a,b,c \leq k$, $a+b+c=k$ y $k \geq 2$, un subconjunto $\mathbb{R}^k_{a,b,c}$ es una colección de renglones obtenida por el CT $\binom{k}{a,b,c}$ y su cardinalidad es igual a dicho coeficiente. El subconjunto

de renglones es generado por la evaluación de todas las combinaciones usando $0^a1^b2^c$ símbolos, es decir, el símbolo 0 es usado a veces, el símbolo 1 es usado b veces, el símbolo 2 es usado c veces sobre un renglón de tamaño k.

La construcción de un CA ternario denotado por $\mathbb{A}$ consta de tres pasos.

1. Se generan todos los CTs de orden k, los CTs representan un subconjunto de renglones $\mathbb{T}$.

2. A través de un algoritmo de *backtracking* que implementa una técnica de ramificación y poda se recorre el espacio de búsqueda de manera más eficiente, se busca el conjunto de CTs que construyan un CA con el menor número de renglones.

3. Transforma los CTs encontrados en el paso anterior en un CA.

Para ejemplificar la construcción anterior, si se quiere construir un CA de $k = 3$, $CA(N; 2, 3, 3)$, primero se generan todos los CTs, $\binom{k}{3,0,0}$, $\binom{k}{2,1,0}$, $\binom{k}{1,2,0}$, $\binom{k}{0,3,0}$, $\binom{k}{2,0,1}$, $\binom{k}{1,1,1}$, $\binom{k}{0,2,1}$, $\binom{k}{1,0,2}$, $\binom{k}{0,1,2}$ y $\binom{k}{0,0,3}$. En el segundo paso, se seleccionan los CTs que construyan un CA con el menor número de renglones, los cuales son $\binom{k}{3,0,0}$, $\binom{k}{0,3,0}$, $\binom{k}{0,0,3}$ y $\binom{k}{1,1,1}$ (Tabla 2.4). En el tercer paso, se transforman los CTs y se construye el CA (Figura 2.2).

2.1.5 Producto de CAs de Fuerza dos

En el 2006, Colbourn *et al.* [44] muestran un método de construcción por bloques, multiplicando dos CAs de fuerza dos. La multiplicación se realiza teniendo a $A = (a_{i,j})$ como $CA(N; 2, k, v)$ y $B = (b_{i,j})$ como $CA(M; 2, l, v)$. El producto de $A \otimes B$ es una matriz $C = (c_{i,j})$ de tamaño $(N + M) \times kl$. Asignando a $c_{i,(f-1)k+g}$ el valor de $a_{i,g}$ para $1 \leq i \leq N$, $1 \leq f \leq l$ y $1 \leq g \leq k$ y a $c_{N+i,(f-1)k+g}$ el valor de $b_{i,f}$ para $1 \leq i \leq M$, $1 \leq f \leq l$ y $1 \leq g \leq k$. En esencia se hacen k

Tabla 2.4: CTs seleccionados y los bloques de renglones que representan.

CT	Renglones
$\binom{k}{3,0,0}$	0 0 0
$\binom{k}{0,3,0}$	1 1 1
$\binom{k}{0,0,3}$	2 2 2
$\binom{k}{1,1,1}$	0 1 2 1 2 0 2 0 1 0 2 1 1 0 2 2 1 0

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \\ 2 & 1 & 0 \end{pmatrix}$$

Figura 2.2: $CA(9;2,3,3)$ construido con los CTs.

copias de $B = (b_{i,j})$ y son agregadas a las l copias de $A = (a_{ij})$ (Figura 2.3). Por lo tanto, cada dos columnas de C surgen de dos diferentes columnas de A o de dos diferentes columnas de B, dando como resultado un $CA(N+M;2,kl,v)$.

a_{11}	a_{12}	$\dots$	a_{1k}	a_{11}	a_{12}	$\dots$	a_{1k}	$\dots$	a_{11}	a_{12}	$\dots$	a_{1k}
a_{11}	a_{12}	$\dots$	a_{1k}	a_{11}	a_{12}	$\dots$	a_{1k}	$\dots$	a_{11}	a_{12}	$\dots$	a_{1k}
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
a_{N1}	a_{N2}	$\dots$	a_{Nk}	a_{N1}	a_{N2}	$\dots$	a_{Nk}	$\dots$	a_{N1}	a_{N2}	$\dots$	a_{Nk}
b_{11}	b_{11}	$\dots$	b_{11}	b_{12}	b_{12}	$\dots$	b_{12}	$\dots$	b_{1l}	b_{1l}	$\dots$	b_{1l}
b_{21}	b_{21}	$\dots$	b_{21}	b_{22}	b_{22}	$\dots$	b_{22}	$\dots$	b_{2l}	b_{2l}	$\dots$	b_{2l}
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
b_{M1}	b_{M1}	$\dots$	b_{M1}	b_{M2}	b_{M2}	$\dots$	b_{M2}	$\dots$	b_{Ml}	b_{Ml}	$\dots$	b_{Ml}

Figura 2.3: Matriz resultante de tamaño $(N+M) \times kl$ del producto de los $CA(N;2,k,v)$ y $CA(M;2,l,v)$.

Esta construcción se puede extender a los $MCAs$ tomando en cuenta que al estar agregando las copias se puede dar el caso de que el alfabeto v_i $1 \leq i \leq k$ del MCA $(N;2,k,(v_1,\dots,v_k))$ A sea mayor que el alfabeto w_j $1 \leq j \leq l$ del $MCA(M;2,l,(w_1,\dots,w_l))$ B, $v_i > w_j$, si esto pasa, en cada ocurrencia de $v_i - w_j$, los elementos se cambian por el valor de w_j.

TEOREMA 1.

Suponemos que existe

1. Un $MCA(N; 2, k, (v_1, \ldots, v_k))$, A, con un perfil de comodines $(d_1, \ldots, d_k)$.
2. Por cada $1 \leq i \leq k$, un $MCA(M_i; 2, l_i, (w_{i1}, \ldots, w_{il}))$, B_i, con un perfil de comodines $(f_{ij}, \ldots, f_{il})$ y para el cual $w_{ij} < v_i$ para $1 \leq j \leq l_i$

Entonces para $T = N + max_{i=1}^{k}(M_i - d_i)$ existe un $MCA(T; 2, kl, (w_{11}, \ldots, w_{1l_k}, \ldots, w_{k1}, \ldots, w_{kl_k}))$

Si la matriz A y la matriz B tienen renglones constantes, al realizar la multiplicación de las dos matrices, la matriz C contendrá renglones constantes redundantes. Un renglón de un MCA es constante si y solo si los pares (x, x) se encuentran cubiertos en dicho renglón. Los renglones constantes redundantes de C se pueden eliminar aplicando el TEOREMA 1, de esta manera, en algunas instancias, el número de renglones se reduce. Moura *et al.* [45] explotan los renglones constantes para reducir el tamaño del MCA resultante.

Desafortunadamente, en algunos casos, se deben sacrificar algunas columnas para obtener renglones constantes. Si tenemos un $CA(N; 2, k_1 + k_2, v)$ y tiene una partición $\mathcal{A}_1$ de tamaño $N \times k_1$, una partición $\mathcal{A}_2$ de tamaño $N \times k_2$, una partición $\mathcal{X}$ de tamaño $v \times k_2$ y una partición $\mathcal{P}$ de tamaño $v \times k_1$ con la condición de que todas las columnas de $\mathcal{P}$ sean permutaciones de v, entonces el CA es un *Partitioned Covering Array* denotado por $PCA(N; 2, (k_1, k_2), v)$.

2.1.6 Construcción de CA Utilizando Construcciones Tipo Roux

En el 2006 Colbourn *et al.* [46] mostraron una construcción recursiva para fuerzas tres y cuatro generalizando varias construcciones de tipo Roux. G. Roux en su tesis doctoral [47] estableció que

se puede construir un $CAN(3, 2k, 2)$ a partir de $CAN(3, k, 2)$ y $CAN(2, k, 2)$ ($CAN(3, 2k, 2) \leq CAN(3, k, 2)+CAN(2, k, 2)$). Teniendo los ingredientes $A = CA(N; 3, k, 2)$ y $B = CA(M; 2, k, 2)$ es posible construir un CA denotado por C de $N+M$ renglones y $2k$ columnas $CA(N+M; 3, 2k, 2)$, teniendo $\bar{B}$ como la matriz complementaria de B, la matriz C se construye mediante:

$$C = \begin{pmatrix} AA \\ B\bar{B} \end{pmatrix}$$

Para ejemplificar lo anterior, si queremos construir un CA a partir de los $CA(5; 2, 4, 2)$ (Figura 2.4) como la matriz A y $CA(8; 3, 4, 2)$ (Figura 2.5(a)) como la matriz B. Copiamos dos veces la matriz A y ponemos la matriz B abajo de la primer copia de la matriz A y la matriz $\bar{B}$ (Figura 2.5(b)) abajo de la segunda copia de A. Construyendo el $CA(13; 3, 8, 2)$ (Figura 2.6).

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Figura 2.4: CA utilizado como primer ingrediente para la construcción tipo Roux $CA(8; 3, 4, 2)$.

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

(a) $CA(5; 2, 4, 2)$.

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

(b) CA complemento $CA(5; 2, 4, 2)$.

Figura 2.5: CA utilizado como el segundo ingrediente para la construcción tipo Roux.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Figura 2.6: $CA(13; 3, 8, 2)$ resultante de la construcción tipo Roux.

2.1.7 Potenciación de CAs

En el 2005 Hartman [12] presento un método para aumentar el número de columnas al cuadrado k^2 de un $CA(N; t, k, v)$. El proceso requiere de un $CA(N; t, k, v)$ como base llamado A y un OA de $\lambda = 1$ con k^2 renglones, $T(v, t) + 1$ columnas, $t = 2$ y $v = k$, $OA_1(k^2; 2, T(v, t) + 1, k)$ llamado B. Cada elemento del OA es utilizado como índice de una columna de A. $T(v, t)$ es el número de Turán de v y t y estos denotan el número máximo de arcos en un grafo v-partita con t nodos.

El procedimiento de potencia consiste en crear una matriz C de tamaño $k^2 \times T(v, t) + 1$. Cada elemento de C contiene una columna de A. $B[i, j]$ es la entrada (i, j) de B y A_i es la i-ésima columna de A, donde $1 \leq i \leq k$. La celda (i, j) de C es la columna $B[j, i]$ de A, $C[i, j] = A_{B[j,i]}$. El resultado es un $CA(N(T(v, t) + 1); t, k^2, v)$. La Figura 2.7 muestra este proceso. Este método es una generalización del método propuesto por Tang y Chen [48] en 1984.

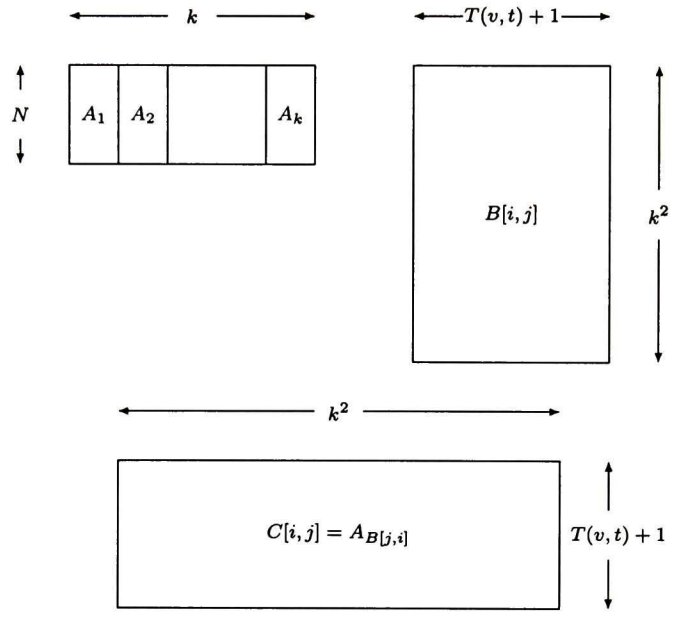


Figura 2.7: Proceso para elevar al cuadrado el número de columnas de un CA.

2.1.8 Construcción por Grupos

En 1999 Chateauneuf *et al.* [31] mostraron un método para construir CAs de $t = 3$, en el 2005 Stevens y Meagher [33] adaptaron el método para construir CAs de $t = 2$..

La construcción de Stevens y Meagher consiste en seleccionar un grupo $G < Sym_v$ y un vector $w \in \mathbb{Z}_v^k$ llamado Vector Inicializador (VI). El VI se debe de buscar dependiendo del grupo G. Un vector $w \in \mathbb{Z}_v^k$ es un VI sí todos los conjuntos de d_i, $1 \leq i \leq k-1$, donde d_i se define como $d_i = \{(w_j, w_{j+1}) \,|\, j = 0, 1, \ldots, k-1\}$ con el subíndice es módulo k, contienen al menos un elemento de cada órbita del grupo de acción G de los pares de $\mathbb{Z}_v$.

Una vez que se tiene el VI denotado por w, se construye una matriz $\mathcal{M}$ de tamaño $k \times k$ yuxtaponiendo las rotaciones del vector w. Al actuar el grupo G sobre la matriz $\mathcal{M}$ se producen $v-1$ matrices de tamaño $k \times k$. Éstas matrices se deben yuxtaponer junto con un vector C donde

cada $\mathcal{C}_i = 0$ tal que $0 \leq i \leq k-1$ se forma el CA. El vector $\mathcal{C}$ asegura que todas las tuplas $(0,0)$ están cubiertas.

Ejemplificando la construcción anterior suponiendo que tenemos los valores de $v = 3$, $k = 5$ y $t = 2$, el grupo $G = \{\kappa, (1,2)\}$ y el VI $w = \{0,1,1,1,2\}$, con el vector w se construye la siguiente matriz:

$$\mathcal{M} = \begin{pmatrix} 0 & 1 & 1 & 1 & 2 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 & 1 \\ 1 & 1 & 2 & 0 & 1 \\ 1 & 1 & 1 & 2 & 0 \end{pmatrix}$$

Los elementos de G actuando sobre la matriz $\mathcal{M}$ producen las matrices:

$$\mathcal{M}_\kappa = \begin{pmatrix} 0 & 1 & 1 & 1 & 2 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 & 1 \\ 1 & 1 & 2 & 0 & 1 \\ 1 & 1 & 1 & 2 & 0 \end{pmatrix} \quad \text{y} \quad \mathcal{M}_{(1,2)} = \begin{pmatrix} 0 & 2 & 2 & 2 & 1 \\ 1 & 0 & 2 & 2 & 2 \\ 2 & 1 & 0 & 2 & 2 \\ 2 & 2 & 1 & 0 & 2 \\ 2 & 2 & 2 & 1 & 0 \end{pmatrix}$$

Donde el elemento κ es el elemento identidad, por lo tanto $\mathcal{M}_\kappa = \mathcal{M}$, la matriz $\mathcal{M}_{(1,2)}$ se obtiene intercambiando los símbolos 1 y 2 de la matriz $\mathcal{M}$.

Al concatenar las matrices $\mathcal{M}_\kappa$, $\mathcal{M}_{(1,2)}$ y el vector $\mathcal{C}$ se construye un CA de 11 renglones $CA(11;2,5,3)$ (Figura 2.8).

$$\left(\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & 1 & 2 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 & 1 \\ 1 & 1 & 2 & 0 & 1 \\ 1 & 1 & 1 & 2 & 0 \\ \hline 1 & 2 & 2 & 2 & 0 \\ 0 & 1 & 2 & 2 & 2 \\ 2 & 0 & 1 & 2 & 2 \\ 2 & 2 & 0 & 1 & 2 \\ 2 & 2 & 2 & 0 & 1 \end{array}\right)$$

Figura 2.8: $CA(11;\ 2,\ 5,\ 3)$, CA construido utilizando el $VI = \{0,1,1,1,2\}$, el grupo $G = \{\delta, (1,2)\}$ y el vector $\mathcal{C}$.

En el 2012 Lobb *et al.* [37] presentaron una generalización de este método en el cual se maneja más de un símbolo fijo[1]. Al conjunto de símbolos fijos los denotan por $\mathcal{F}$, donde $|\mathcal{F}|$ es la cardinalidad de los símbolos fijos.

Al yuxtaponer las matrices resultantes se construye una matriz $\mathcal{T}$ de tamaño $k(v - |\mathcal{F}|) \times k$ y a la matriz $\mathcal{T}$ se le debe de yuxtaponer otra matriz llamada $\mathcal{Z}$. La matriz $\mathcal{Z}$ debe de cubrir todas las combinaciones de $t = 2$ con $v = |\mathcal{F}|$, por lo tanto, la matriz $\mathcal{Z}$ debe de ser un CA con $N = CAN(k, t, |\mathcal{F}|)$. El CA resultantes es el siguiente $CA(k(v - |\mathcal{F}|) + CAN(k, t, |\mathcal{F}|); t, k, v)$.

$$\left(\begin{array}{ccccccccc} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Figura 2.9: $CA(6; 2, 9, 2)$, CA utilizado para yuxtaponerlo con la matriz $\mathcal{T}$.

[1]No son afectados por el grupo de acción

Para ejemplificar la construcción de Lobb *et al.* suponemos que tenemos los valores de $k = 9$, $v = 4$ y $t = 2$, el $VI = \{0, 1, 3, 2, 3, 3, 1, 0, 2\}$ y $G = \{\delta, (2, 3)\}$, al yuxtaponer las dos matrices resultantes nos da como resultado la matriz $\mathcal{T}$ de la Figura 2.10. Para construir el CA hay que agregar un CA con $v = 2$, el CA que se agrega se muestra en la Figura 2.9, el cual tiene seis renglones $CA(6; 2, 9, 2)$, al yuxtaponer el CA de seis renglones a la matriz $\mathcal{T}$ se construye un CA de 24 renglones $CA(24; 4, 9, 2)$.

$$\mathcal{T} = \left(\begin{array}{cccccccccc}
0 & 1 & 3 & 2 & 3 & 3 & 1 & 0 & 2 \\
2 & 0 & 1 & 3 & 2 & 3 & 3 & 1 & 0 \\
0 & 2 & 0 & 1 & 3 & 2 & 3 & 3 & 1 \\
1 & 0 & 2 & 0 & 1 & 3 & 2 & 3 & 3 \\
3 & 1 & 0 & 2 & 0 & 1 & 3 & 2 & 3 \\
3 & 3 & 1 & 0 & 2 & 0 & 1 & 3 & 2 \\
2 & 3 & 3 & 1 & 0 & 2 & 0 & 1 & 3 \\
3 & 2 & 3 & 3 & 1 & 0 & 2 & 0 & 1 \\
1 & 3 & 2 & 3 & 3 & 1 & 0 & 2 & 0 \\
\hline
0 & 1 & 2 & 3 & 2 & 2 & 1 & 0 & 3 \\
3 & 0 & 1 & 2 & 3 & 2 & 2 & 1 & 0 \\
0 & 3 & 0 & 1 & 2 & 3 & 2 & 2 & 1 \\
1 & 0 & 3 & 0 & 1 & 2 & 3 & 2 & 2 \\
2 & 1 & 0 & 3 & 0 & 1 & 2 & 3 & 2 \\
2 & 2 & 1 & 0 & 3 & 0 & 1 & 2 & 3 \\
3 & 2 & 2 & 1 & 0 & 3 & 0 & 1 & 2 \\
2 & 3 & 2 & 2 & 1 & 0 & 3 & 0 & 1 \\
1 & 2 & 3 & 2 & 2 & 1 & 0 & 3 & 0
\end{array}\right)$$

Figura 2.10: Matriz $\mathcal{T}$ obtenida de la yuxtaposición de las matrices obtenidas del grupo $G = \{\delta, (2, 3)\}$ del vector $VI = \{0, 1, 3, 2, 3, 3, 1, 0, 2\}$.

2.1.9 Ciclotomía

En el 2009 Colbourn [35] presentó una técnica para construir CAs cíclicos a la que llamó ciclotomía, esta construcción solo es para potencias de primos en k. La construcción la realiza con un vector llamado Vector Inicializador (VI).

La construcción del VI se realiza a través de un elemento primitivo ε de los campos finitos de Galois $\mathbb{F}(q)$ con $q \equiv 1(\text{mód } v)$. Por cada q y ε, el $VI = (x_i : i \in \mathbb{F}_q) \in \mathbb{F}_q^q$ mediante el establecimiento de $x_0 = 0$ y $x_i = j \text{ mód } v$ donde $i = \varepsilon^j$ para $i \in \mathbb{F}_q^*$.

Una manera alternativa para construir el VI es a través de la tabla de logaritmos presentada por Torres-Jiménez *et al.* [39] en el 2010. Teniendo a $\mathcal{T}$ como la tabla de logaritmos, donde una celda en particular se denota por $\mathcal{T}_i$, $o \leq i \leq k-1$, el VI se construye por $VI_i = \mathcal{T}_i \text{ mód } v$ y $VI_0 = 0$.

Una vez que se tiene el VI, aplicando la operación de rotación se construye una matriz $\mathcal{A}$ de tamaño $k \times k$ a través de la operación de rotación sobre el VI. Al tener la matriz $\mathcal{A}$, pueden construir dos instancias de CAs, estas instancias son las siguientes:

- $CA(k; t, k, v)$. Esta instancia se obtiene si la matriz $\mathcal{A}$ contiene todas las tuplas de v^t en todas las submatrices de tamaño $k \times t$, de no ser así, se puede construir la siguiente instancia.
- $CA(kv; t, k, v)$. Esta instancia se obtiene yuxtaponiendo v copias de la matriz $\mathcal{A}$ y a cada copia se le aplica la operación $\mathcal{A}_{i,j} = (\mathcal{A}_{i,j} + s)$, donde s indica el número de la copia en $0 \leq s \leq v-1$ y $0 \leq i, j$, produciendo una matiz $\mathcal{X}$ de tamaño $kv \times k$, si la matriz $\mathcal{X}$ cubre las v^t combinaciones en cada submatriz de tamaño $kv \times t$.

Tabla 2.5: Tabla de logaritmos del valor 7.

Posición	Valor
1	0
2	2
3	1
4	4
5	5
6	3

Para ejemplificar la construcción anterior suponiendo que tenemos los valores $k = 7$ y $v = 2$, construimos la tabla de logaritmos (Tabla 2.5) y a cada $\mathcal{T}_i$ se realiza la operación $\mathcal{T}_i = \mathcal{T}_i \text{ mód } v$,

asignando el valor de 0 a la primera posición puesto que el logaritmo de 0 no existe, al hacer esto se obtiene el vector $\{0,0,0,1,0,1,1\}$. Construimos la matriz $\mathcal{A}$ rotando el vector y se construye el CA que se muestra en la Figura 2.11, construyendo un $CA(7;2,7,2)$.

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Figura 2.11: $CA(7;2,7,2)$, CA creado con el $VI = \{0,0,0,1,0,1,1\}$ de $k = 7$ y $v = 2$.

2.1.10 Vectores de Permutación

En el 2006 Sherwood *et al.* [49, 50] presentaron un método para construir CAs utilizando vectores de permutación (PV por sus siglas en inglés "*Permutation Vector*"). Un PV denotado por κ, es un vector de tamaño v^t, en donde un elemento en particular se denota por κ_i para $0 \leq i \leq v^t - 1$, cada κ_i se calcula de la siguiente forma:

$$\beta_0^{(i)} + (h_1 \times \beta_1^{(i)}) + (h_2 \times \beta_2^{(i)}) + \ldots + (h_{t-1} \times \beta_{t-1}^{(i)}) \tag{2.5}$$

Realizando las operaciones de suma y multiplicación a través de los GF(v), donde $\beta^{(i)}$ es la i-esima tupla de tamaño t de la forma $i = \sum_{j=0}^{t-1} v^j \beta_j^{(i)}$ y h es una tupla de tamaño $t-1$. Puesto que los PVs tienen las mismas v valores en los primeros v posiciones, estas posiciones se pueden eliminar de los PVs, estos vectores son vectores de permutación reducidos (SPV por sus siglas en inglés "*Shortened Permutation Vector*").

Para construir un CA utilizando los SPV se sigue los siguientes pasos, se escogen los valores para d y k y se forma una matriz $\mathcal{F}$ de tamaño $d \times k$ en donde en cada entrada de $\mathcal{F}$ es un número desde

cero hasta $v^{t-1}-1$. Se reemplaza cada entrada de $\mathcal{F}$ por el SPV indexado por la representación en base v, se agregan los v renglones constantes resultando en un $CA(d(v^t-v)+v; v, k, t)$.

Para ejemplificar la construcción tenemos los valores $d=2$, $k=10$ y la siguiente matriz $\mathcal{F}$

$$\mathcal{F} = \begin{pmatrix} 4 & 0 & 8 & 7 & 3 & 6 & 1 & 4 & 6 & 7 \\ 1 & 3 & 4 & 4 & 0 & 8 & 3 & 6 & 2 & 7 \end{pmatrix}$$

cambiamos la matriz $\mathcal{F}$ a su representación en base v y quedaría de la siguiente manera:

$$\mathcal{F} = \begin{pmatrix} 11 & 00 & 22 & 21 & 01 & 02 & 10 & 11 & 02 & 12 \\ 10 & 01 & 11 & 11 & 00 & 22 & 01 & 02 & 20 & 12 \end{pmatrix}$$

tomamos cada entrada de $\mathcal{F}$ como el vector h de la ecuación (2.5) y realizamos las operaciones de la siguiente manera (tomamos como $h=(1,1)$) para construir el SPV:

$$\beta^{(3)} = (0,1,0) = 0 + (1 \times 1) + (1 \times 0) = 1$$

$$\beta^{(4)} = (1,1,0) = 1 + (1 \times 1) + (1 \times 0) = 2$$

$$\beta^{(5)} = (2,1,0) = 2 + (1 \times 1) + (1 \times 0) = 0$$

$$\vdots$$

$$\beta^{(26)} = (2,2,2) = 2 + (1 \times 2) + (1 \times 2) = 0$$

al juntar todos los SPV, le agregamos los tres renglones constantes y nos da como resultado un $CA(54; 3, 10, 3)$.

2.2 Métodos Exactos

Los métodos exactos utilizan la búsqueda exhaustiva para encontrar un CA óptimo para los valores k, t y v. Para recorrer el espacio de búsqueda de manera mas inteligente incorporan diferentes técnicas como la ramificación y poda.

2.2.1 Algoritmo de Backtracking

En el 2009 Bracho-Rios *et al.* [51] propusieron un método exacto para construir CA de $v = 2$ para cualquier valor de t y k por medio de un algoritmo de ramificación y poda llamado *New Backtracking Algorithm* (NBA) con el número de símbolos balanceado por columna.

En esta técnica construyen una matriz de tamaño $N \times k$, luego construyen el elemento l perteneciente al conjunto de todas las posibles columnas con $\left\lfloor \frac{N}{2} \right\rfloor$ ceros y es insertado en la primer columna de una solución parcial $\mathbb{M}$. Entonces el siguiente elemento $l_{i-1} + 1$ es construido, y si el renglón i es mas pequeño que el $i + 1$ y este es un CA parcial, entonces el elemento es agregado, en cualquier otro caso se intenta con el siguiente elemento. Si ningún elemento se pudo insertar en la columna actual de $\mathbb{M}$, éste va hacia la última columna que se le agregó un elemento y se trata de agregar un nuevo elemento. Cuando las k columnas han sido insertadas el procedimiento termina. Para reducir el espacio de búsqueda se fija un bloque de la matriz $\mathbb{M}$ llamado bloque fijo.

Las desventajas del NBA son que cada columna agregada debe ser verificada para determinar si la nueva columna cumple el criterio de que el renglón i es mas pequeño que el $i + 1$ y que el bloque fijo no es la mejor inicialización posible, por lo tanto, muchas de las veces el *backtrack* lo hace en las primeras t columnas.

En el 2010, Bracho-Rios [52] en su tesis de maestría, supera las desventajas del NBA, al algoritmo lo llama *Improved Backtracking Algorithm* (IBA). El IBA divide en bloques de iguales y diferentes renglones. Esto garantizará que los renglones están ordenados. Al mover la última columna diferente a la siguiente posición valida se garantiza que las columnas están ordenadas lexicográficamente.

2.2.2 Algoritmo EXACT

En el 2006 Yian y Zhang [53] presentaron una herramienta para construir CAs y $MCAs$ de manera exhaustiva, llamado EXACT (*EXhaustive seArch of Combinatorial Test suites*). La herramienta se basa en la búsqueda de *backtracking*, tratando de asignar cada celda de la matriz de tamaño $N \times k$ y realiza el *backtracking* cuando ya no es posible construir el CA o MCA. Esta herramienta incorpora diferentes técnicas para reducir el tiempo de búsqueda. Una técnica que utilizan consiste en fijar una submatriz de tamaño $mb \times t$, donde $mb = v_0 \times v_1 \times \ldots \times v_{t-1}$ para reducir el *backtracking* desperdiciado. Otra técnica consiste en no explorar soluciones isomórficas que previamente ya fueron exploradas. Para eliminar las soluciones isomórficas, utilizaron la técnica llamada SCEH (The Sub-Combination Eqcualization Heurisitc),el cual verifica que en cualquier columna del CA, cada símbolo aparece casi el mismo número de veces, es decir, que esté balanceado.

2.2.3 Programación con Restricciones

En el 2006 Hnich *et al.* [54] exploraron cuatro modelos de programación con restricciones (CP por sus siglas en inglés "*Constraint Programming*"). Los modelos son: modelo de matriz *naïve*, modelo de matriz alternativa, modelo integrado y modelo de búsqueda local.

El modelo de matriz *naïve* consiste en una matriz de tamaño $N \times k$ de variables enteras, x_{ri}, para $1 \leq r \leq N$ y $1 \leq i \leq k$, tal que, si el valor del parámetro i está en el vector r es m, $r_{ij} = m$.

El modelo de matriz alternativa usa una matriz de variables enteras. Cada uno de los N renglones en esta matriz representa una posible configuración de los parámetros como el modelo anterior. Sin embargo, ahora hay $\binom{k}{t}$ columnas, cada una representa una de las posibles combinaciones de tamaño t. Por lo tanto, en este modelo, cada variable representa la tupla de variables (x_{ri}, x_{rj}, x_{rk}). El dominio de cada variable es $\{0, 1, \ldots, 2^t - 1\}$.

El modelo integrado asocia las variables de los dos modelos, unidos por limitaciones de canalización, de acuerdo a la manera en fue propuesto por Cheng *et al.* [55].

El modelo de búsqueda local realiza una búsqueda local a través del problema de satisfactibilidad (SAT), puesto que, los problemas SAT son vistos como un problema de satisfactibilidad con restricciones (CSP por sus siglas en inglés "*Constraint Satisfaction Problem*") con variables booleanas y restricciones no booleanas. La transformación la realizan de la siguiente manera: Se tiene una matriz $\mathcal{M}$ de tamaño $N \times k$ de enteros pertenecientes a $\mathbb{Z}_v$. Por cada renglón i, columna j y el valor x se define una variable booleana $m_{i,j,x}$ la cual es verdadera si y solo si x aparece en la posición (i, j). También se tiene una matriz $\mathcal{A}$ de tamaño $N \times \binom{k}{t}$ de enteros que pertenecen a $\mathbb{Z}_{v^t}$. Cada renglón i, columna j' y valor y se define una variable booleana $a_{i,j',y}$. Las siguientes limitaciones están definidas para toda $1 \leq i \leq N$, $1 \leq j \leq k$, $1 \leq j' \leq \binom{k}{t}$, $x, x' \in \mathbb{Z}_v$ y $y, y' \in \mathbb{Z}_{v^t}$. Cada posición de $\mathcal{M}$ y $\mathcal{A}$ debe de tener un símbolo:

$$\bigvee_x m_{i,j,x} \tag{2.6}$$

$$\overline{m_{i,j,x}} \vee \overline{m_{i,j,x'}} \tag{2.7}$$

$$\bigvee_{y} a_{i,j',y} \tag{2.8}$$

$$\overline{a_{i,j',y}} \vee \overline{a_{i,j',y'}} \tag{2.9}$$

donde $x < x'$ en (2.7) y $y < y'$ en (2.9). para cubrir la limitación:

$$\bigvee_{j'} a_{i,j',y} \tag{2.10}$$

Para canalizar las dos matrices infieren que los valores de las t entradas en $\mathcal{M}$ para las entradas correspondientes en $\mathcal{A}$:

$$\overline{a_{i,j',y}} \vee m_{i,j,x} \tag{2.11}$$

La búsqueda local empieza con una asignación aleatoria de valores a las variables booleanas, se selecciona una variable (en algunas ocasiones más) y se les reasigna el valor. Para seleccionar la variable y el valor se hace por búsqueda heurística.

2.2.4 Construcción de CAs Usando una Transformación al Problema de Satisfactibilidad (SAT)

En el 2008 López-Escogido, en su tesis de maestría [56], presento una transformación de $MCAs$ de $t = 2$ al problema de SAT. Dicha transformación es similar a la reportada por Hnich *et al.* [54].

La transformación de López-Escogido requiere de v_j, $0 \leq j \leq k-1$, variables booleanas $m_{i,j,x}$, $0 \leq x \leq v_j$, por cada elemento $m_{i,j}$ de una matriz $\mathcal{M}$ asociada a un MCA. El modelo de transformación requiere de los siguientes tres conjuntos de cláusulas

$$\pi_1 = \bigvee_{i=0}^{N-1} \left(\bigwedge_{\forall j,k | 0 \leq j < k, 0 \leq x < v} m_{ijx} \right) \tag{2.12}$$

$$\pi_2 = \bigvee_{i=0}^{N-1} \left(\bigwedge_{\forall j,x,y | 0 \leq j < k, 0 \leq x < y < v} (\overline{m_{i,j,x}} \vee \overline{m_{i,j,y}}) \right) \tag{2.13}$$

$$\pi_3 = \bigvee_{\forall x,y \mid 0 \leq x \leq y < v} \left(\bigvee_{\forall j,l \mid 0 \leq j < l < k} \left(\bigwedge_{\forall i \mid 0 \leq i < N} (m_{i,j,x} \wedge m_{i,l,y}) \right) \right) \tag{2.14}$$

Estos tres conjuntos de cláusulas (2.12), (2.13) y (2.14) sirven para asegurar que las variables booleanas $m_{i,j,x}$ asociadas al elemento $m_{i,j}$ tomen al menos un valor. Haciendo la conjunción de las tres ecuaciones se obtiene la fórmula completa de una instancia de MCA ($F = \pi_1 \wedge \pi_2 \wedge \pi_3$).

2.3 Métodos Voraces

Los métodos voraces son aquellos que a través de una búsqueda rápida garantizan construir CA, proporcionando una solución en menor tiempo que una búsqueda exhaustiva.

2.3.1 Generación de Casos Eficiente Automático (AETG)

En 1997 Cohen *et al.* [57, 58, 11] presentaron una herramienta para generar casos de prueba a la que llamaron AETG (Automatic Efficient Test Generator). El AETG es una herramienta que genera CA y MCA para $v \geq 2$ y $t \geq 2$, aunque solo se uso para generar casos de $t = 2$ y $t = 3$ [58]. La herramienta construye los casos de prueba generando un renglón a la vez, el renglón se genera de acuerdo a métodos probabilísticos. Los métodos probabilísticos construyen el renglón que tenga la mayor probabilidad de cubrir el mayor número de t-tuplas faltantes, tomando en cuenta

las tuplas ya cubiertas en los renglones anteriores. Este proceso se repite hasta que todas las tuplas estén cubiertas.

2.3.2 Algoritmo de Densidad Determinista

En el 2007 Bryce y Colbourn [59] presentaron un generador de casos de prueba para $t = 2$ al que llamaron Algoritmo de Densidad Determinista (DDA por sus siglas en inglés "Deterministic Density Algorithm"). La característica principal del DDA es que construye un renglón a la vez del CA. El DDA fija factores de manera dinámica basados en la densidad y agrega nuevos renglones hasta que todas las interacciones son cubiertas. El DDA consta de cuatro pasos, en el primer paso se calculan los factores de densidad, en el segundo paso se aplica una regla para desempatar factores (solo si existen dos o más densidades de factores iguales), en el tercer paso se calculan las densidades por nivel y en el cuarto paso se aplica una regla para desempatar niveles (solo cuando existen dos o más niveles con la misma densidad).

El factor de densidad (δ) es calculado de acuerdo al número de pares no cubiertos entre el factor i y el factor j ($r_{i,j}$) y la mayor cardinalidad de los dos factores ($\mathscr{L}_{max}$) siguiendo uno de los siguientes tres casos

a) Si ambos factores tienen más de un nivel faltante $\delta_{i,j} = \left(\frac{r_{i,j}}{\mathscr{L}_{max}^2}\right)^2$.

b) Si la diferencia de los factores es uno $\delta_{i,j} = \left(\frac{r_{i,j}}{\mathscr{L}_{max}}\right)^2$.

c) Si ambos factores tienen exactamente un nivel menos y si el nuevo par cubre $\delta_{i,j} = 1{,}0$; en otro caso $\delta_{i,j} = 0{,}0$

Las reglas de desempate de factores consisten en seleccionar el mayor de los factores en orden lexicográfico. Una vez que un factor es seleccionado se le asigna un nivel. El nivel de densidad se

calcula para un nivel especifico v_i en relación a un factor individual k_j de acuerdo a los siguientes dos casos:

a) Si k_j tiene más de un nivel faltante con v_i, $r_{i,j}$ es el número de niveles no cubiertos con v_i y $\mathscr{L}_{max}$ es el máximo número de niveles asociados a cualquier factor, $\delta = \left(\frac{r_{i,j}}{\mathscr{L}_{max}}\right)$.

b) Si k_j tiene solo un nivel faltante y el nuevo par es cubierto, $\delta = 1{,}0$; en cualquier otro caso $\delta = 0{,}0$.

2.3.3 Orden por Parámetro y Demás Variantes

En 1998 Tai y Lei [60] muestran una estrategia de construcción llamada orden por parámetro (IPO, por sus siglas en inglés *In-Parameter-Order*). El IPO construye CA de $t = 2$ expandiendo el CA en renglones y columnas. Posteriormente en el 2007 Tai y Lei [61] presentaron una generalización del IPO llamada orden por parámetro generalizado (IPOG por sus siglas en inglés *In Parameter Order Generalized*). El IPOG generalizó la construcción del CA para $t \geq 2$. En el 2008 Forbes *et al.* [62] presentaron una mejora del algoritmo del IPOG, llamando a esta mejora refinamiento sobre el orden por parámetro general (IPOG-F).

Los tres algoritmos están basados en la misma estrategia, usando un CA de $k-1$ columnas para construir un CA de k columnas. Para agregar un nuevo parámetro al CA se puede realizar por medio de alguno de las siguientes opciones:

a) Crecimiento horizontal. Extiende cada renglón agregando un nuevo elemento.

b) Crecimiento vertical. Agrega un nuevo renglón si es necesario. Este crecimiento se realiza despues de completar el crecimiento horizontal.

El IPO y IPOG pueden realizar el crecimiento horizontal de dos maneras, una es por clases de equivalencia (IPO_H_EC) y la otra es por valores individuales (IPO_H_IV). El IPO_H_EC calcula el

conjunto de permutaciones, luego encuentra la permutación con el menor número de combinaciones que cubra la mayoría de las t-tuplas faltantes para usarla en el crecimiento horizontal. El IPO_H_IV construye un conjunto π con todas las tuplas faltantes entre p_i y cualquier otro $p_1, p_2, \ldots, p_{i-1}$, después se selecciona un p_i que cubra la mayoría de t-tuplas faltantes, se remueven de π y se agregan a $\mathbb{T}$. El IPOG-F realiza el crecimiento horizontal seleccionando de manera voraz en renglón y el elemento para el renglón, esto posibilitó al IPOG-F producir mejores resultados.

Los tres métodos realizan el crecimiento vertical de la siguiente manera, primero definen un conjunto π que contiene todas las t-tuplas faltantes del CA denotado por $\mathbb{T}$, por cada tupla $(p_k.w, p_i.u)$ del conjunto π se verifica si en $\mathbb{T}$ hay un renglón que contiene un elemento comodín en la posición p_k y el valor de u en la posición p_i, al elemento comodín en la posición p_k toma el valor w, para cualquier otro caso, se agrega un nuevo renglón asignando en la posición p_k el valor de w, en la posición p_i el valor u y en las posiciones restantes se asignan elementos comodines.

2.3.4 Coverage Inheritance

En el 2012 Calvagna y Gargantini [63] mostraron un método para construir $MCAs$ de fuerza variable agregando un parámetro a la vez. El método inicializa un MCA cubriendo las primeras t columnas las t-tuplas. El resto de los parámetros $j-1$ parámetros $(t \leq j \leq k)$ se combinarán para construir un MCA de fuerza t. El procedimiento para agregar una nueva columna consta de los siguientes seis pasos:

1. Se escoge una columna i y se copia en la columna j tal que $i < j$ asignando x al símbolo z de la columna i tal que $z > v_j$, esto con el fin de indicar que la entrada se encuentra libre para ser asignada. Al copiar la columna j cada subarreglo de t columnas cubre las características del MCA de fuerza t a excepción de las columnas i y j. A esta propiedad se le denomina *Coverage Inheritance*.

2. Por cada tupla faltante se realizan los siguientes pasos:

   i. Se inicializan un conjunto de banderas, una por cada posición entrada de la columna j. Cada bandera indica si la entrada puede ser modificada.

   ii. Modificar una entrada de la columna j para incrementar las t-tuplas cubiertas con respecto a la columna i.

   iii. Si al modificar la entrada se destruyen t-tuplas heredadas, de manera recursiva se modifican las entradas libres para recuperar la t-tupla destruida.

   iv. Cuando no hay mas entradas libres para cambiar se agrega un nuevo renglón para recuperar la t-tupla destruida y termina la recursividad.

2.4 Métodos Heurísticos

Los métodos heurísticos suelen dar buenos resultados en tiempos de cómputo razonables. Entre los diferentes algoritmos existentes destacan los algoritmos genéticos, la búsqueda tabú y el recocido simulado.

2.4.1 Algoritmos Genéticos

Los algoritmos genéticos (GA por sus siglas en inglés *Genetic Algorithm*) fueron presentados por John Holland en 1975 [64]. Los GA son algoritmos de búsqueda que se basan en la simulación de la evolución biológica, los cuales hacen evolucionar una población de individuos sometiéndola a acciones aleatorias, semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos mas adaptados (sobreviven) y cuáles los menos aptos (son descartados). Los GA han sido usados para resolver problemas de optimización combinatoria.

En el 2001 Stardom [65] presentó un GA para construir CAs. El método requiere como entrada el tamaño de la población $\mathcal{S}$. La población $\mathcal{S}$ es un conjunto de matrices con t-tuplas faltantes. La función de evaluación verifica el número de t-tuplas faltantes. Para seleccionar los elementos a combinar, el conjunto $\mathcal{S}$ se divide en dos grupos de tamaño $\frac{|\mathcal{S}|}{2}$, de cada grupo se selecciona de manera aleatoria una matriz y se combinan esas dos matrices para producir $|\mathcal{S}|$ individuos nuevos. La operación de combinación consiste en seleccionar un conjunto $\mathcal{E}$ de coordenadas (i, j) de un padre y copiándolas en un hijo, el resto de las coordenadas para completar al hijo se toman del segundo padre. El conjunto $\mathcal{E}$ de un padre puede ser los primeros i renglones completos, los primeros j renglones completos o un bloque de los primero i renglones y las primeras j columnas, para $0 \leq i \leq N-1$ y $0 \leq j \leq k-1$. Después se aplica la operación de mutación a los nuevo $\mathcal{S}'$ individuos. La operación de mutación consiste en cambiar el elemento de una entrada seleccionada de manera aleatoria por otro elemento diferente al anterior de la entrada. Se evalúan los conjuntos $\mathcal{S}$ y $\mathcal{S}'$ y se seleccionan las matrices que tengan el menor número de t-tuplas faltantes para pasar a la siguiente generación.

En el 2004 Shiba *et al.* [66] proponen un GA que es una modificación del AETG [58]. La representación que proponen es que cada cromosoma del GA es un caso de prueba. La función de evaluación está definida para un caso de prueba S como el número de las tuplas que no son cubiertas por el conjunto de prueba dado pero están cubiertas por S. En la inicialización de la población, los m candidatos son generados de manera aleatoria y para la selección se utiliza una selección por torneo, esta selección es aplicada a los m casos en la población P. Este método tiene la característica de que se tiene σ cromosomas en la población que se mantienen y sobreviven a la siguiente generación intacta. La operación de cruza intercambia entre dos cromosomas los valores por cada posición independientemente con una probabilidad de 0.5. La mutación reemplaza el valor de una posición con otro valor tomado de manera aleatoria. Al final de cada ciclo se evalúa si se estancó, si el número de generaciones excede al periodo máximo, se aplica una mutación masiva a cada posición de todos los cromosomas.

2.4.2 Búsqueda Tabú

La búsqueda tabú (TS por sus siglas en inglés *Tabu Search*) [67], está basada en que las decisiones tomadas anteriormente no deben de ser tomadas en cuenta en un lapso de tiempo. Estas decisiones se llaman decisiones tabú. Los algoritmos que se basan en esta búsqueda definen una lista tabú, la cual contendrá una cierta cantidad de decisiones que no se repetirán en un lapso de tiempo, esto para evitar regresar a un óptimo local.

En 2004 Nurmela [68] utilizó TS para construir CA de fuerza dos y tres. El método empieza inicializando una matriz de $N \times k$ con valores aleatorios. El costo de la matriz es el número de las tuplas que no están cubiertas. Luego se selecciona una tupla aleatoria que no esta cubierta. Verifican qué renglón requiere un solo cambio de un elemento, tal que, el renglón cubra la tupla seleccionada, estos cambios representan los movimientos en el vecindario actual. En el caso de que no exista un renglón, se selecciona un renglón y se escribe la tupla en él. El costo del movimiento es el total de movimientos realizados y el movimiento con menor costo es seleccionado, siempre y cuando el movimiento no sea un movimiento tabú. En el caso de que exista más de un movimiento con el menor costo y que no estén en la lista tabú, se selecciona un movimiento al azar. Este proceso se repite hasta que todas las t-tuplas estén cubiertas o que el número de movimientos ha llegado al límite.

En el 2010 González-Hernández *et al.* [69] utilizó un algoritmo de TS para construir $MCAs$ de fuerza variable. El método empieza generando una solución inicial $\mathcal{M}$ de tres maneras, la primer manera es de generando cada entrada de manera aleatoria, la segunda es a través de la distancia de Hamming maximizando esta distancia entre los renglones y la tercera es con los elementos balanceados. Las modificaciones que se realizan a la solución $\mathcal{M}$ es a través de tres funciones de vecindad. Para las funciones de vecindad se requiere definir tres conjuntos, el conjunto $\mathcal{C}$ contiene las

tuplas que están cubiertas, el conjunto $\mathcal{A}$ contiene todas las combinaciones de símbolos que deben ser cubiertas por las tuplas de $\mathcal{C}$ y el conjunto $\mathcal{R}$, el cual, cada elemento $r_i \in \mathcal{R}$, será un conjunto de prueba del MCA que se construirá. La primer función está dividida en dos partes, la primer parte busca una tupla $c_i \in C$ se encuentre cubierta, pero que al menos contenga una combinación de símbolos a' faltantes, la segunda parte busca el mejor caso $r \in \mathcal{R}$, donde la combinación de símbolos a' pueda ser sustituida. La segunda función selecciona de manera aleatoria una columna de la solución $\mathcal{M}$, después, por cada caso $r_i \in \mathcal{R}$, se cambia el símbolo $r_{i,j}$ donde j es el j-ésimo símbolo en $r_i \in \mathcal{R}$ y se evalúa el número de combinaciones de símbolos faltantes. La tercer función es una generalización de la segunda función, el cual, realiza todos los cambios de los símbolos en todo el conjunto $\mathcal{R}$. La lista tabú almacena el renglón, la columna y el símbolo usados por la función de vecindad y también almacena el número tuplas faltantes. El tiempo de expiración $\mathcal{T}$ es definido como el número de veces que la función de evaluación es ejecutada.

En el 2012 González-Hernández [21] propuso un algoritmo de TS para construir $MCAs$ de fuerza dos hasta fuerza seis. El método empieza creando una matriz $\mathcal{M}$ de tamaño $N \times k$. Para crear la solución inicial, proponen tres algoritmos, el primero llena la matriz con valores aleatorios, el segundo inicializa la matriz con la mayor distancia de Hamming y el tercero inicializa cada columna con un número balanceado de símbolos. La exploración del vecindario se realiza a través de cuatro funciones de vecindad. La primer función de vecindad selecciona de manera aleatoria una celda de la matriz $\mathcal{M}$ y realiza todos los posibles cambios de símbolos en la celda. La segunda función trabaja sobre el conjunto $\mathcal{R} = \{r_1, \ldots, r_N\}$, en el cual, cada elemento $r_i \in \mathcal{R}$ será un caso de prueba del MCA que está siendo construido, la función de vecindad selecciona una columna del conjunto de prueba y cambia el símbolo de $r_{i,j}$, donde j es el j-ésimo símbolo en $r_i \in \mathcal{R}$. La tercer función de vecindad realiza todos los cambios de símbolos del conjunto de pruebas $\mathcal{R}$. La cuarta función de vecindad consta de dos partes, la primer parte selecciona una tupla que va a ser cubierta, tal que, esta tupla contenga al menos un símbolo de la combinación faltante, la segunda parte consiste en encontrar el

mejor caso $r \in \mathcal{R}$. La lista tabú está formada por la tupla $(\mathcal{N}, v, i, j, m)$, donde $\mathcal{N}$ es la función de vecindad utilizada y v es el símbolo que fue asignado en la celda $m_{i,j} \in \mathcal{M}$. Los movimientos que se agregan a la lista son aquellos que no disminuye o aumenta el número de tuplas faltantes en la matriz $\mathcal{M}$. Para evaluar las matrices resultantes de las funciones de evaluación cuentan el número de tuplas faltantes. Para el criterio de terminación utilizaron dos criterios, que el número de tuplas faltantes sea cero y que el número de evaluaciones halla llegado al limite.

2.4.3 Recocido Simulado

El recocido simulado [70] (SA por sus siglas en inglés *Simulated Annealing*) es un algoritmo de búsqueda que se basa en la simulación de enfriar lentamente un sólido que se encuentra inicialmente en estado líquido. El enfriamiento de sólidos en un proceso físico en el cual, primero se calienta el sólido hasta que el sólido cambie a estado líquido, entonces, el objeto es enfriado lentamente y esto hace que sus partículas se encuentren en un estado bajo de energía.

En el 2001 Stardom en su tesis de maestría [65], realizó una comparativa entre el SA, TS y GA, la cual, el SA entregó los mejores resultados entre los tres métodos. El SA inicia con un $CA(N; t, k, v)$ como la solución inicial, la función de evaluación regresa el número tuplas faltantes para completar el CA, la función de vecindario selecciona de manera aleatoria un elemento de la matriz y cambia el valor por otro valor diferente que pertenezca al alfabeto y el decremento de la temperatura se realiza a través de un factor de enfriamiento δ_t, multiplicando el δ_t con la temperatura actual y actualizando la temperatura actual con el resultado de la multiplicación.

En el 2008 Covarrubias-Flores en su tesis de maestría [71], presentó un SA para construir CAs de alfabeto binario y fuerza variable. Covarrubias-Flores propuso que la solución inicial $\mathcal{M}$ sea creada de manera aleatoria, asignando a cada celda un valor aleatorio 0 o 1 y debe satisfacer que en cada

columna el número de símbolos esté balanceado, es decir, la cantidad de unos se obtiene por la fórmula $\left\lceil \frac{N}{2} \right\rceil$ y la cantidad de ceros por $N - \left\lceil \frac{N}{2} \right\rceil$. La función de evaluación cuenta el número de las t-tuplas faltantes. Para realizar cambios en la matriz $\mathcal{M}$, propuso cinco funciones de vecindad, la primer función selecciona de manera aleatoria una celda $m_{i,j} \in \mathcal{M}$ y se cambia, la segunda función realiza β veces la primer función y selecciona la que minimice el valor resultante de la función de evaluación, la tercer función intercambia el valor de dos elementos seleccionados de manera aleatoria y que estén en la misma columna, siempre y cuando, los valores sean diferentes, la cuarta función selecciona de manera aleatoria un elemento $m_{i,j} \in \mathcal{M}$ y evalúa todos los posibles intercambios que se pueden hacer con los elementos de la misma columna y la última función de vecindad crea un nuevo vecino realizando todos los posibles intercambios en una columna j elegida aleatoriamente.

En 2008 Cohen *et al.* [72] aplicaron un método que combina métodos algebraicos y SA para generar casos de fuerza tres para diferentes valores de k y v. A este enfoque lo denominaron *Augmented Annealing* (AA). El SA inicia creando una solución inicial aleatoria S, la solución S se transforma en S' a través de la función de vecindario, la cual, consiste en seleccionar una columna de manera aleatoria y de manera aleatoria se selecciona un elemento de la matriz, el valor de dicho elemento de la matriz se reemplaza por el de otro elemento de la matriz que no pertenece a la columna seleccionada. El costo de la solución S' es el número de combinaciones faltantes. Si S' resulta en una solución factible de menor o igual costo, se acepta la solución S'. Si S' es de mayor costo, S' es aceptado con la probabilidad de $e^{-(c(S')-c(S))}/T$.

En el 2010 Martínez-Peña *et al.* [43], presentan un algoritmo de SA para construir CAs de alfabeto tres utilizando los CTs. El algoritmo empieza construyendo una solución inicial $(\mathbf{A})$ seleccionando $t + k$ CTs, donde t es la fuerza y k es el número de columnas del CA ternario. Para realizar los cambios en la solución $\mathbf{A}$, propusieron cuatro funciones de vecindad, la primer función de vecindad agrega o quita el i-ésimo CT en $\mathbf{A}$, luego mide la solución $\mathbf{A}'$ generada. La segunda función genera

todos los vecinos de $\mathbf{A}$, el cual, el número de vecinos generados es de $\binom{k+2}{2}$ y por cada vecino generado mide la calidad de la solución, es decir, realiza $2\binom{k+2}{2}$. La tercer función intercambia el i-ésimo CT en $\mathbf{A}$ por el j-ésimo CT que no está en $\mathbf{A}$ dos veces y miden la solución generada. La cuarta función es una generalización de la función anterior, realizando $4\left\lfloor\binom{k+2}{2}\right\rfloor\left\lceil\binom{k+2}{2}\right\rceil$. La función de evaluación minimiza el número de combinaciones no cubiertas u agregando la división del número de renglones del vecino N entre 3^k

$$u + \frac{N}{3^k}$$

Para detener el algoritmo utilizaron tres criterios, el primero es que la temperatura final ha sido alcanzada, que el número de iteraciones al límite establecido y que el número de llamadas a la función de evaluación ha sido superada.

En el 2012 Torres-Jiménez y Rodriguez-Tello [73] utilizaron un SA para construir CA de alfabeto binario y fuerza variable. La representación que utilizaron fue una matriz de tamaño $N \times k$ con símbolos del conjunto $\mathbb{Z}_v$ denotada por A. La matriz A se inicializa de manera que en cada columna tenga un número balanceado de símbolos, esto se hace asignando de manera aleatoria $\left\lceil\frac{N}{2}\right\rceil$ unos a cada columna y $\left\lfloor\frac{N}{2}\right\rfloor$ ceros a cada columna. La función de evaluación verifica el número t-tuplas faltantes a la solución potencial. Las funciones de vecindad se basan en búsqueda local, una de las funciones cambia el valor de la posición i, j de A por algún otro valor contenido en el conjunto de v, esta función se llama $swich(A, i, j)$, la segunda función de vecindad intercambia los valores de los renglones i y l de A dentro de la columna j, a esta función la llamaron $swap(A, i, j, l)$. Para detener el algoritmo, se usaron tres criterios, el primer criterio es cuando la temperatura actual llega a la temperatura final, el segundo criterio es cuando se encuentra el CA válido y el último criterio es

cuando después de varios decrementos de la temperatura no mejora le solución actual. Este trabajo fue una mejora del algoritmo presentado por Torres-Jiménez y Rodriguez-Tello [39] en el 2010.

En el 2012 Avila-George *et al.* [74] paralelizaron un SA para construir CAs de alfabeto ternario. El algoritmo secuencial crea una matriz $\mathcal{M}$ de tamaño $N \times k$. La matriz $\mathcal{M}$ es inicializada a través de la distancia de Hamming entre dos renglones. La función de evaluación verifica el número de símbolos faltantes en la matriz $\mathcal{M}$. Para las funciones de vecindad, se utilizaron dos funciones diferentes, la primer función selecciona de manera aleatoria una tupla faltante y trata de asignarla en la solución $\mathcal{M}$, la segunda función selecciona de manera aleatoria una posición de la matriz $\mathcal{M}$ y realiza todos los posibles cambios de símbolos. Para detener el algoritmo utilizaron tres criterios, cuando se alcance la temperatura final, cuando cesa el progreso o cuando el CA es encontrado. Para paralelizar la versión secuencial del SA, utilizaron tres enfoques, el primero fue de búsqueda independiente, el segundo fue de búsqueda semi-independiente y la tercera fue de búsqueda cooperativa. La búsqueda independiente es ejecutar en cada procesador el SA secuencial, solo que, todos inician con la misma solución inicial. La búsqueda semi-independiente divide la cadena de markov en $\mathcal{P}$ subcadenas. La búsqueda cooperativa utiliza comunicación asíncrona accesando a un estado global para eliminar los tiempos muertos.

2.5 Manipulación de CA

Los métodos de manipulación realizan operaciones a un CA para crear CAs más adecuados para algún propósito o verificar si una matriz es o no CA. Algunas de las operaciones son: maximizar el número de renglones constantes, aumentar el número de comodines, reducir de manera óptima el CA, reducir el número de renglones del CA o verificar un CA.

2.5.1 Detección de Símbolos Comodines

En el 2011 González-Hernández *et al.* [75] presentaron un método para detectar símbolos comodines en un CA. El método consta de tres pasos:

1. Determinar que t-tuplas están cubiertas una sola vez, fijando los elementos de estas tuplas.

2. De acuerdo a los elementos que no están fijos se determinan todas las posibles configuraciones de comodines.

3. Se realiza una búsqueda seleccionando una t-tupla de cada combinación que no esté fija y se selecciona aquella combinación de t-tuplas que maximice el número de elementos comodines.

Para determinar que t-tuplas están cubiertas una sola vez, se verifican las $\binom{k}{t}$ combinaciones de las columnas y por cada combinación se cuentan las veces que aparece cada t-tupla para fijar aquellas tuplas que solo aparezcan una sola vez. Una vez que se fijaron las t-tuplas se verifican las $\binom{k}{t}$ combinaciones de columnas para determinar si algún elemento no está fijo y de ser así, se agrega a una lista la t-tupla y la combinación en la que se encuentra. Al tener la lista, se selecciona una t-tupla de cada combinación que no esté fija para maximizar el número de elementos comodines a través de un algoritmo de ramificación y poda.

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Figura 2.12: $CA(8; 2, 16, 2)$, CA usado para detectar elementos comodines.

Para ejemplificar la detección suponemos que tenemos un $CA(8;2,16,2)$ (Figura 2.12), recorremos todas sus combinaciones de columnas y denotamos los elementos no fijos por el símbolo $*$ (Figura 2.13), se crea la lista con todas las combinaciones y las tuplas que contengan un elemento no fijo y a través de la búsqueda de ramificación y poda se seleccionan las t-tuplas que maximicen los elementos comodines detectando los primeros doce elementos del último renglón como elementos comodines (Figura 2.14).

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & * & * & * & 0 & * & * & * & * & * & * & * & * & * & 1 \\ * & * & * & * & * & * & * & * & * & * & * & * & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 2.13: $CA(8;2,16,2)$, CA con los elementos fijos.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{2} & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 2.14: $CA(8;2,16,2)$, CA con los primeros doce elementos del último renglón como elementos comodines.

2.5.2 Maximización de Renglones Constantes

En el 2010 Quiz-Ramos [76] en su trabajo de tesis de maestría presenta un método para maximizar el número de renglones constantes transformando el CA en un grafo, donde cada renglón del CA es un nodo del grafo y existe un arco entre dos nodos del grafo si los renglones correspondientes son

compatibles. Dos renglones $\mathcal{R}$ y $\mathcal{R}'$ son compatibles si $\mathcal{R}_i \neq \mathcal{R}'_i$, donde $0 \leq i \leq k-1$. El problema de maximización de renglones constantes se convierte en determinar el MAXCLIQUE [77] del grafo.

El método primero genera un grafo con N nodos y el número de arcos es determinado por la pareja de renglones compatibles existentes en el CA, luego se busca el MAXCLIQUE que tiene el grafo, al encontrar el MAXCLIQUE se realizan los cambios de símbolos en el CA con los renglones del MAXCLIQUE y se determina el número de renglones constantes construidos.

Tabla 2.6: $CA(12;2,4,3)$, Tabla con las etiquetas desde cero hasta once de los renglones del CA.

Etiqueta	Renglones			
0	0	2	2	1
1	2	1	1	0
2	1	0	0	2
3	0	0	1	2
4	0	1	1	0
5	0	2	0	0
6	1	0	2	0
7	1	1	0	1
8	1	2	1	2
9	2	0	0	1
10	2	1	2	2
11	2	2	1	1

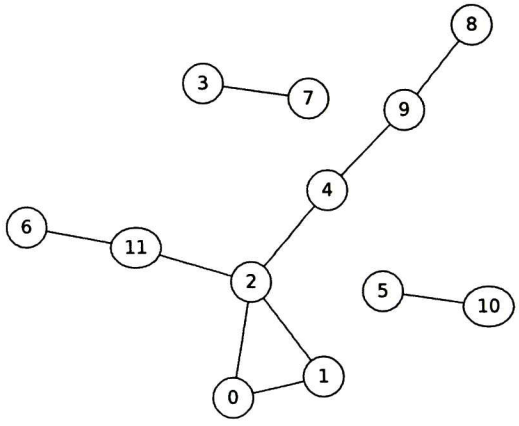


Figura 2.15: Grafo del CA, el grafo tiene once nodos y diez arcos.

Para ejemplificar la maximización de renglones constantes tomamos un $CA(12;2,4,3)$ (Tabla 2.6) y transformamos el CA en un grafo, como se muestra en la Figura 2.15. En ese grafo se ve que el MAXCLIQUE esta en los renglones 0, 1 y 2, de esos renglones cambiamos los símbolos para juntar el mismo símbolo en un solo renglón y obtenemos un CA con tres renglones constantes (Figura 2.16).

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 1 & 1 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 2 & 1 & 2 \\ 2 & 0 & 0 & 1 \\ 2 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 \end{pmatrix}$$

Figura 2.16: $CA(12; 2, 4, 3)$, CA obtenido después de maximizar los renglones constantes, los renglones que se hicieron constantes fueron los últimos tres renglones.

2.5.3 Reducción Óptima de CAs

En el 2011 Carrizales-Turrubiates [25], en su tesis de maestría presenta doce algoritmos para resolver el problema de Reducción Óptima de Covering Arrays (PROCA). El problema de PROCA consiste en encontrar una submatriz β de tamaño $N - \delta \times k - \Delta$, donde δ y Δ son números enteros tal que, $(0 \leq \delta \leq N - v^t) \wedge (0 \leq \Delta \leq k - t)$ a partir de un $CA(N; t, k, v)$ o un cuasi-CA(*N; t, k, v*)[2] denotada por $\mathcal{A}$, satisfaciendo al menos $\delta > 0$ o $\Delta > 0$. Una solución del PROCA requiere de los conjuntos $\mathcal{J}_R$ y $\mathcal{J}_C$, donde $\mathcal{J}_R$ es el conjunto de renglones tomados de $\mathcal{A}$ y $\mathcal{J}_C$ es el conjunto de columnas tomadas de $\mathcal{A}$ para formar la submatriz β.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Figura 2.17: Ejemplo de la matriz de entrada $\mathcal{A}$ del PROCA $\mathcal{A} = CA(6; 2, 5, 2)$

[2]Dicha matriz tiene un número relativamente pequeño de tuplas faltantes en relación con el número de las tuplas a cubrir.

Para ejemplificar la solución, tenemos una matriz $\mathcal{A}$ como se muestra en la Figura 2.17, tomamos los renglones del conjunto $\mathcal{J}_R = \{0, 2, 4, 5\}$ y las columnas del conjunto $\mathcal{J}_C = \{2, 3, 4\}$ para formar un CA de $CA(4; 2, 3, 2)$ (Figura 2.18).

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Figura 2.18: Ejemplo de la submatriz $\beta = CA(4; 2, 3, 2)$ obtenida de la matriz $\mathcal{A}$ cuando $\delta = 2$ y $\Delta = 2$.

2.5.4 Validación de CAs

En el 2010 Avila-George [24], en su tesis de maestría propuso un algoritmo paralelo y otro algoritmo *grid*, para validar CAs y $MCAs$. El algoritmo secuencial en el que se basaron para desarrollar los dos algoritmos trabaja de la siguiente manera, se tiene una matriz $\mathcal{P}$ de tamaño $\wp \times \prod_{i=1}^{t-1} MaxV_i$, donde $\wp$ es la cardinalidad de las tuplas y $MaxV_i$ contiene todas las t cardinalidades de v de valor mayor, un vector $\mathcal{V}$ de tamaño k con las cardinalidades de las variables y una matriz $\mathcal{J}$ de tamaño $\wp \times t$ donde cada renglón representa una tupla del CA. En $\mathcal{P}$ se lleva el registro de las combinaciones de símbolos existentes por cada tupla del CA, cada una de las tuplas se mapea a un renglón de $\mathcal{P}$, siguiendo la fórmula (2.15).

$$\sum_{i=0}^{t-1} \binom{x_i}{i+1} \tag{2.15}$$

Para verificar que todas las combinaciones de los símbolos de cada t columnas del CA aparezcan al menos una vez, es necesario mapear cada t combinaciones de símbolos a un valor decimal, el cual, corresponde a una columna de $\mathcal{P}$, el mapeo se lleva acabo siguiendo la siguiente fórmula

$$\sum_{i=1}^{t-1} J_i$$

donde $J_i = J_{i-1} \times V_{J_i} + J_i$. Para CAs uniformes, se valida que en la matriz $\mathcal{P}$ no existe en ningún cero y para los $MCAs$ se valida que no existe ningún cero para cada tupla (J_i) en las primeras $\prod_{i=0}^{t-1} V_{J_i}$ columnas de su renglón correspondiente en $\mathcal{P}$.

El algoritmo paralelo divide en bloques las tuplas y cada bloque se asigna a cada procesador, cada procesador (incluido el procesador maestro) calcula el número de tuplas faltantes y los resultados los envían al procesador maestro (excepto el procesador maestro), el procesador maestro recibe los resultados de los otros procesadores.

El algoritmo *grid* es el mismo que el algoritmo paralelo, con la diferencia de que, el algoritmo *grid* se aplica cuando los CA son de $t > 5$ y $k > 1000$ y que éste no solo se reparte en un cluster, sino se puede repartir en clústers, en PCs o supercomputadoras.

2.5.5 Fusión y Fusión Generalizada

En el 2008 Colbourn [34] muestra una operación para la construcción de un $CA(M; t, kv-1)$ a partir de un $CA(N; t, k, v)$ a través de sustituir las ocurrencias del elemento $v-1$ por un símbolo del conjunto $\{0, 1, \ldots, v-2\}$ y usar un mecanismo para eliminar interacciones redundantes entre parámetros. A este procedimiento se le nombró fusión en el 2010 por Colbourn *et al.* [78]. El operador de fusión establece el CAN de acuerdo a la ecuación (2.16) [34].

$$CAN(t,k,v) \leq CAN(t,k,v+1) - \begin{cases} 3 & si \quad t = 2, k \leq v+1, v \text{ es un número primo} \\ 2 & \text{cualquier otro caso} \end{cases} \tag{2.16}$$

Para el primer caso, se selecciona un renglón r_1 y renombramos en cada columna c_j, donde $0 \leq j \leq k-1$, de manera que el valor $v-1$ aparezca en cada tupla (r_1, c_j). Después se selecciona

otro renglón r_2, tal que $r_1 \neq r_2$. Reemplazamos cada tupla (r_i, c_j) que contenga el elementos $v-1$, tal que $r_i \neq r_1 \neq r_2$ y $o \leq i \leq N-1$, con el valor de la tupla (r_2, c_i). Al terminar se eliminan los renglones r_1 y r_2, dando como resultado un $CA(N-2, t, k, v-1)$.

Para el segundo caso, tenemos un $CA(N; t, k, v)$ óptimo $\mathcal{A}$, permutamos los símbolos en las columnas de tal manera que obtengamos un renglón constante con el elementos $v-1$, este renglón se elimina y se marcan todas las ocurrencias del elementos $v-1$ con $*$ en el resto de los renglones. Reemplazamos los símbolos $*$ con uno de los símbolos del conjunto $\{0, 1, \ldots, v-2\}$ y se construye un $CA(v^2; 2, v+1, v-1)$.

En el 2012 Rodríguez-Cristerna [27], en su tesis de maestría generalizó el operador de fusión, trabajando con CA y matrices con tuplas faltantes.

La generalización fue dividida en dos fases, la primer fase consiste en interpretar, reemplazar los elementos comodines, ordenar, detectar los elementos comodines y eliminar y reemplazar los renglones redundantes (el reemplazo de los elementos comodines y el ordenamiento es opcional) y la segunda fase la conforma un algoritmo heurístico.

El proceso empieza con un CA o una matriz con t-tuplas faltantes, el paso de interpretar cambia los elementos de la matriz de entrada al alfabeto del CA de salida. El reemplazo de elementos comodines cambia los elementos del CA de entrada que no pertenezcan al conjunto $\mathbb{Z}_v$ del CA de salida, estos pasos solo se realizan si el alfabeto del CA de entrada y el CA de salida son diferentes. El ordenamiento busca un CA isomórfico que tenga el mayor número de elementos comodines en la esquina inferior derecha. La detección de elementos comodines busca los nuevos elementos comodines. El algoritmo heurístico se encarga de completar las t-tuplas faltantes de la matriz.

2.6 Evaluación de las Clasificaciones

Cada clasificación de los diferentes métodos para construir CAs tiene su ventaja y desventaja. Los métodos algebraicos tienen la ventaja de que construyen CAs en tiempo polinomial, la desventaja que tienen estos métodos es que cada método es aplicable a un caso específico, por ejemplo, es caso de $v = 2$ y $t = 2$. Los métodos exactos garantizan la construcción del CA óptimo, el problema de estos métodos es que realizan un búsqueda exhaustiva y solo son aplicables a CAs pequeños. Los métodos heurísticos son generales, es decir, son aplicables a cualquier valor de k, t y v, la gran desventaja de estos métodos es que no garantizan la construcción del CA y requieren de mucho tiempo de cómputo. Finalmente, los métodos de manipulación construyen CAs más adecuados para algún propósito, por ejemplo, maximizar el número de renglones constantes o incrementar el número de elementos comodines, el problema de estos métodos radica en que requieren de soluciones previamente construidas. La Tabla 2.7 muestra la ventaja y desventaja de cada clasificación de los métodos existentes.

Tabla 2.7: Tabla comparativa de las ventajas y desventajas de las clasificaciones de los métodos para construir CAs.

Clasificación	Ventaja	Desventaja
Algebraicos	Construyen CAs en tiempo polinomial siguiendo reglas predefinidas.	Sólo son aplicables a casos específicos.
Exactos	Encuentran el CA óptimo.	El tiempo de búsqueda es exponencial.
Voraces	Son rápidos.	Construyen CA no óptimos.
Heurísticos	Son generales cualquier valor de k, t y v.	Requieren de mucho tiempo y no garantizan la construcción de un CA.
Manipulación	Crean CAs más adecuados para algún propósito.	Requieren de CAs previamente construidos.

2.7 Resumen

En este capítulo, se presentaron diversos métodos para construir CAs, las cuales se clasificaron en cinco categorías: métodos algebraicos, métodos exactos, métodos voraces, métodos heurísticos y métodos de manipulación.

Los métodos exactos construyen CAs óptimos, pero el rango de valores para t, k y v es pequeño, puesto que, realizan una búsqueda exhaustiva del espacio de búsqueda. Los métodos voraces construyen CAs en un tiempo menor que los métodos exactos, estos métodos no garantizan que los CAs construidos sean óptimos. Los métodos heurísticos no garantizan la construcción de CA óptimos, en la mayoría de los casos los métodos heurísticos proveen una buena solución. Los métodos de manipulación realizan operaciones a CAs previamente construidos, tratando de mejorar la calidad de dicho CA, ya sea detectando elementos comodines, maximizando el número de renglones constantes o reducir los renglones o validar los CA. Los métodos algebraicos generalmente siguen un proceso para construir CAs a través de fórmulas matemáticas, generalmente requieren de otros diseños matemáticos como vectores, campos finitos e Galois u otros CAs.

De los métodos exactos se revisaron los métodos: algoritmo de *backtracking*, algoritmo EXACT, programación con restricciones y construcción de CAs usando una transformación al problema de satisfactibilidad. Los métodos voraces revisados fueron: herramienta AETG, el DDA, el IPO y sus variantes y *coverage inheritance*. Para los métodos heurísticos se clasificaron en tres tipos diferentes de heurísticas, estas heurísticas fueron, GA, TS y RS. Los métodos de manipulación de CA se revisaron los métodos: detección de elementos comodines, maximización de renglones constantes, reducción óptima de CAs, validación de CAs y el operador de fusión y fusión generalizada.

Dentro de los métodos algebraicos se revisaron los métodos: construcción de Bush, construcción de CA de fuerza dos y alfabeto binario, vectores de pesos constantes, construcción a través de CTs, producto de CAs de fuerza dos, construcción de CA utilizando construcciones tipo Roux, potenciación de CAs, construcción por grupos, ciclotomía y vectores de permutación. De todos lo métodos revisados dentro de los métodos algebraicos destacan dos métodos, estos métodos son: las construcción por grupos y ciclotomía. Estos métodos tienen en común el uso de un VI para construir el CA, la diferencia entre estos dos métodos es que la construcción por grupos, solo puede construir CAs de $t = 2$ de tamaño $k(v - |\mathcal{F}|) \times k$ y el método ciclotomía construye CAs cuando k es una potencia de un número primo ($k = p^{\alpha}$), el tamaño de los CAs que puede construir varia de acuerdo a las tuplas que cubre el VI, los tamaños son: $kv \times k$ y $k \times k$. Puesto que hay una gran cantidad de valores para k no explorados por estos dos métodos para $t = 2$ hasta $t = 6$, en este trabajo de tesis se propone una generalización de estos dos métodos. En el siguiente capítulo se presentará la metodología que se propone para generalizar estas dos construcciones.

3

Metodología de Construcción de CAs utilizando VIs

Al revisar los diversos métodos que existen para construir CAs, dentro de los métodos algebraicos se observa que en los métodos donde se utiliza un VI para construir el CA están limitados por los valores de k o de t. En este capítulo se presentará una generalización de esos métodos para construir CAs utilizando VIs. La generalización se enfocará en abordar los valores de k y t que no son explorados. El capítulo se divide en cuatro secciones, la sección de definiciones, la sección de metodología y la sección de un algoritmo exacto. En la sección de definiciones se encontrarán los conceptos de los operadores utilizados, *vector canónico*, *vectores inicializadores* y *órbitas*. La sección de metodología describirá la metodología propuesta que generaliza la construcción de CAs utilizando VIs. La metodología se divide en dos etapas, la etapa de búsqueda y construcción; y la etapa de manipulación. La etapa de búsqueda y construcción se compone de cuatro módulos, el módulo para fijar tuplas, el módulo que generará los vectores, el módulo para validar que los vectores son VIs y

el módulo de construcción del CA; y la etapa de manipulación la componen dos módulos, el módulo para detectar elementos comodines y el módulo para reducir renglones al CA. De cada módulo de la metodología se mostrará el algoritmo que utiliza. En la última sección se describirá el algoritmo de búsqueda exacto para la etapa de búsqueda y construcción. Al final se muestra un resumen del capítulo.

3.1 Definiciones

Para comprender mejor la metodología hay que definir algunos conceptos importantes, estos conceptos son: los operadores de *Rotación* y *Traslación*, *vectores canónicos*, *vectores inicializadores* y *órbitas*. Para los operadores de *Rotación* y *Traslación* se definirán de manera formal y se mostrará un ejemplo de cada uno de ellos. Para los *vectores inicializadores* se mostrará la importancia de utilizar varios VIs para construir un CA. Dentro del concepto de *vector canónico* se describirán las características que tienen, también se mencionarán las ventajas que ofrecen y se mostrará un ejemplo de un *vector canónico*. Y por último, se explicará una representación diferente de las órbitas a través de sus distancias y se mostrarán las fórmulas matemáticas para contar el número total de órbitas diferentes junto con la manera para saber cuales órbitas son.

3.1.1 Operadores

Para construir un CA utilizando un VI se requiere de dos operadores, la operación de *Rotación* y la *Traslación*. La operación de *Rotación* se define de la siguiente manera:

DEFINICIÓN 15 (Rotación).

Dado un vector $\mathcal{S}$ de tamaño k donde un elemento de $\mathcal{S}$ se designa por $\mathcal{S}_i$ $0 \leq i \leq k-1$, la operación de *Rotación* produce en vector $\mathcal{S}'_i = \mathcal{S}_{(i+\sigma) \bmod k}$ donde $0 \leq \sigma \leq k-1$.

EJEMPLO 1 (Ejemplo de la operación de *Rotación*).

Para $k = 5$ y $v = 3$, la operación de *Rotación* con $\sigma = 2$ sobre el vector $\mathcal{S} = \{0, 0, 0, 0, 1\}$ produce el vector $\mathcal{S}' = \{0, 1, 0, 0, 0\}$.

La operación de *Traslación* se define de la siguiente manera:

DEFINICIÓN 16 (Traslación).

Dado un vector $\mathcal{S}$ de tamaño k donde un valor particular de $\mathcal{S}$ se designa por $\mathcal{S}_i$ $0 \leq i \leq k-1$, la operación de *Traslación* sobre el vector $\mathcal{S}$ produce un vector $\mathcal{S}'_i = (\mathcal{S}_i + \delta) \bmod v$ donde $0 \leq \delta \leq v - 1$.

EJEMPLO 2 (Ejemplo de la operación de *Traslación*).

Para $k = 5$ y $v = 3$, la operación de *Traslación* con $\delta = 2$ sobre el vector $\mathcal{S} = \{0, 0, 0, 0, 1\}$ produce el vector $\mathcal{S}' = \{2, 2, 2, 2, 0\}$.

3.1.2 Vectores Inicializadores

Un VI es un vector de tamaño k el cual puede contener los elementos del conjunto $\mathbb{Z}_v$, la cantidad total de vectores para los valores de k y v son v^k. El VI cubre una cierta cantidad de t-tuplas a través de las operaciones de *Rotación* (DEFINICIÓN 15) y Traslación (DEFINICIÓN 16) en vk renglones. En algunas ocasiones, los vk renglones no son suficientes para cubrir al menos una vez las $\mathbb{Z}_v^t$ tuplas que se requieren para que sea un CA y en otras ocasiones las tuplas que cubre un VI no son suficientes para cubrir al menos una vez las $\mathbb{Z}_v^t$ tuplas necesarias. La manera para cubrir estas tuplas faltantes es utilizando más de un VI. Cada VI cubre tuplas que los otros VIs no cubren. En la Figura 3.1 se muestra un CA construido utilizando los vectores $\{0, 0, 1, 1\}$ y $\{0, 0, 0, 1\}$, produciendo un CA con 16 renglones.

$$\left(\begin{array}{cccc}
\mathbf{0} & \mathbf{0} & 1 & \mathbf{1} \\
\mathbf{1} & \mathbf{0} & 0 & \mathbf{1} \\
1 & 1 & 0 & 0 \\
\mathbf{0} & \mathbf{1} & 1 & \mathbf{0} \\
\mathbf{1} & \mathbf{1} & 0 & \mathbf{0} \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & 1 \\
\mathbf{1} & \mathbf{0} & 0 & \mathbf{0} \\
0 & 1 & 0 & 0 \\
\mathbf{0} & \mathbf{0} & 1 & \mathbf{0} \\
1 & 1 & 1 & 0 \\
\mathbf{0} & \mathbf{1} & 1 & \mathbf{1} \\
1 & 0 & 1 & 1 \\
\mathbf{1} & \mathbf{1} & 0 & \mathbf{1}
\end{array}\right)$$

Figura 3.1: $CA(16; 3, 4, 2)$. CA construido con los vectores $\{0, 0, 1, 1\}$ y $\{0, 0, 0, 1\}$, con $\delta = 0, 1$ y $\sigma = 0, 1, 2, 3$ para cada vector. Cada vector cubre tuplas que el otro vector no cubre.

3.1.3 Vectores Canónicos

Puesto que las operaciones de *Rotación* y *Traslación* pueden producir vk vectores diferentes, estos vk vectores son equivalentes entre sí y cada vector obtenido a través de estas operaciones existe en los v^k vectores diferentes, por lo tanto solo existen $\frac{v^k}{vk}$ vectores que pertenecen a soluciones diferentes. Por cada vk vectores obtenidos por las operaciones de *Rotación* y *Traslación* existe un vector que es el mas pequeño de todos, a este vector se le llama Vector Canónico (VC). La definición formal de un VC es la siguiente:

Definición 17 (Vector Canónico).

Dado un conjunto de vectores $\mathcal{E}$ de cardinalidad no mayor a vk, un vector canónico (VC) de este conjunto es un vector e tal que $e \in \mathcal{E}$ satisfaciendo:

- Inicia con una cantidad de ceros μ, $\mu \geq 1$.
- El siguiente elemento seguido de los μ ceros es 1.
- El último símbolo es diferente de 0.
- El número de ceros consecutivos intermedios no debe de ser mayor a los μ ceros iniciales.

Al utilizar los VCs se reduce el espacio de búsqueda de tamaño v^k a $\frac{v^k}{vk}$ vectores.

Ejemplo 3 (Ejemplo de un VC).

Para los valores de $k = 14$ y $v = 3$ un VC sería el siguiente: $\{0,0,1,2,1,2,2,1,1,0,2,0,2,2\}$, el cual, tiene 2 ceros iniciales, es decir, el valor de $\mu = 2$, después de los μ ceros fijos le sigue un uno, el último símbolo es diferente de cero y número de ceros consecutivos no es mayor a los μ ceros iniciales.

3.1.4 Órbitas

Para verificar que los VCs son VIs, las combinaciones de las posiciones de los VIs se dividen en grupos. Un grupo se define de la siguiente manera:

Definición 18 (Grupo).

Un grupo $\mathcal{G}$ es una pareja $(\mathbb{C}, \tau)$, donde $\mathbb{C}$ es un conjunto no vacío y τ es una operación binaria tal que $\mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C}$.

La manera en que se agrupan las combinaciones de las posiciones de los VI se realiza de tal manera que el grupo sea un grupo cíclico. Un grupo cíclico se define de la siguiente manera:

Definición 19 (Grupo Cíclico).

Un grupo $\mathbb{G}$ es cíclico si existe $g \in \mathbb{G}$ tal que $g^n \in \mathbb{G}$, donde n es un número entero.

La característica del grupo cíclico es que, al elevar a alguna potencia cualquier elemento del grupo, se pueden obtener los demás elementos del grupo. Puesto que g es un vector, al elevar el vector a una potencia se realiza un desplazamiento módulo k. Tomando un elemento de todos los grupos cíclicos del VI se verifica si la matriz resultante es un CA. A este conjunto de elementos de les llama órbitas. Una órbita se define de la siguiente manera:

Definición 20 (Órbita).

Una órbita g es un vector de tamaño t el cual es el elemento $g \in \mathbb{G}$ donde $\mathbb{G}$ es un grupo cíclico. [36]

Una órbita puede ser representada por un vector $\mathcal{X}$ de tamaño t donde un elemento en particular se denota por $\mathcal{X}_i$, $0 \leq i \leq t-1$. Cada $\mathcal{X}_i$ representa la distancia que hay entre un par de los elementos de la órbita de un vector de tamaño k. La suma de todas las distancias del vector $\mathcal{X}$ debe ser el valor de k.

Ejemplo 4 (Ejemplo del Vector de Distancias de una Órbita).

Para los valores de $k = 8$ y $t = 2$ y la órbita de las posiciones 1 y 3, la distancia que hay entre la posición 1 y la posición 3 es dos y la distancia que hay entre la posición 3 y la posición 1 es seis, por lo tanto el vector de distancias $\mathcal{X}$ sería el siguiente:

$\mathcal{X} = \{2, 6\}$

El vector $\mathcal{X}$ puede ser representado por la ecuación Diofántica (3.1), teniendo la restricción de que cada $\mathcal{X}_i$ debe de ser mayor a cero ($\mathcal{X}_i > 0$).

$$\sum_{i=0}^{t-1} \mathcal{X}_i = k \tag{3.1}$$

El número de combinaciones de k posiciones tomadas de t en t con repeticiones nos da un total de $\binom{k+t-1}{t}$ soluciones. Pero en la ecuación (3.1) se tiene una restricción adicional, que todos los elementos deben de tener al menos una unidad, para cubrir esta restricción se substrae t valores de $k+t-1$, quedando solo $k-1$ y como la posición $\mathcal{X}_{t-1}$ se obtiene con la suma de todas las $t-2$ posiciones anteriores, solo se requieren $t-1$ posiciones. Por lo tanto para la ecuación (3.1) tiene en total $\binom{k-1}{t-1}$ soluciones. Al universo de vectores se le denota por $\mathcal{H}$ y un vector en particular se denota por $\mathcal{H}^i$, donde un elemento de $\mathcal{H}^i$ se denota por $\mathcal{H}^i_m$, $0 \leq m \leq t-1$.

Ejemplo 5 (Ejemplo de todos los vectores $\mathcal{H}^i$).

Para los valores $k = 8$ y $t = 2$, el número total de vectores $\mathcal{H}^i$ es siete las cuales son:

$$\begin{array}{cccccc} \mathcal{H}^1 = \{1,7\} & \mathcal{H}^3 = \{3,5\} & \mathcal{H}^5 = \{5,3\} & & & \\ \mathcal{H}^2 = \{2,6\} & \mathcal{H}^4 = \{4,4\} & \mathcal{H}^6 = \{6,2\} & \mathcal{H}^7 = \{7,1\} \end{array}$$

EJEMPLO 6 (Ejemplo de la obtención del vector de distancias).

Si tenemos los valores de $k = 10$ y $t = 5$; una de las soluciones de (3.1) sería el vector: $\mathcal{X} = \{1, 1, 2, 3, 3\}$.

Una vector de distancias $\mathcal{H}^i$ se considera equivalente a otro vector de distancias $\mathcal{H}^j$ si $\mathcal{H}^i_m = \mathcal{H}^j_{m+\epsilon \bmod t}$ cuando $1 \leq \epsilon \leq t-1$, por lo tanto, por cada $\mathcal{H}^i$ existen cuando mucho t equivalentes (3.2), dicha ecuación es una cota inferior.

EJEMPLO 7 (Ejemplo de Vectores Diferentes).

Para los valores $k = 8$ y $t = 2$, el número total de vectores $\mathcal{H}^i$ es cuatro, los cuales son:

$$\mathcal{H}^1 = \{1, 7\} \quad \mathcal{H}^2 = \{2, 6\} \quad \mathcal{H}^3 = \{3, 5\} \quad \mathcal{H}^4 = \{4, 4\}$$

$$\phi = \frac{\binom{k-1}{t-1}}{t} \tag{3.2}$$

Pero la (3.2) no cuenta todos los vectores de distancias, puesto que, hay vectores de distancias que no tienen t vectores equivalentes. Para contar el total de vectores (ϕ), hay que completar los t equivalentes a aquellos vectores de distancias que no tengan t equivalentes. Para completar los t vectores equivalentes se separaron en tres casos, el caso cuando t es un número primo ($t = 2$, $t = 3$ y $t = 5$), el caso cuando $t = 4$ y el caso cuando $t = 6$. Las ecuaciones para contar el total de vectores se muestran a continuación:

$$\phi = \frac{\binom{k-1}{t-1}}{t} + \begin{cases} \left((t-1)\left(\left\lfloor \frac{t-k \bmod t}{t} \right\rfloor\right)\right) & \text{si } t \text{ es un número primo.} \\ \left((t-1)\left(\left\lfloor \frac{t-k \bmod t}{t} \right\rfloor\right)\right) + & \\ (t-2)\frac{\left(\binom{\frac{k}{2}-1}{\frac{t}{2}-1} - \left(\left\lfloor \frac{\frac{t}{2}-k \bmod \frac{t}{2}}{\frac{t}{2}} \right\rfloor\right)\right)}{\frac{t}{2}}((k+1) \bmod 2) & \text{si } t = 4. \\ \left((t-1)\left(\left\lfloor \frac{t-k \bmod t}{t} \right\rfloor\right)\right) + & \\ (t-2)\frac{\left(\binom{\frac{k}{3}-1}{\frac{t}{3}-1} - \left(\left\lfloor \frac{\frac{t}{3}-k \bmod \frac{t}{3}}{\frac{t}{3}} \right\rfloor\right)\right)}{\frac{t}{3}}\left(\frac{3-k \bmod 3}{3}\right) + & \\ (t-3)\frac{\left(\binom{\frac{k}{2}-1}{\frac{t}{2}-1} - \left(\left\lfloor \frac{\frac{t}{2}-k \bmod \frac{t}{2}}{\frac{t}{2}} \right\rfloor\right)\right)}{\frac{t}{2}}((k+1) \bmod 2) & \text{si } t = 6. \end{cases} \tag{3.3}$$

En el caso cuando t es un número primo solo existe un vector que no tiene ningún vector de distancias equivalentes, por lo tanto, se tienen que agregar $t-1$ vectores para completar los t vectores equivalentes. En los siguientes dos casos ($t = 4$ y $t = 6$), el vector único (sin vectores equivalentes) también existe y en las ecuaciones se cuenta más de una vez, por lo tanto, se tiene que eliminar de las demás ecuaciones.

El caso cuando $t = 4$, existen $\binom{\frac{k}{2}-1}{\frac{t}{2}-1}$ con dos vectores de distancias equivalentes, para completar los t vectores equivalentes, al total de vectores con dos vectores de distancias equivalentes se multiplica por dos (faltan dos vectores para completar los t) y se elimina el vector único.

Para el último caso ($t = 6$), hay vectores que tienen dos vectores de distancias equivalentes y también hay vectores de distancias que tienen tres vectores de distancias equivalentes. Para los vectores que tienen dos vectores de distancias equivalentes, existen $\binom{\frac{k}{3}-1}{\frac{t}{3}-1}$ vectores con dos vectores de distancias equivalentes, para completar los t vectores de distancias equivalentes, se tienen que multiplicar por $t-2$ y eliminar el vector de distancias único. Los vectores que tienen tres vectores

de distancias equivalentes son $\binom{\frac{k}{2}-1}{\frac{t}{2}-1}$, para completar los t vectores de distancias equivalentes se multiplican por $t-3$ y se elimina el vector único. A continuación se muestra un ejemplo de cada t.

EJEMPLO 8 (Ejemplo de las órbitas para $t=2$).

Para los valores de $t=2$ y $k=6$, los vectores de distancias que se deben de cubrir son:

$$\mathcal{H}^1 = \{1,5\} \quad \mathcal{H}^2 = \{2,4\} \quad \mathcal{H}^3 = \{3,3\}$$

La ecuación (3.2) da como resultado 2.5, puesto que el vector $\mathcal{H}^3$ solo tiene un vector equivalente. utilizando la ecuación (3.3) da como resultado 3.

EJEMPLO 9 (Ejemplo de las órbitas para $t=3$).

Para los valores de $t=3$ y $k=6$, los vectores de distancias que se deben de cubrir son:

$$\mathcal{H}^1 = \{1,1,4\} \quad \mathcal{H}^2 = \{1,2,3\} \quad \mathcal{H}^3 = \{1,3,2\} \quad \mathcal{H}^4 = \{2,2,2\}$$

La ecuación (3.2) da como resultado 3.333, puesto que el vector $\mathcal{H}^4$ solo tiene un vector equivalente. utilizando la ecuación (3.3) da como resultado 4.

EJEMPLO 10 (Ejemplo de las órbitas para $t = 4$).

Para los valores de $t = 4$ y $k = 8$, los vectores de distancias que se deben de cubrir son:

$$\begin{array}{llll} \mathcal{H}^1 = \{1,1,1,5\} & \mathcal{H}^4 = \{1,1,3,3\} & \mathcal{H}^7 = \{1,1,4,2\} \\ \mathcal{H}^2 = \{1,1,2,4\} & \mathcal{H}^5 = \{1,2,2,3\} & \mathcal{H}^8 = \{1,2,3,2\} \\ \mathcal{H}^3 = \{1,2,1,4\} & \mathcal{H}^6 = \{1,3,1,3\} & \mathcal{H}^9 = \{1,3,2,2\} \\ & & \mathcal{H}^{10} = \{2,2,2,2\} \end{array}$$

La ecuación (3.2) da como resultado 8.75, puesto que el vector $\mathcal{H}^6$ solo tiene dos vectores equivalentes y el vector $\mathcal{H}^{10}$ solo tiene un vector equivalente. utilizando la ecuación (3.3) da como resultado 10.

EJEMPLO 11 (Ejemplo de las órbitas para $t = 5$).

Para los valores de $t = 5$ y $k = 10$, los vectores de distancias que se deben de cubrir son:

$$\begin{array}{llll} \mathcal{H}^1 = \{1,1,1,1,6\} & \mathcal{H}^{10} = \{1,1,2,3,3\} & \mathcal{H}^{19} = \{1,1,3,3,2\} \\ \mathcal{H}^2 = \{1,1,1,2,5\} & \mathcal{H}^{11} = \{1,2,1,3,3\} & \mathcal{H}^{20} = \{1,2,2,3,2\} \\ \mathcal{H}^3 = \{1,1,2,1,5\} & \mathcal{H}^{12} = \{1,1,3,2,3\} & \mathcal{H}^{21} = \{1,3,1,3,2\} \\ \mathcal{H}^4 = \{1,1,1,3,4\} & \mathcal{H}^{13} = \{1,2,2,2,3\} & \mathcal{H}^{22} = \{1,1,4,2,2\} \\ \mathcal{H}^5 = \{1,1,2,2,4\} & \mathcal{H}^{14} = \{1,1,4,1,3\} & \mathcal{H}^{23} = \{1,2,3,2,2\} \\ \mathcal{H}^6 = \{1,2,1,2,4\} & \mathcal{H}^{15} = \{1,2,3,1,3\} & \mathcal{H}^{24} = \{1,3,2,2,2\} \\ \mathcal{H}^7 = \{1,1,3,1,4\} & \mathcal{H}^{16} = \{1,1,1,5,2\} & \mathcal{H}^{25} = \{1,1,5,1,2\} \\ \mathcal{H}^8 = \{1,2,2,1,4\} & \mathcal{H}^{17} = \{1,1,2,4,2\} & \mathcal{H}^{26} = \{2,2,2,2,2\} \\ \mathcal{H}^9 = \{1,1,1,4,3\} & \mathcal{H}^{18} = \{1,2,1,4,2\} & \end{array}$$

La ecuación (3.2) da como resultado 25.2, puesto que el vector $\mathcal{H}^{26}$ solo tiene un vector equivalente. utilizando la ecuación (3.3) da como resultado 26.

EJEMPLO 12 (Ejemplo de las órbitas para $t = 6$).

Para los valores de $t = 6$ y $k = 10$, los vectores de distancias que se deben de cubrir son:

$$\begin{array}{llll} \mathcal{H}^1 = \{1,1,1,1,1,5\} & \mathcal{H}^9 = \{1,1,2,2,1,3\} & \mathcal{H}^{16} = \{1,1,2,2,2,2\} \\ \mathcal{H}^2 = \{1,1,1,1,2,4\} & \mathcal{H}^{10} = \{1,2,1,2,1,3\} & \mathcal{H}^{17} = \{1,2,1,2,2,2\} \\ \mathcal{H}^3 = \{1,1,1,2,1,4\} & \mathcal{H}^{11} = \{1,1,3,1,1,3\} & \mathcal{H}^{18} = \{1,1,3,1,2,2\} \\ \mathcal{H}^4 = \{1,1,2,1,1,4\} & \mathcal{H}^{12} = \{1,1,1,1,4,2\} & \mathcal{H}^{19} = \{1,2,2,1,2,2\} \\ \mathcal{H}^5 = \{1,1,1,1,3,3\} & \mathcal{H}^{13} = \{1,1,1,2,3,2\} & \mathcal{H}^{20} = \{1,1,1,4,1,2\} \\ \mathcal{H}^6 = \{1,1,1,2,2,3\} & \mathcal{H}^{14} = \{1,1,2,1,3,2\} & \mathcal{H}^{21} = \{1,1,2,3,1,2\} \\ \mathcal{H}^7 = \{1,1,2,1,2,3\} & \mathcal{H}^{15} = \{1,1,1,3,2,2\} & \mathcal{H}^{22} = \{1,1,3,2,1,2\} \\ \mathcal{H}^8 = \{1,1,1,3,1,3\} & & \end{array}$$

La ecuación (3.2) da como resultado 21, puesto que el vector $\mathcal{H}^{19}$ solo tiene tres vectores equivalentes. utilizando la ecuación (3.3) da como resultado 22.

3.2 Metodología de Construcción de CAs utilizando VIs

La metodología generalizada que se propone para la resolución del problema de construcción de VIs está dividida en dos etapas, la etapa de búsqueda y construcción; y la etapa de manipulación.

La etapa de búsqueda y construcción está compuesta por cuatro módulos, el módulo de fijar tuplas, el módulo de generación de vectores, el módulo de validación de vectores y el módulo de construcción del CA. En la etapa de búsqueda y construcción se buscarán todos los VIs que construyan un CA, estos CAs se pasarán a la siguiente etapa.

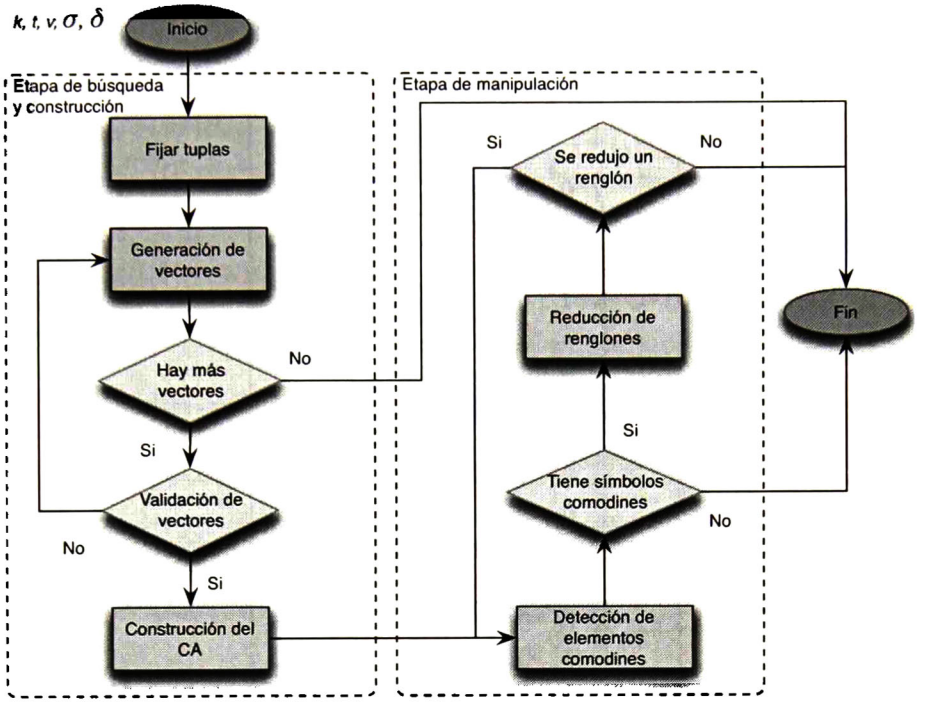


Figura 3.2: Diagrama de la metodología. En el diagrama de pueden observar las dos etapas que componen la metodología, **a)** la etapa de búsqueda y construcción; y **b)** la etapa de manipulación. Dentro de la etapa de búsqueda y construcción se observan los módulos de fijar tuplas, generación de vectores, validación de vectores y construcción del CA y dentro de la etapa de manipulación se observan los módulos de detección de elementos comodines y reducción de renglones.

La etapa de manipulación la componen dos módulos, el módulo de detección de elementos comodines y el módulo de reducción de renglones. La etapa de manipulación se enfocará en reducir los renglones de los CAs detectando los elementos comodines y tratando de reducir los renglones usando los elementos comodines. La Figura 3.2 muestra los módulos de la metodología.

3.2.1 Fijar tuplas

Las órbitas se pueden utilizar para diversos propósitos, uno de estos propósitos es el de verificar si los VCs son VIs y otro propósito es el de fijar algunas tuplas y posteriormente agregar las tuplas

que se fijaron a la matriz resultante de las operaciones de *Rotación* y *Traslación* para construir el CA.

$$\left(\begin{array}{ccccc}
0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
\hline
0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 1
\end{array}\right)$$

Figura 3.3: $CA(22; 4, 5, 2)$. CA construido utilizando los vectores $\{0, 1, 0, 1, 1\}$ y $\{0, 0, 0, 1, 1\}$, fijando las tuplas constantes y posteriormente agregando los 2 renglones constantes.

Unas tuplas útiles para agregar son las t-tuplas constantes. Las t-tuplas constantes son aquellas t-tuplas que tienen el mismo símbolo en las t posiciones de la tupla. Para cubrir estas t-tuplas solo es necesario agregar v renglones, donde cada renglón contiene k veces un símbolo del conjunto $\mathbb{Z}_v$. Por ejemplo, para los valores de $k = 5$, $v = 2$ y $t = 4$, con los vectores $\{0, 1, 0, 1, 1\}$ y $\{0, 0, 0, 1, 1\}$ se construye una matriz de 20 renglones, a esta matriz le faltan las tuplas constantes, al agregar los dos renglones constantes a la matriz resultante, se se cubren todas las tuplas del conjunto $\mathbb{Z}_v^t$ (Figura 3.3).

$$\left(\begin{array}{ccccccccc}
0 & 1 & 2 & 1 & 0 & 3 & 2 & 3 & 3 \\
3 & 0 & 1 & 2 & 1 & 0 & 3 & 2 & 3 \\
3 & 3 & 0 & 1 & 2 & 1 & 0 & 3 & 2 \\
2 & 3 & 3 & 0 & 1 & 2 & 1 & 0 & 3 \\
3 & 2 & 3 & 3 & 0 & 1 & 2 & 1 & 0 \\
0 & 3 & 2 & 3 & 3 & 0 & 1 & 2 & 1 \\
1 & 0 & 3 & 2 & 3 & 3 & 0 & 1 & 2 \\
2 & 1 & 0 & 3 & 2 & 3 & 3 & 0 & 1 \\
1 & 2 & 1 & 0 & 3 & 2 & 3 & 3 & 0 \\
0 & 1 & 3 & 1 & 0 & 2 & 3 & 2 & 2 \\
2 & 0 & 1 & 3 & 1 & 0 & 2 & 3 & 2 \\
2 & 2 & 0 & 1 & 3 & 1 & 0 & 2 & 3 \\
3 & 2 & 2 & 0 & 1 & 3 & 1 & 0 & 2 \\
2 & 3 & 2 & 2 & 0 & 1 & 3 & 1 & 0 \\
0 & 2 & 3 & 2 & 2 & 0 & 1 & 3 & 1 \\
1 & 0 & 2 & 3 & 2 & 2 & 0 & 1 & 3 \\
3 & 1 & 0 & 2 & 3 & 2 & 2 & 0 & 1 \\
1 & 3 & 1 & 0 & 2 & 3 & 2 & 2 & 0 \\
\hline
1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right)$$

Figura 3.4: $CA(24; 2, 9, 4)$. CA construido con los vectores $\{0, 1, 2, 1, 0, 3, 2, 3, 3\}$ y $\{0, 1, 3, 1, 0, 2, 3, 2, 2\}$, los cuales son equivalentes al vector $\{0, 1, 2, 1, 0, 3, 2, 3, 3\}$ con dos símbolos fijos.

Otras t-tuplas que también son útiles para fijar, son las t-tuplas de un conjunto $\mathbb{Z}_v$ menor al del CA a construir, puesto que, al agregar un CA de ese conjunto $\mathbb{Z}_v$ se cubren todas las combinaciones que se fijaron. El fijar tuplas de un conjunto $\mathbb{Z}_v$ menor, es equivalente a fijar símbolos propuesta por Lobb *et al.* [37].

Teorema 2.

Si existe un CA construido con un VI con $\mathcal{F}$ símbolos fijos entonces existen $v - |\mathcal{F}|$ VCs equivalentes con $\delta = 0$.

Algoritmo 1 Algoritmo para fijar las tuplas que se agregarán posteriormente al CA.

```
1: procedure FIXTUPLAS(P, φ, constantes, F)
2:     I es un vector de tamaño t
3:     if constantes está activa then
4:         for i = 0; i < v; i++ do
5:             for j = 0; j < t; j++ do
6:                 I_j ← i
7:             end for
8:             for j = t − 1; j ≥ t; j−− do
9:                 pt ← pt + I_j v^j
10:            end for
11:            for j = 0; j < φ; j++ do
12:                for l = 0; l < k; l++ do
13:                    P_{pt,j,l} ← P_{pt,j,l} + 1
14:                end for
15:            end for
16:        end for
17:    end if
18:    if F > 0 then
19:        α ← F^t
20:        for i = 0; i < α; i++ do
21:            for j = t − 1; j ≥ t; j−− do
22:                pt ← pt + I_j v^j
23:            end for
24:            for j = 0; j < φ; j++ do
25:                for l = 0; l < k; l++ do
26:                    P_{pt,j,l} ← P_{pt,j,l} + 1
27:                end for
28:            end for
29:            for i = t − 1; i ≥ 0; i−− do
30:                I_i ← I_i + 1
31:                if I_i == F then
32:                    I_i ← 0
33:                else
34:                    break
35:                end if
36:            end for
37:        end for
38:    end if
39: end procedure
```

Puesto que la operación de *Rotación* al ser aplicada al VI con $|\mathcal{F}|$ símbolos fijos produce $v - |\mathcal{F}|$ vectores en donde los símbolos fijos no son afectados por esta operación, cada vector resultante de la operación es un VC con $\delta = 0$. Por ejemplo, para los valores $k = 9$,$v = 4$ y $t = 2$; y los vectores $\{0, 1, 2, 1, 0, 3, 2, 3, 3\}$ y $\{0, 1, 3, 1, 0, 2, 3, 2, 2\}$ se obtiene una matriz de 18 renglones a la que le faltan las combinaciones $(0, 0)$, $(0, 1)$, $(1, 0)$ y $(1, 1)$, agregando el $CA(6; 2, 9, 2)$ se construye

el $CA(24; 2, 9, 4)$, estos vectores son equivalentes al vector $\{0, 1, 2, 1, 0, 3, 2, 3, 3\}$ con los símbolos 0 y 1 fijos (Figura 3.4).

El Algoritmo 1 muestra el procedimiento para fijar las t-tuplas. El algoritmo requiere lo siguiente:

- La estructura $\mathcal{P}$ para fijar las t-tuplas.
- El valor de ϕ.
- La bandera de *constantes* para fijar las tuplas constantes,
- El número de símbolos de las t-tuplas a fijar $\mathcal{F}$.

En la línea 2 se declara la estructura I de tamaño t y en la línea 3 se verifica si la bandera *constantes* esta activa. En las líneas 5-7 se asigna el símbolo i a cada posición de I. En las líneas 8-10 se transforma la t-tupla a su representación numérica, en las líneas 12-15 se fija la t-tupla en la estructura $\mathcal{P}$. En la línea 18 se verifica si hay símbolos para fijar. En la línea 19 se asigna a α el total de t-tuplas a fijar. En las líneas 21-23 se transforma la t-tupla a su valor numérico y en las líneas 24-28 se fija la t-tupla en la estructura $\mathcal{P}$. En las líneas 29-36 se genera la siguiente t-tupla a fijar.

3.2.2 Generación de Vectores

El módulo de generación de vectores se encarga de generar los vectores para después ser verificados en el módulo de validación. El módulo generará solo VCs utilizando el código Gray [79] v-ario, (v es el alfabeto del CA). Un ejemplo del código Gray se muestra a continuación:

EJEMPLO 13 (Ejemplo de la Codificación Gray).

Si tenemos los valores de $k = 4$ y $v = 2$, los vectores a verificar utilizando el código Gray serian:

$$\{0,0,0,0\}\ \{0,\mathbf{1},1,0\}\ \{\mathbf{1},1,0,0\}\ \{1,\mathbf{0},1,0\}$$
$$\{0,0,0,\mathbf{1}\}\ \{0,1,1,\mathbf{1}\}\ \{1,1,0,\mathbf{1}\}\ \{1,0,1,\mathbf{1}\}$$
$$\{0,0,\mathbf{1},1\}\ \{0,1,\mathbf{0},1\}\ \{1,1,\mathbf{1},1\}\ \{1,0,\mathbf{0},1\}$$
$$\{0,0,1,\mathbf{0}\}\ \{0,1,0,\mathbf{0}\}\ \{1,1,1,\mathbf{0}\}\ \{1,0,0,\mathbf{0}\}$$

Al generar los z VCs utilizando el código Gray se puede realizar la verificación de los vectores (módulo de verificación) de manera más ágil, puesto que, el cambio entre los vectores verificados anteriormente y los nuevos vectores a verificar, solo cambiaron en una sola posición.

El Algoritmo 2 muestra el procedimiento para generar los VCs. El algoritmo requiere de la estructura SV de tamaño $z \times k$ y los valores de μ, z y k. En la línea 2 se generan los z vectores en SV utilizando el algoritmo propuesto en [80]. En la línea 3 se valida que los vectores se encuentren ordenados de menor a mayor, eso se hace para evitar no considerar los mismos vectores en orden diferente. De la línea 5 hasta la línea 34, se verifican las características de los VCs (a excepción de la característica el 1 seguido de los μ ceros iniciales). Este proceso se repite mientras los vectores no estén ordenados de menor a mayor, que los vectores no sean VCs y que existan mas VCs. Al final se regresa un **True** si se generaron z VCs o se regresa **False** se hay no hay mas VCs para ese valor de μ.

Para ejemplificar el Algoritmo 2 tenemos los valores de $z = 1$, $k = 5$, $v = 2$ y $\mu = 2$; y la estructura SV con el vector $SV = \{0,0,0,0,1\}$. Primero se genera el vector $SV = \{0,0,0,1,1\}$ con el código Gray, luego se verifica que los z vectores estén ordenados de mayor a menor, puesto

Algoritmo 2 Algoritmo para generar los VCs.

```
1: procedure NEXTVC(SV, μ, k, v, z)
2:     repeat
3:         Generación de los z vectores usando el código Gray v-ario
4:         Validar SV_i < SV_{i+1} donde 0 ≤ i ≤ z − 1
5:         for i = 0; i < z; i++ do
6:             for j = zc = 0; SV_{i,j} ≡ 0 and j < k; j++ do
7:                 zc ← zc + 1
8:             end for
9:             if zc ≥ μ then
10:                for j = zc, zc = 0; j < k; j++ do
11:                    if SV_{i,j} ≡ 0 then
12:                        zc ← zc + 1
13:                    else if zc < μ then
14:                        zc ← μ + 1
15:                        break
16:                    else
17:                        zc ← 0
18:                    end if
19:                end for
20:            end if
21:            if zc > 0 then
22:                vzc ← True
23:            else
24:                vzc ← False
25:            end if
26:            if sv_{i,k−1} ≡ 0 then
27:                lsdz ← True
28:            else
29:                lsdz ← False
30:            end if
31:            if lsdz or vzc then
32:                break
33:            end if
34:        end for
35:    until (SV_i ≤ SV_{i+1} donde 0 ≤ i ≤ z − 1 or lsdz or vzc) and existan más VCs
36:    if posición cambiada es ≤ μ then
37:        return False
38:    else
39:        return True
40:    end if
41: end procedure
```

que sólo es uno, esta condición es válida. contamos el número de ceros iniciales de cada vector y verificamos si el número de ceros sea mayor a los μ ceros iniciales, después se verifica que el número de ceros intermedios después de los μ ceros iniciales no sea mayor a los μ ceros iniciales, inmediatamente después se verifica que el último símbolo sea diferente de cero, mientras alguna de estas condiciones no se cumpla, se generarán vectores hasta que se encuentre uno que cumpla con todas las características de los VCs. Puesto que el vector $\{0,0,0,1,1\}$ cumple con todas las características, se regresa este vector.

3.2.3 Validación

La validación de los VCs generados por el módulo anterior se realiza verificando las órbitas de dichos vectores. Para cada VC se verifica que órbitas cubre y se van almacenando en una matriz $\mathcal{P}$ de tamaño $v^t \times \phi$. Para verificar las órbitas de los VCs se desarrolló un procedimiento llamado *Recálculo Local* (RL). El Algoritmo 3 muestra el procedimiento general para realizar el RL. El algoritmo requiere los valores de z, t, δ y σ; y de las estructuras:

- SV de tamaño $z \times k$, en este arreglo se almacenan los z VCs a verificar.
- PSV de tamaño $z \times k$, en este arreglo se almacenan los z VCs que se verificaron anteriormente.
- $\mathcal{P}$ de tamaño $v^t \times \phi \times k$ para almacenar las t-tuplas que se cubren con las órbitas.
- $\mathcal{O}$ de tamaño $\phi \times t$ almacena todos los vectores de distancias de las órbitas.

El RL consta de cinco pasos, en el primer paso se comparan las estructuras SV y PSV para detectar que posiciones cambiaron, este paso se muestra en las líneas 2 a 4. Una vez que se detectó una posición, en el paso dos se realiza un ajuste decrementar de las órbitas con las t-tuplas en las que participa esa posición, la línea 5. La línea 6 muestra el paso tres, el cual consiste en actualizar el nuevo valor en la estructura PSV El paso cuatro se muestra en la línea 7, en este paso se realiza el ajuste incremental de las órbitas con las t-tuplas en las que participa esa posición

Algoritmo 3 Algoritmo validar que los z VCs son VIs.

```
1: procedure isSV(SV, PSV. P, O, z, t, δ, σ, φ)
2:     for i = 0; i < z; i++ do                                  ▷ Paso 1
3:         for j = 0; x < k; j++ do
4:             if PSV_{i,j} ≠ SV_{i,j} then
5:                 Ajuste(PSV, P, O, i, t, j, DEC, δ, σ, φ)      ▷ Paso 2
6:                 PSV_{i,j} = SV_{i,j}                          ▷ Paso 3
7:                 Ajuste(PSV, P, O, i, t, j, INC, δ, σ, φ)      ▷ Paso 4
8:             end if
9:         end for
10:    end for
11:    for i = 0; i < v^t; i ++ do                               ▷ Paso 5
12:        for j = 0; j < φ; j ++ do
13:            if P_{i,j} = 0 then
14:                return False
15:            end if
16:        end for
17:    end for
18:    return True
19: end procedure
```

con el valor nuevo. En las líneas 11 a 1 se muestra el paso cinco, este paso verifica en la estructura $\mathcal{P}$ no contenga ninguna entrada con el valor 0, de ser así, se regresa el valor de **True** indicando que los nuevos VCs contienen todas las tuplas en todas las órbitas, en caso contrario regresa el valor de **False**. Para hacer el ajuste de incremento o decremento, se muestra en el Algoritmo 4, este algoritmo requiere de las estructuras PSV, $\mathcal{P}$ y $\mathcal{O}$; y los valores vc, t, a, OP, δ y σ, el parámetro vc indica el VC que sufrió un cambio, el parámetro t indica la fuerza, el parámetro a la posición que cambió, y los parámetros δ y σ indican las rotaciones y traslaciones a realizar. En las líneas 2 y 3 el algoritmo declara las estructuras:

- I es un arreglo de tamaño t para almacenar la t-tupla que se modificó.
- D es un arreglo de tamaño t para almacenar el vector de distancias de la órbita.
- S es un arreglo de tamaño t para realizar la *Traslación* cuando se usan símbolos fijos.

En la línea 4 se recorren todas las órbitas y en la línea 5 se toma la órbita i almacenada en $\mathcal{O}$ y se almacena en D, posteriormente, en la línea 6 se recorren todas las rotaciones del vector de distancias de la órbita i y en la línea 7 se realizan las rotaciones del vector D. En la línea 10 se recorren todas las traslaciones de la tupla contenida en las distancias D. De la línea 11 a la 14, se toma la t-tupla contenida en el vector vc a partir de la posición a y en las líneas 15 a 17 se traslada la tupla de acuerdo a su δ. En las líneas 18 a la 25 se realiza la traslación de la tupla cuando $|\mathcal{F}| > 0$. En las líneas 26 a 36 se realizan las rotaciones de la tupla y de acuerdo a OP se realiza el incremento o decremento de la t-tupla.

EJEMPLO 14 (Ejemplo del algoritmo isSV).

Tenemos los valores $z = 1$, $t = 2$, $\phi = 2$, $\delta_0 = \{0,1\}$ y $\sigma_0\{01,2,3,4\}$; en el arreglo $SV = \{0,0,0,1,1\}$, el arreglo $PSV = \{0,0,0,0,0\}$, la matriz $\mathcal{P}$ de tamaño $4 \times 2 \times 5$, en $\mathcal{P}_{0,0} = \{5,5,5,5,5\}$, en $\mathcal{P}_{0,1} = \{5,5,5,5,5\}$, en $\mathcal{P}_{1,0} = \{0,0,0,0,0\}$, en $\mathcal{P}_{1,1} = \{0,0,0,0,0\}$, en $\mathcal{P}_{2,0} = \{0,0,0,0,0\}$, en $\mathcal{P}_{2,1} = \{0,0,0,0,0\}$, en $\mathcal{P}_{3,0} = \{5,5,5,5,5\}$ y en $\mathcal{P}_{3,1} = \{5,5,5,5,5\}$ y en el arreglo $\mathcal{O}$ de tamaño 2×2 tenemos los vectores de distancia $\mathcal{O}_0 = \{1,4\}$ y en $\mathcal{O}_1 = \{2,3\}$. Se actualiza $\mathcal{P}$ eliminando y luego agregando la t-tuplas en las que participa las posiciones cambiadas

PSV=$\{0,0,0,\mathbf{0},\mathbf{0}\}$

SV=$\{0,0,0,\mathbf{1},\mathbf{1}\}$

$$\mathcal{P}_0 = \{\{3,3,3,3,3\},\{1,1,1,1,1\}\}$$
$$\mathcal{P}_1 = \{\{2,2,2,2,2\},\{4,4,4,4,4\}\}$$
$$\mathcal{P}_2 = \{\{2,2,2,2,2\},\{4,4,4,4,4\}\}$$
$$\mathcal{P}_3 = \{\{3,3,3,3,3\},\{1,1,1,1,1\}\}$$

Puesto que el arreglo $\mathcal{P}$ no contiene ninguna entrada con el valor de 0, se regresa **True**.

Algoritmo 4 Algoritmo que ajusta las órbitas de los μ VCs.

```
1: procedure AJUSTE(PSV, P, O, vc, t, a, OP, δ, σ, φ)
2:     I es un arreglo de tamaño t
3:     D es un arreglo de tamaño t
4:     for i = 0; i < φ; i++ do
5:         D ← O_i
6:         for j = 0; j < t; j++ do                    ▷ Rotar distancias de la órbita
7:             for x = 0; x < t; x++ do
8:                 D_x ← D_{x+j}
9:             end for
10:            for l = 0; l < |δ_vc|; l++ do           ▷ Trasladar tupla y ajustar tuplas cubiertas
11:                for x = 0; x < t; x++ do
12:                    m ← D_x
13:                    I_x ← PSV_{vc,m+a mód k}
14:                end for
15:                for x = 0; x < t; x++ do
16:                    I_x ← D_x + δ_{vc,l} mód v
17:                end for
18:                if |F| > 0 then
19:                    for x = 0; x < t; xx++ do
20:                        if S_x == True or I_x < |F| then
21:                            S_x = True
22:                            I_x = I_x + |F|
23:                        end if
24:                    end for
25:                end if
26:                for x = 0; x < |σ|_v c; x++ do
27:                    for m = t − 1; m ≥ 0; m++ do
28:                        pt+ = I_m v^m
29:                    end for
30:                    tmp ← σ_{vc,x}
31:                    if OP ≡ DEC then
32:                        P_{pt,i,tmp+q} ← P_{pt,i,tmp+q} − 1
33:                    else
34:                        P_{pt,i,tmp+q} ← P_{pt,i,tmp+q} + 1
35:                    end if
36:                end for
37:            end for
38:        end for
39:    end for
40: end procedure
```

3.2.4 Transformación de VIs a CA

En el trabajo de Lobb *et al.* [37] utilizan las operaciones de *Rotación* (DEFINICIÓN 15) y *Traslación* (DEFINICIÓN 16), aplicando la *Traslación* v veces con $\delta = 1, \ldots, v-1$ y por cada

Traslación aplicando la *Rotación* k veces con $\sigma = 1, \ldots, k-1$, dando como resultado un CA de tamaño $vk \times k$. Las operaciones que se utilizan en este trabajo son las utilizadas por Lobb *et al.* , variando los valores para δ y σ para cada VI encontrado. El Algoritmo 5 muestra el procedimiento para transformar los VIs a CA. El algoritmo requiere de los siguientes arreglos:

- El arreglo δ la cual contiene las traslaciones de cada VI.
- El arreglo σ contiene las rotaciones de cada VI
- El arreglo SV contiene los z VIs.

En las líneas 2 y 3 el algoritmo declara el archivo de salida $\mathcal{F}$ y el arreglo $\mathcal{T}$ de tamaño k, el cual contendrá el VI para aplicarle las rotaciones y traslaciones correspondientes. En la línea 4 se recorren los z VIs que están contenidos en SV. En la línea 6, se copia el i-ésimo VI de SV a $\mathcal{T}$. De la línea 7 hasta la línea 9 se traslada el vector $\mathcal{T}$ de acuerdo a su $\delta_{i,j}$. Desde la línea 10 hasta la línea 15, se rota el vector $\mathcal{T}$ de acuerdo a su $\sigma_{i,l}$ y se guarda en el archivo de salida $\mathcal{F}$.

EJEMPLO 15 (Ejemplo del algoritmo TRANSFORMACIÓN).

Tenemos $SV = \{0,0,0,1,1\}$, $\delta = \{0,1\}$ y $\sigma = \{0,1,2,3,4\}$. Construimos una matriz de acuerdo a δ y σ, la cual quedaría de la siguiente manera:

$$\left(\begin{array}{ccccc} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{array}\right)$$

Algoritmo 5 Algoritmo de transformación de VIs a CA.

```
1: procedure TRANSFORMACIÓN(δ, σ, SV)
2:     F es el archivo de salida
3:     T es un arreglo de tamaño k
4:     for i = 0; i < z; i++ do
5:         for j = 0; j < |δ_i|; j+ do
6:             T ← SV_i
7:             for l = 0; l < k; l++ do                ▷ Trasladar vector
8:                 T_l = T_l + δ_{i,j}
9:             end for
10:            for l = 0; l < |σ_i|; l++ do            ▷ Rotar vector
11:                for u = 0; u < k; u++ do
12:                    a ← σ_{i,l}
13:                    F ← T_{(u+a) mód k}
14:                end for
15:            end for
16:        end for
17:    end for
18: end procedure
```

3.2.5 Detección de Elementos Comodines

Al terminar la etapa de búsqueda y construcción se inicia la etapa de manipulación, la etapa de manipulación tratará de reducir los renglones de los CAs construidos en la etapa anterior. Una opción para reducir los renglones es detectar los CAs que contengan elementos comodines con el módulo de detección de elementos comodines. El módulo de detección de elementos comodines consta de cuatro pasos:

1. Encontrar las t-tuplas que sólo se repiten una sola vez en el CA.

2. Fijar las t-tuplas que sólo se repiten una sola vez.

3. Encontrar las t-tuplas que aparecen más de una vez.

4. De las t-tuplas que aparecen más de una vez, seleccionar aquellas que maximicen los elementos comodines y que siga siendo un CA.

Algoritmo 6 Algoritmo para la detección de elementos comodines.

procedure DETECCIÓN(CA)
  $\mathcal{T}$ es un arreglo de tamaño v^t
  $\mathcal{B}$ es un arreglo de tamaño $N \times k$
  $\mathcal{F}$ es un arreglo de tamaño $N \times k$
  $\mathcal{F}'$ es un arreglo de tamaño $N \times k$
  inicializar $\mathcal{F}$ con -1
  $u \leftarrow 0$
  for all $c \in \binom{k}{t}$ **do**
    for $i = 0; i < N; i++$ **do**
      $m \leftarrow CA_{ic}$
      $\mathcal{T}_m \leftarrow \mathcal{T}_m + 1$
    end for
    for $i = 0; i < N; i++$ **do**
      for all $m \in \mathcal{T}$ **do**
        if $m > 1$ **then**
          $\mathcal{F}_m \leftarrow CA_{im}$
        end if
      end for
    end for
  end for
  for all $c \in \binom{k}{t}$ **do**
    for $i = 0; i < N; i++$ **do**
      $m \leftarrow \mathcal{F}_{ic}$
      if $m \equiv -1$ **then**
        $\mathcal{R} \leftarrow m$
      end if
    end for
  end for
  for all $c \in \binom{k}{t}$ **do**
    $\mathcal{F}' \leftarrow \mathcal{F}$
    Seleccionar un $q \in \mathcal{R}_c$
    $\mathcal{F}'_{qc} \leftarrow CA_{qc}$
    for $i = 0; i < N; i++$ **do**
      for $j = 0; j < k; k++$ **do**
        if $\mathcal{F}'_{i,j} = -1$ **then**
          $\kappa \leftarrow \kappa + 1$
        end if
      end for
    end for
    if $u < \kappa$ **then**
      $\mathcal{B} \leftarrow \mathcal{F}'$
      $u \leftarrow \kappa$
    end if
  end for
  return $\mathcal{B}$
end procedure

En el Algoritmo 6 muestra el procedimiento para detectar elementos comodines. En las líneas 2-6 el algoritmo declara los siguientes arreglos auxiliares.

- $\mathcal{T}$, es un arreglo de tamaño v^t para contar las veces que aparece cada tuplas en el CA.
- $\mathcal{B}$, es un arreglo de tamaño $N \times k$ en la cual se almacenará el CA que maximice los elementos comodines.
- $\mathcal{F}$, es un arreglo de tamaño $N \times k$ en donde se fijarán las tuplas que solo están una sola vez en el CA.
- $\mathcal{F}'$, es un arreglo de tamaño $N \times k$ en la que se probarán las diferentes posibles tuplas que contienen posibles elementos comodines.

En la línea 6 se inicializa la matriz $\mathcal{F}$ en -1, esto con el fin de poder detectar los posibles elementos comodines. La línea 7 se inicializa una variable llamada u, esta variable contará los elementos comodines de la mejor solución. De las líneas 8-20 se encuentran las tuplas que sólo aparecen una vez en el CA y se fijan en la matriz $\mathcal{F}$. En las siguientes ocho líneas (líneas 21-28) se encuentran las tuplas que contienen elementos comodines y en las líneas 29-44 se buscan aquellas tuplas que maximicen los elementos comodines y que siga siendo un CA.

Para ejemplificar el algoritmo anterior, suponemos que tenemos un $CA(7; 2, 11, 2)$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Se recorren las $\binom{11}{2}$ combinaciones para fijar las 2-tuplas que solo aparecen una sola vez. La matriz $\mathcal{F}$ queda de la siguiente manera:

$$\mathcal{F} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & \mathbf{-1} & \mathbf{-1} \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

En este caso, no hay 2-tuplas que sean posibles elementos comodines y que aparezcan más de dos veces, por lo tanto, $\mathcal{R}$ está vacío. Puesto que $\mathcal{R}$ está vacío, los elementos que están en -1 en $\mathcal{F}$ son elementos comodines.

3.2.6 Reducción de Renglones

Una vez que se han detectado todos los elementos comodines, se inicia el proceso para reducir renglones. El proceso recorre todos los renglones del CA buscando aquellos renglones que se puedan reemplazar los elementos comodines por los elementos de los renglones y así eliminar esos renglones seleccionados. En caso de que exista un renglón con todos sus elementos comodines, este renglón se elimina. A este proceso se le llamó REDUCCIÓN (Algoritmo 7).

3.3 Algoritmo de Búsqueda Exacto

Para construir los VIs se diseño un algoritmo de búsqueda exhaustiva de la etapa de búsqueda y construcción. El Algoritmo 8 toma como entrada los valores de k, t, v, z, los valores δ y σ para cada vector, la opción *constantes* para fijar las t-tuplas constantes y el valor de $\mathcal{F}$. El algoritmo exacto consta de tres pasos, el primero calcula ϕ con el número de órbitas e inicializa las estructuras:

- SV de tamaño $z \times k$.
- PSV de tamaño $z \times k$.
- $\mathcal{P}$ de tamaño $v^t \times \phi \times k$.

Algoritmo 7 Algoritmo del módulo de reducción de renglones.

```
1: procedure REDUCCIÓN(CA)
2:     M es un arreglo de tamaño k
3:     B ← CA
4:     repeat
5:         for i = 0; i < N; i++ do
6:             C ← B
7:             for j = 0; j < N; j++ do
8:                 s ← False
9:                 for l = 0; l < |M|; l++ do
10:                    if M_l ≡ j then
11:                        s ← True
12:                    end if
13:                end for
14:                if s == False then
15:                    for l = 0; l < k; l++ do
16:                        if CA_{j,l} = v then
17:                            C'_{j,l} ← CA_{i,l}
18:                        end if
19:                    end for
20:                end if
21:            end for
22:        end for
23:        if C' es un CA then
24:            B ← C'
25:        end if
26:    until C' es un CA
27:    return B
28: end procedure
```

- $\mathcal{O}$ de tamaño $\phi \times t$.

y fija las tuplas de acuerdo a la bandera *constantes* y $\mathcal{F}$ con el Algoritmo FIXTUPLAS. Se inicializa la estructura $\mathcal{O}$ con los ϕ vectores de distancias de las órbitas equivalentes. A μ se le asigna el valor 1 y si *constantes* está activa, se fijan las t-tuplas constantes en $\mathcal{P}$. En el segundo paso se fijan los μ ceros seguido del 1 al primer vector en SV y se generan los VCs para los z vectores utilizando el Algoritmo NEXTSV, por cada z VCs se hace un RL utilizando el Algoritmo ISSV para ajustar las órbitas que cubren esos VCs, is ya no hay más VCs con μ ceros, se incrementa el valor

Algoritmo 8 Algoritmo de búsqueda exacto para la construcción de VIs.

```
1: procedure EXACTO(k, t, v, z, δ, σ, constantes, F)
2:     se calcula φ                                          ▷ Paso 1
3:     SV es un arreglo de tamaño z × k
4:     PSV es un arreglo de tamaño z × k
5:     P es un arreglo de tamaño v^t × φ × k
6:     O es un arreglo de tamaño φ × t
7:     μ ← 1
8:     Inicializar O con los vectores de distancias de las órbitas
9:     FIXTUPLAS (P, φ, constantes, F)
10:    repeat                                                ▷ Paso 2
11:        for i = 0; i < μ; i++ do
12:            SV_{0,i} ← 0
13:        end for
14:        SV_{0,μ} = 1
15:        while NEXTSV (SV, μ, k, v, z) do
16:            if ISSV (SV, PSV, P, O, z, t, δ, σ) then
17:                TRANSFORMACIÓN(δ, σ, PSV)                 ▷ Paso 3
18:            end if
19:        end while
20:        μ ← μ + 1
21:    until μ < (k − 1)
22: end procedure
```

de μ en 1 y se repite el proceso mientras $\mu < (k-1)$. En el tercer paso se transforman los z VCs de PSV en un CA y se guarda en un archivo de salida.

Para ejemplificar el algoritmo exacto, suponemos que tenemos los valores $k = 5$, $v = 2$ y $t = 2$ y *constantes* está desactivado. Primero se asigna $\phi = 2$, $\mathcal{O}$ queda de la siguiente forma:

$$\mathcal{O}_0 = \{1, 4\}$$
$$\mathcal{O}_0 = \{2, 3\}$$

y $\mathcal{P}$ de la siguiente:

$$\mathcal{P}_0=\{\{5,5,5,5,5\},\{5,5,5,5,5\}\}$$
$$\mathcal{P}_1=\{\{0,0,0,0,0\},\{0,0,0,0,0\}\}$$
$$\mathcal{P}_2=\{\{0,0,0,0,0\},\{0,0,0,0,0\}\}$$
$$\mathcal{P}_3=\{\{5,5,5,5,5\},\{5,5,5,5,5\}\}$$

Como la opción *constantes* no está activa y $\mathcal{F}=0$, el arreglo $\mathcal{P}$ se queda igual. A continuación se genera el siguiente VC a través del Algoritmo 2 y se obtiene $VI=\{0,0,0,0,1\}$, inmediatamente después se verifica si el vector VI contiene en todas sus órbitas todas las t-tuplas a través del Algoritmo 3. Una vez que se valida SV el arreglo $\mathcal{P}$ queda de la siguiente manera:

$$\mathcal{P}_0=\{\{3,3,3,3,3\},\{3,3,3,3,3\}\}$$
$$\mathcal{P}_1=\{\{2,2,2,2,2\},\{2,2,2,2,2\}\}$$
$$\mathcal{P}_2=\{\{2,2,2,2,2\},\{2,2,2,2,2\}\}$$
$$\mathcal{P}_3=\{\{3,3,3,3,3\},\{3,3,3,3,3\}\}$$

Por lo tanto, $SV=\{0,0,0,0,1\}$ sí cubre en todas sus órbitas todas las t-tuplas y se construye el CA, el cual queda de la siguiente manera:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

3.4 Resumen

En este capítulo se mostró la metodología que se siguió para resolver el problema de la construcción de VIs, dividiendo el capítulo en tres secciones principales: la sección de definiciones,

la sección de la metodología y la sección del algoritmo de búsqueda exacto. Dentro de la sección de definiciones se encuentran las operaciones de *Rotación* y *Traslación*, el uso de más de un VI para construir el CA, la definición del concepto de VC y dentro de la definición de órbitas, se muestra una representación de las órbitas a través de las distancias entre los pares de elementos de la órbita, las fórmulas para contar el número de vectores de distancias de las órbitas.

La metodología se dividió en dos etapas, la etapa de búsqueda y construcción y la etapa de manipulación. La etapa de búsqueda y construcción se centró en la búsqueda de todos aquellos VIs que construyan un CA y construir los CAs. Ésta etapa la componen los módulos fijar tuplas, el módulo de generación de vectores, el módulo de validación de vectores y el módulo de construcción de CAs. La etapa de manipulación se enfocó en detectar los elementos comodines y a través de ellos, tratar de reducir los renglones de los CAs construidos en la etapa anterior. Los módulos que componen la etapa de manipulación son los módulos de detección de elementos comodines y la reducción de renglones.

En la última sección se mostró el algoritmo exacto que se desarrolló para la etapa de búsqueda y construcción. Para saber si el algoritmo exacto puede construir CAs generaliza los métodos previamente reportados, se requirió de hacer una experimentación computacional, dicha experimentación computacional se mostrará en el siguiente capítulo, mostrando las las cotas que se obtienen a través de este método generalizado y comparándolas con las cotas obtenidas en los trabajos previos.

4

Resultados

En el capítulo anterior se presentó la generalización para construir un CA utilizando VIs, dividiendo la metodología en dos etapas, la etapa de búsqueda y construcción; y la etapa de manipulación. Dentro de la etapa de búsqueda y construcción se mostraron los módulos: fijar tuplas, generación de vectores, validación de vectores y construcción del CA; la etapa de manipulación contiene los módulos: detección de elementos comodines y reducción de renglones. De cada uno de estos módulos se mostró un algoritmo y al final del capítulo se mostró un algoritmo exacto para buscar los VIs que integra los módulos de fijar tuplas, generación de vectores, validación de vectores y construcción del CA. En este capítulo se muestran los resultados obtenidos a través del algoritmo exacto (etapa de búsqueda y construcción) y los algoritmos de detección de elementos comodines y reducción de renglones (etapa de manipulación). El capítulo se divide en tres secciones, en la primer sección se muestra los casos seleccionados para verificar la generalidad de la metodología, así como los detalles de la experimentación. En la segunda sección se mostrarán los resultados obtenidos siguiendo la metodología y se compararán con los reportados en los trabajos relacionados

Tabla 4.1: Casos seleccionados para verificar si la metodología propuesta es general a los metodos que utilizan un VI previamente reportados.

k	t	v	$\mathcal{F}$	*Traslaciones*
5	2	3	1	
6	2	4	1	-
7	2	4	1	
9	2	4	2	-
7	2	5	1	
8	2	5	1	-
10	2	5	2	
20	3	3	0	0,1,2
22	3	3	0	0,1,2
23	3	3	0	0,1,2
21	4	2	0	0,1
22	4	2	0	0,1
25	4	2	0	0,1

y los mejores reportados. Al final se muestra un resumen del capítulo.

4.1 Experimentación Computacional

El algoritmo exacto de la etapa de búsqueda y construcción (Algoritmo 8) y los algoritmos de la etapa de manipulación (el Algoritmo 6 del módulo de detección de elementos comodines y el Algoritmo 7 del módulo de reducción de renglones) fueron implementados en el lenguaje de programación C.

Para verificar si la metodología es una generalización de los otros métodos previamente reportados (Lobb *et al.* [37] y Colbourn [35]) se seleccionaron diferentes casos con diferente número de símbolos fijos $\mathcal{F}$ (para los reportados por Lobb *et al.*) y algunos casos con pocas columnas (para los reportados por Colbourn). Los casos seleccionados se muestran en la Tabla 4.1.

Para comparar el método general con las mejores cotas reportadas en el estado del arte se definió una experimentación computacional seleccionando un rango de valores para t, k y v. El rango de valores seleccionados para la experimentación computacional se muestran en la Tabla 4.2.

Tabla 4.2: Rango de valores de t, v y k utilizados para la experimentación computacional de la etapa de búsqueda y construcción.

Rango de valores		
t	v	k
3	2	$3,\ldots,6$
3	3	$9,\ldots,26$
3	4	$9,\ldots,26$
3	5	$3,\ldots,26$
4	2	$11,\ldots,26$
4	3	$6,\ldots,26$
4	4	$6,\ldots,26$
5	2	$7,\ldots,26$
5	3	$11,\ldots,26$
6	2	$9,\ldots,26$

Para la etapa de búsqueda y construcción se utilizó el clúster híbrido de supercómputo Xiuhcoatl del CINVESTAV. El clúster Xiuhcoatl cuenta con 1056 procesadores INTEL distribuidos entre 88 servidores y cuenta con 2304 procesadores AMD distribuidos entre 72 servidores. Cada servidor INTEL cuentan con 2112 GB en memoria RAM y cada servidor AMD cuentan con 4600 GB en memoria RAM. El clúster cuenta con 67.16 TB de almacenamiento usables. A diferencia de la etapa de búsqueda y construcción, la etapa de manipulación se realizo en una computadora iMAC. La computadora iMAC cuenta con un procesador intel i5 de segunda generación, 4 GB en memoria RAM y 500 GB en disco duro. Los resultados obtenidos de cada etapa se muestran en la siguiente sección.

4.2 Resultados

Para verificar que la metodología es una generalización de los métodos que construyen CA utilizando VIs que han sido reportados se ejecutó el Algoritmo 8 con los valores que muestra la Tabla 4.1 y comparándolos con los que reportados en [35, 37]. Los resultados de esta comparación se muestran en la Tabla 4.3.

Tabla 4.3: Tabla comparativa de los resultados obtenidos y los reportados por Lobb.

k	t v $\mathcal{F}$ Ingrediente	N	N
	VIs	reportada	obtenida
5	2 3 1 $CAN(2,5,1) = 1$	11	11
	$\{0,1,1,2,1\}$ $\{0,2,2,1,2\}$		
6	2 4 1 $CAN(2,6,1) = 1$	19	19
	$\{0,1,1,3,3,1\}$ $\{0,2,2,1,1,2\}$ $\{0,3,3,2,2,3\}$		
7	2 4 1 $CAN(2,7,1) = 1$	22	22
	$\{0,1,0,1,3,3,1\}$ $\{0,2,0,2,1,1,2\}$ $\{0,3,0,3,2,2,3\}$		
9	2 4 2 $CAN(2,9,2) = 6$	24	24
	$\{0,1,2,2,3,2,1,0,3\}$ $\{0,1,3,3,2,3,1,0,2\}$		
7	2 5 1 $CAN(2,7,1) = 1$	29	29
	$\{0,1,1,3,2,4,1\}$ $\{0,2,2,4,3,1,2\}$		
	$\{0,3,3,1,4,2,3\}$ $\{0,4,4,2,1,3,4\}$		
8	2 5 2 $CAN(2,8,1) = 1$	33	33
	$\{0,1,1,4,4,2,1,2\}$ $\{0,2,2,1,1,3,2,3\}$		
	$\{0,3,3,2,2,4,3,4\}$ $\{0,4,4,3,3,1,4,1\}$		

Continúa en la siguiente página

k	t VIs	v	$\mathcal{F}$	Ingrediente	N reportada	N obtenida
10	2	5	2	$CAN(2,8,1) = 1$	33	33
	$\{0,1,2,0,3,1,3,4,3,3\}$ $\{0,1,3,0,4,1,4,2,4,4\}$					
	$\{0,1,4,0,2,1,2,3,2,2\}$					

Para comprobar que los resultados obtenidos utilizando los valores de la Tabla 4.2 son de calidad competitiva se accedió a la página de Colbourn [81] el día 21 de Noviembre del 2013. La página contiene las tablas de *Covering Arrays* con los mejores resultados reportados en el estado del arte.

Al ejecutar el Algoritmo 8 en el cúster Xiuhcoatl, se lograron mejorar dos resultados sin utilizar la etapa de manipulación. Del resto de las instancias, en diecisiete casos se igualó el mejor reportado y en nueve casos se quedó a pocos renglones para igualar al mejor reportado. En la Tabla 4.4 se muestran los VIs obtenidos en esta etapa.

Tabla 4.4: Tabla comparativa de los resultados obtenidos y los mejores reportados.

k	t VIs	v	Rotaciones	Traslaciones	Renglones constantes	Resultado	Mejor reportado	Δ
5	2	2	0,1 1,2 0 0	0 1 1 0	No	6	6	0
	$\{0,1,0,1,1\}$		$\{0,0,1,1,1\}$	$\{0,0,1,0,1\}$	$\{0,0,0,0,1\}$			
4	2	5	0,1,2,3	0,1	Si	25	25	0
	$\{0,1,3,2\}$							

Continúa en la siguiente página

k	t	v	Rotaciones	Traslaciones	Renglones constantes	Resultado	Mejor reportado	Δ
	VIs							
5	2	5	0,1,2,3,4	0,1	No	25	25	0
	$\{0,0,1,3,1\}$							
3	3	2	0,1	0,1,2	Si	8	8	0
	$\{0,0,1\}$							
4	3	2	0,1	0,1,2,3	No	8	8	0
	$\{0,0,0,1\}$							
5	3	2	0,1	0,1,2,3,4	No	10	10	0
	$\{0,0,0,0,1\}$							
6	3	2	0,1	0 ... 5	No	12	12	0
	$\{0,0,0,0,0,1\}$							
11	3	2	0	0 ... 10	Si	13	12	1
	$\{0,0,0,1,0,0,1,0,1,1,1\}$							
16	3	2	0	0 ... 15	Si	18	17	1
	$\{0,0,0,0,0,1,0,1,0,0,1,1,1,0,1,1\}$							
17	3	2	0	0 ... 16	Si	19	18	1
	$\{0,0,0,0,0,0,1,1,1,0,1,0,1,1,0,1,1\}$							
4	3	3	0,1,2	0,1,2,3 0,1,2,3	Si	27	27	0
	$\{0,1,0,2\}$ $\{0,0,1,1\}$							

Continúa en la siguiente página

k	t	v	Rotaciones	Traslaciones	Renglones constantes	Resultado	Mejor reportado	Δ
	VIs							
5	3	3	0,1,2	$0,\ldots,4$	Si	33	33	0
				$0,\ldots,4$				
	$\{0,1,0,2,2\}$			$\{0,0,2,1,2\}$				
14	3	3	0,1,2	0,...,13	Si	**45**	50	-5
	{0, 0, 1, 2, 1, 2, 2, 1, 1, 0, 2, 0, 2, 2}							
16	3	3	0,1,2	0,...,15	Si	**51**	59	-8
	$\{0,0,0,1,2,1,1,0,1,1,1,0,2,0,0,1\}$							
20	3	3	0,1,2	0,...,19	No	60	59	1
	{0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 0, 2, 1, 0, 1, 2, 0, 2, 1, 1}							
22	3	3	0,1,2	0,...,21	No	66	66	0
	$\{0,0,0,0,0,0,1,0,1,2,2,1,0,1,1,2,0,1,0,2,1,1\}$							
23	**3**	**3**	**0,1,2**	**0,...,22**	**No**	**69**	**69**	**0**
	{0, 0, 0, 0, 0, 0, 1, 2, 2, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 2, 2, 1}							
24	**3**	**3**	**0,1,2**	**0,...,23**	**No**	**72**	**71**	**1**
	{0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 2, 2, 1, 0, 2, 1, 2, 0, 0, 0, 2, 0, 1, 1}							
11	**4**	**2**	**0,1**	**0,...,10**	**Si**	**24**	**24**	**0**
	{0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1}							
17	4	2	0 0	0,...,16	Si	36	35	1
				0,...,16				
	$\{0,0,1,0,0,1,0,1,1,1,0,0,1,1,1,0,1\}$			$\{0,0,0,1,0,1,1,1,1,1,0,1,0,0,0,1,1\}$				

Continúa en la siguiente página

k	t	v	Rotaciones	Traslaciones	Renglones constantes	Resultado	Mejor reportado	Δ
	VIs							
19	4	2	0,1	0,...,18	Si	40	39	1
	$\{0,0,0,0,1,0,1,0,1,1,1,1,0,0,1,0,0,1,1\}$							
21	4	2	0,1	0,...,20	No	42	42	0
	$\{0,0,0,1,0,1,1,1,0,1,1,0,1,1,1,1,0,1,1,1,1\}$							
21	4	2	0,1	0,...,20	Si	44	42	2
	$\{0,0,0,0,0,0,1,1,0,1,0,0,1,0,1,1,1,0,1,1,1\}$							
23	4	2	0,1	0,...,22	No	46	46	0
	$\{0,0,1,0,0,1,1,1,1,1,0,0,1,1,1,0,1,1,1,0,1,0,1\}$							
24	4	2	0,1	0,...,23	Si	50	50	0
	$\{0,0,0,1,0,0,1,1,0,1,0,1,0,0,1,1,1,1,1,0,1,1,1,1\}$							
25	4	2	0,1	0,...,24	No	50	50	0
	$\{0,0,1,0,0,1,1,0,1,0,1,1,1,0,0,1,1,1,1,1,1,0,1,0,1\}$							
26	4	2	0,1	0,...,25	No	52	51	1
	$\{0,0,0,0,0,1,0,1,0,0,1,0,0,0,0,0,1,1,0,1,1,1,0,1,1,1\}$							
7	5	2	0,1	0,1	0,1	0,...,6	No	42	42	0
						0,...,6				
						0,...,6				
	$\{0,0,0,1,0,0,1\}$					$\{0,0,0,0,1,0,1\}$	$\{0,0,0,0,0,1,1\}$			

Para la etapa de manipulación se tomaron las instancias en las que no se logró igualar el mejor resultado reportado en el estado del arte. El total de instancias para esta etapa fue nueve. Durante esta etapa se lograron igualar siete instancias con los mejores reportados en el estado del arte de las nueve instancias. En la Tabla 4.5 se muestran los resultados obtenidos en la etapa de manipulación.

Dentro de los casos que se muestran en la Tabla 4.5, los que se muestran con un $*$ se obtuvieron utilizando el algoritmo ZEROSUM, este es otro procedimiento diferente a la etapa de manipulación. El algoritmo ZEROSUM consiste en agregar una nueva columna, el valor que se le asigna a esa nueva columna del renglón j del CA se realiza siguiendo la operación: $(v-1) - \sum_{i=0}^{k-1} CA_{j,i} \bmod v$, a los vectores rotados se les aplica el algoritmo ZEROSUM y a los renglones constantes se les agregó el complemento del ZEROSUM, este método es similar al reportado por Colbourn [35].

Tabla 4.5: Tabla comparativa de los resultados obtenidos de la etapa de manipulación y los mejores reportados.

	k	t	v	N etapa anterior	N obtenida	Mejor reportado	Δ
	11	3	2	13	12	12	0
	16	3	2	18	17	17	0
	17	3	2	19	18	18	0
	20	3	3	60	59	59	0
	17	4	2	36	35	35	0
	19	4	2	40	39	39	0
$*$	22	4	2		44	44	0

4.3 Resumen

En este capítulo se mostraron los resultados obtenidos con los algoritmos presentados en el capítulo de la metodología. El capítulo se dividió en dos principales secciones, la sección de la experimentación computacional y la sección de resultados. Dentro de la sección de experimentación computacional se comentaron las características de la implementación. También se definió el rango de la experimentación computacional para el algoritmo de la etapa de búsqueda y construcción; y los casos que se seleccionaron para comprobar si la metodología es general a los métodos previamente reportados. En la sección de resultados se dio evidencia de la efectividad de la metodología propuesta.

Los resultados fueron comparados con los mejores reportados en el estado del arte. Dentro de la etapa de búsqueda y construcción se mejoraron dos instancias, 24 instancias se igualaron, ocho instancias se quedaron a un renglón de igualar al mejor reportado y solo un caso se quedó a dos renglones de igualar al mejor reportado. Durante la etapa de manipulación se utilizaron las instancias que no pudieron igualar al mejor reportado y en siete de esos casos se logró igualar al mejor reportado. En el siguiente capítulo se hablará sobre las conclusiones que se obtuvieron al realizar este trabajo de tesis.

5

Conclusiones y Trabajo Futuro

En el capítulo anterior se mostraron los resultados obtenidos a través de la metodología propuesta para construir CAs utilizando VIs. El capítulo anterior se dividió en dos secciones, la sección de experimentación computacional y la sección de resultados. En la sección de experimentación computacional se mostraron los casos que se seleccionaron para verificar la generalización de la metodología y los casos que se seleccionaron para comparar la metodología con los mejores reportados en el estado del arte. En la sección de resultados se compararon con los mejores reportados en el estado del arte. En este capítulo se mostrarán las conclusiones finales de este trabajo de tesis. El capítulo se divide en dos secciones, la sección de contribuciones y la sección de trabajo futuro. En la sección de contribuciones se resumirán las principales contribuciones que se derivaron en el desarrollo de este trabajo de tesis. La sección de trabajo futuro se mostrarán algunas maneras para complementar o mejorar este trabajo de tesis.

5.1 Contribuciones

En este trabajo de tesis se mostró una generalización de los trabajos de construcción por grupos de Lobb *et al.* [37] y ciclotomía de Colbourn [35]. La generalización se centró principalmente en:

a) Abordar los valores de t que no eran explorados por Lobb *et al.* y los valores de k que no eran explorados por Colbourn.

b) La posibilidad de complementar los VIs con renglones constantes u otro CA.

c) El uso de más de un VI para construir CAs.

Las principales contribuciones que se derivan de este trabajo de tesis son:

- La definición formal de los VCs.
- Una generalización para construir CAs de $t = 2$ hasta $t = 6$.
- La metodología para construir CAs utilizando VIs.
- La definición de dos nuevas cotas de CAs. En la Tabla 5.1 se muestran esas instancias.

Tabla 5.1: Instancias mejoradas durante este trabajo de tesis.

k	t	v	N anterior	N nueva
14	3	3	50	45
16	3	3	59	51

En la Tabla 5.2 se muestran las cotas que fueron igualadas.

- Las ecuaciones matemáticas para contar el número total de órbitas para $t = 2$ hasta $t = 6$.

Tabla 5.2: Instancias igualadas durante este trabajo de tesis.

t	k	v	N	t	k	v	N	t	k	v	N
2	5	2	6	3	4	2	8	3	23	3	69
2	5	2	11	3	5	2	10	4	11	2	24
2	6	4	19	3	6	2	12	4	17	2	35
2	7	4	22	3	11	2	12	4	19	2	39
2	9	4	24	3	16	2	17	4	21	2	42
2	7	5	29	3	17	2	18	4	22	2	44
2	10	5	36	3	4	3	27	4	23	2	46
2	4	5	25	3	5	3	33	4	24	2	50
2	5	5	25	3	20	3	59	4	25	2	50
3	3	2	8	3	22	3	66	5	7	2	42
2	8	5	33								

5.2 Trabajo Futuro

A continuación se menciona como extender la metodología en este trabajo de tesis.

- Extender el cálculo del número de órbitas a fuerzas mayores a 6.
- Paralelización del algoritmo exacto. Al paralelizar el algoritmo se puede tener las siguientes ventajas:
  - **a)** Reducir el tiempo de computo requerido.
  - **b)** Construir VIs para valores k, t y v mayores a los usados en esta tesis.
- Utilizar otros operadores sobre los VIs. En esta tesis se usaron las operaciones de *Rotación* y *Traslación* pero usando otros operadores (uso de grupos abelianos) se podrán obtener nuevos resultados.

Apéndices

A

Entradas y Salidas de los Algoritmos

Los CAs construidos con los algoritmos de la metodología son almacenados en archivos. Para el contenido de los archivos se utilizó la siguiente estructura utilizando la notación de Backus-Naur (BNF por sus siglas en inglés):

< contenido >::=< encabezado > < cuerpo >

< encabezado >::=< comentarios > < parámetros >

< comentarios >::=< comentario > [< comentario >]

< comentario >::=['C' < texto > | 'c' < texto >]

< texto >::='0' | '1' | ... | '9' | 'a' | 'b' | ... | 'z' | 'A' | 'B' | ... | 'Z'

< parámetros >::=< renglones > < columnas > < alfabeto > < fuerza >

< renglones >::=< números >

< columnas >::=< números >

< alfabeto >::=< números > '^' < números >

< fuerza >::=< números >

< números >::=< número > [< números >]

< número >::='0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

< cuerpo >::=< renglones >

< renglones >::=< renglón > [< renglones >]

< renglón >::=< números > [< renglón >]

Un ejemplo del contenido de un archivo sería el siguiente:

EJEMPLO 16 (Ejemplo de un archivo de CA).

Para los valores $N = 6$, $k = 10$, $v = 2$ y $t = 2$, un archivo de un CA para esos valores sería el siguiente:

```
C Convering Array con N=6 k=10 v=2^10 t=2
c
6 10 2^10 2
0 0 0 0 0 0 0 0 0 0
1 1 1 0 1 1 0 1 0 0
1 1 0 1 1 0 1 0 1 0
1 0 1 1 0 1 1 0 0 1
0 1 1 1 0 0 0 1 1 1
0 0 0 0 1 1 1 1 1 1
```

Cada algoritmo recibe diferentes parámetros de entrada, a continuación se mostrarán los parámetros de entrada de cada algoritmo utilizando la notación BNF. El algoritmo de la etapa de búsqueda y construcción maneja una variedad de valores de entrada, dichos parámetros se muestran a continuación:

<Algoritmo exacto>::=< valores >

< valores >::=< requeridos > < opcionales >

< requeridos >::=< columnas > < fuerza > < alfabeto > < vectores > < δ > < σ >

< opcionales >::=< tuplas fijas > < complemento >

< tuplas fijas >::='-CONSTANTES' | '-NSYMBOLS' < números >

< complemento>::='-FIX' | '-WC'

< columnas >::='-K' < números >

< fuerza >::='-T' < números >

< alfabeto >::='-V' < números >

< vectores >::='-SVN' < números >

< δ >::='-TR' < vector >

< σ >::='-ROT' < vector >

< vector >::=< números > ',' < vector > | < números > ' ' < vector > | < números >

< números >::=[< números >] < número >

< número >::='0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

Un ejemplo de cómo sería la ejecución del algoritmo exacto se muestra a continuación:

Ejemplo 17 (Ejemplo de ejecución del algoritmo exacto).

Suponemos que el algoritmo exacto se llama exSV, y queremos buscar un VI para construir un CA con los valores de $k = 5$, $v = 3$, $t = 2$, $\delta = 0, 1, 2$, $\sigma = 0, 1, 2, 3, 4$ y queremos que ese CA tenga renglones constantes. La ejecución sería de la siguiente manera: exSV -K 5 -V 3 -T 2 -SVN 1 -TR 0,1,2 -ROT 0,1,2,3,4 -CONSTANTES

Al terminar el algoritmo exacto, genera archivos de salida, cada archivo de salida será un CA diferente, construido con VIs diferentes para los valores de entrada. El nombre de los archivos de salida tienen la siguiente estructura:

< nombre CA >::=< renglones > < columnas > < alfabeto > < fuerza > '.ca'

< renglones >::='N' < números >

< columnas >::='k' < números >

< alfabeto >::='v' < números > '^' < números >

< fuerza >::='t' < números >

< números >::=[< números >] < número >

< número >::='0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

En el nombre se mostrarán los datos principales del CA, estos datos son N, k, v y t, un ejemplo del nombre que generaría el algoritmo exacto es el siguiente:

EJEMPLO 18 (Ejemplo del nombre de un archivo de salida).

Suponemos que se ejecutó el algoritmo exacto con los valores de $k = 5$, $v = 3$, $t = 2$ $\delta = 0, 1, 2$ y $\sigma = 0, 1, 2, 3, 4$. El algoritmo exacto generaría archivos con el siguiente nombre N15k5v3^2t2.ca

Estos archivos se utilizan como entrada al algoritmo de detección de elementos comodines. Las entradas del algoritmo de detección de elementos comodines se muestra a continuación:

< Algoritmo detector de elementos comodines >::=< entrada >

< entrada >::='-FILE' < nombre CA >

A continuación se muestra un ejemplo de como sería la ejecución del algoritmo para detectar elementos comodines:

EJEMPLO 19 (Ejemplo de ejecución del detector de elementos comodines).

Suponiendo que el algoritmo de detección de elementos comodines se llama wc y queremos saber cuantos elementos comodines tiene el archivo llamado N15k5v3^2t2.ca. La ejecución del algoritmo sería de la siguiente manera:

wc -FILE N15k5v3^2t2.ca

Al igual que el algoritmo exacto, el algoritmo de detección de elementos comodines genera como salida archivos en los cuales estará el CA con los elementos comodines. Estos archivos tienen un nombre muy parecido a los que genera el algoritmo exacto, con la diferencia de que el '.ca' cambia a '.wc' y le sigue el número de elementos comodines que tiene el archivo. La notación del nombre se muestra a continuación.

< salida CA >::=< renglones > < columnas > < alfabeto > < fuerza > '.wc-' < números >
< renglones >::='N' < números >
< columnas >::='k' < números >
< alfabeto >::='v' < números > '^' < números >
< fuerza >::='t' < números >
< números >::=[< números >] < número >
< número >::='0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

Un ejemplo del nombre que genera el algoritmo de detección de elementos comodines es el siguiente:

Ejemplo 20 (Ejemplo del nombre de un archivo con elementos comodines).

Suponemos que el algoritmo exacto generó un archivo con el nombre N15k5v3^2t2.ca y dicho CA tiene un símbolo comodín en cada uno de sus renglones. El algoritmo de detección de elementos comodines generaría un archivo con el siguiente nombre:

N15k5v3^2t2.wc-15

Estos archivos son utilizados como entrada para el último algoritmo, el algoritmo de reducción de renglones. El parámetro de entrada del algoritmo se muestra a continuación:

< Algoritmo reducción de renglones >::=< entrada >
< entrada >::='-FILE' < CA comodines >

Para ejemplificar la manera de ejecutar el algoritmo de reducción de renglones, se muestra el siguiente ejemplo:

EJEMPLO 21 (Ejemplo de los parámetros del algoritmo de reducción de renglones).

Suponemos que el algoritmo se llama reduce y queremos reducir los renglones del archivo N15k5v3^2t2.wc-15. La ejecución del algoritmo sería de la siguiente manera:

reduce -FILE N15k5v3^2t2.wc-15

La salida del algoritmo de reducción de renglones son archivos con el CA con renglones menos (si es que es posible) al CA de entrada. Este algoritmo puede que no reduzca ningún renglón del CA. El nombre de estos archivos es igual a los que genera el algoritmo exacto.

Bibliografía

[1] F.-Y. Wang, K. M. Carley, D. Zeng, and W. Mao, "Social computing: From social informatics to social intelligence," *IEEE Intelligent Systems*, vol. 22, pp. 79–83, March 2007.

[2] D. Graham, E. V. Veenendaal, I. Evans, and R. Black, *Fundation of Software Testing*. Cengage Learning Business Press, 2006.

[3] M. Dowson, "The ariane 5 software failure," *ACM SIGSOFT Software Engineering Notes*, vol. 22, p. 84, March 1997.

[4] N. Leveson and C. Turner, "An investigation of the therac-25 accidents," *Computer*, vol. 26, pp. 18–41, July 1993.

[5] R. S. Pressman, *Software Engineering: a practitioner's approach*. Thomas Casson, 2001.

[6] B. Beizer, *Black-Box Testing: Techniques for Functional Testing of Software and Systems*. Katerine Schowalter, 1 ed., 1995.

[7] F. Lammermann, "Benefits of software measures for evolutionary white-box testing," in *Proceedings of Genetic and Evolutionary Computation Conference 2005 (Washington DC, 2005), ACM, ACM*, pp. 1083–1084, Press, 2005.

[8] K. A. Bush, "Orthogonal arrays of index unity," *Annals of Mathematics Statistics*, vol. 23, no. 3, pp. 426–434, 1952.

[9] K. Burr and W. Young, "Combinatorial test techniques: Table-based automation, test generation and code coverage," in *International Conference on Software Testing Analysis and Review*, pp. 503–513, West, 1998.

[10] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton, "The combinatorial design approach to automatic test generation," *IEEE Software*, vol. 13, pp. 83–88, September 1996.

[11] M. Grindal, J. Offutt, M. S. F. Andler Grindal, J. Offut, and S. F. Andler, "Combination testing strategies: A survey," *Software Testing, Verification, and Reliability*, vol. 15, pp. 167–199, 2005.

[12] A. Hartman, "Software and hardware testing using combinatorial covering suites," in *Graph Theory, Combinatorics and Algorithms* (M. Golumbic and I. A. Hartman, eds.), vol. 34 of *Operations Research/Computer Science Interfaces Series*, pp. 237–266, Springer US, 2005.

[13] R. Kuhn, R. Kacker, Y. Lei, and J. Hunter, "Combinatorial software testing," *Computer*, vol. 42, pp. 94 –96, August 2009.

[14] D. R. Kuhn and V. Okum, "Pseudo-exhaustive testing for software," in *Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop*, SEW '06, (Washington, DC, USA), pp. 153–158, IEEE Computer Society, 2006.

[15] K. Z. Zamli, M. F. J. Klaib, M. I. Younis, N. A. M. Isaa, and R. Abdullahb, "Design and implementation of a t-way test data generation strategy with automated execution tool support," *Information Sciences*, vol. 181, pp. 1741–1758, May 2011.

[16] G. Seroussi and N. Bshouty, "Vector sets for exhaustive testing of logic circuits," *IEEE Transactions on Information Theory*, vol. 34, pp. 513–522, May 1988.

[17] A. S. Hedayat, N. J. A. Sloane, and J. Stufken, *Orthogonal arrays: theory and applications*. Springer, 1999.

[18] D. R. Kuhn, D. R. Wallace, A. M. K. D. R. Gallo, Jr., D. R. Wallace, and A. M. Gallo Jr., "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, vol. 30, pp. 418–421, June 2004.

[19] C. J. Colbourn, "Combinatorial aspects of covering arrays," *Le Matematiche*, vol. 58, pp. 121–167, 2004.

[20] I. Izquierdo Marquez, "Construction of towers of covering array," Master's thesis, CINVESTAV-Laboratorio de Tecnologías de Información., 2013.

[21] A. L. González-Hernández, *Un algoritmo de optimización combinatoria para la construcción de covering arrays mixtos de fuerza variable*. PhD thesis, CINVESTAV-Laboratorio de Tecnologías de Información, 2013.

[22] J. Lawrence, R. N. Kacker, Y. Lei, D. R. Kuhn, and M. Forbes, "A survey of binary covering arrays," *The Electronic Journal of Combinatorics*, vol. 18, no. 1, p. P84, 2011.

[23] C. J. Colbourn and M. B. Cohen, "A deterministic density algorithm for pairwise interaction coverage," in *proceeding of: IASTED International Conference on Software Engineering*, pp. 245–252, Proceedings of the International Conference on Software Engineering (SE 2004), February 2004.

[24] H. Avila-George, "Verificación de covering arrays utilizando supercomputación y la computación grid," Master's thesis, Universidad Politécnica de Valencia - Departamento de Sistemas Informáticos y computación, 2010.

[25] O. A. Carrizales-Turrubiates, "Reducción óptima de covering arrays," Master's thesis, CINVESTAV-Laboratorio de Tecnologías de Información, 2011.

[26] J. A. Martínez-Peña, "Construction of covering arrays of ternary alphabet and variable strength," Master's thesis, CINVESTAV-Laboratorio de Tecnologías de Información, 2011.

[27] A. Rodríguez-Cristerna, "Construcción de covering arrays usando una generalización del operador de fusión," Master's thesis, CINVESTAV-Laboratorio de Tecnologías de Información., 2012.

[28] G. O. H. Katona, "Two applications (for search theory and truth functions) of sperner type theorems," *Periodica Mathematica Hungarica*, vol. 3, no. 1-2, pp. 19–26, 1973.

[29] D. J. Kleitman, D. J. J. Spencer Kleitman, and J. Spencer, "Families of k-independent sets," *Discrete Mathematics*, vol. 6, no. 3, pp. 255–262, 1973.

[30] N. J. A. Sloane, "Covering arrays and intersecting codes," *Journal of Combinatorial Designs*, vol. 1, no. 1, pp. 51–63, 1993.

[31] M. Chateauneuf, C. Colbourn, and D. Kreher, "Covering arrays of strength three," *Designs, Codes and Cryptography*, vol. 16, pp. 235–242, 1999.

[32] M. Chateauneuf and D. L. Kreher, "On the state of strength-three covering arrays," *Journal of Combinatorial Designs*, vol. 10, pp. 217–238, May 2002.

[33] K. Meagher and B. Stevens, "Group construction of covering arrays," *Journal of Combinatorial Designs*, vol. 13, pp. 70–77, January 2005.

[34] C. J. Colbourn, "Strength two covering arrays: Existence tables and projection," *Discrete Mathematics*, vol. 308, no. 5–6, pp. 772 – 786, 2008. Selected Papers from 20th British Combinatorial Conference.

[35] C. J. Colbourn, "Covering arrays from cyclotomy," *Designs, Codes and Cryptography*, vol. 55, pp. 201–219, May 2010.

[36] J. R. Lobb, *Hybrid Strength Two Covering Array Constructions: Using Cover Starters to Create Covering Arrays*. PhD thesis, School of Mathematics and Statistics Ottawa-Carleton Institute for Mathematics and Statistics Carleton University, 2010.

[37] J. R. Lobb, C. J. Colbourn, P. Danziger, B. Stevens, and J. Torres-Jimenez, "Cover starters for covering arrays of strength two," *Discrete Mathematics*, vol. 312, pp. 943–956, March 2012.

[38] H. Barker, "Sum and product tables for galois fields," *International Journal in Mathematical Education Science*, vol. 17, pp. 473–485, 1986.

[39] J. Torres-Jimenez and E. Rodriguez-Tello, "Simulated annealing for constructing binary covering arrays of variable strength," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pp. 1–8, july 2010.

[40] A. Rényi, *Foundations of probability*. Holden-Day, 1970.

[41] D. T. Tang and L. S. Woo, "Exhaustive test pattern generation with constant weight vectors," *IEEE Transactions on Computers*, vol. 32, pp. 1145–1150, December 1983.

[42] J. Martinez-Peña and J. Torres-Jimenez, "A branch and bound algorithm for ternary covering arrays construction using trinomial coefficients," *Research in Computing Science*, vol. 49, pp. 61–71, 2010.

[43] J. Martinez-Peña, J. Torres-Jimenez, N. Rangel-Valdez, and H. Avila-George, "A heuristic approach for constructing ternary covering arrays using trinomial coefficients," in *Proceedings of the 12th Ibero-American conference on Advances in artificial intelligence* (A. Kuri-Morales and G. Simari, eds.), IBERAMIA'10, (Berlin, Heidelberg), pp. 572–581, Springer-Verlag, 2010.

[44] C. J. Colbourn, S. S. Martirosyan, G. L. Mullen, D. Shasha, G. B. Sherwood, and J. L. Yucas, "Products of mixed covering arrays of strength two," *Journal of Combinatorial Designs*, vol. 14, p. 124–138, March 2006.

[45] L. Moura, J. Stardom, B. Stevens, L. Williams, A. Moura, J. Stardom, B. Stevens, and A. Williams, "Covering arrays with mixed alphabet sizes," *Journal of Combinatorial Designs*, vol. 11, pp. 413–432, October 2003.

[46] C. J. Colbourn, S. S. Martirosyan, T. V. Trung, and R. A. Walker II, "Roux-type constructions for covering arrays of strengths three and four," *Designs, Codes and Cryptography*, vol. 41, pp. 33–57, October 2006.

[47] G. Roux, *k-propriétés dans des tableaux de n colonnes; cas particulier de la k-surjectivité et de la k-permutivité.* PhD thesis, University of Paris, 1987.

[48] D. T. Tang and C. L. Chen, "Iterative exhaustive pattern generation for logic testing," *IBM Journal of Research and Development*, vol. 28, pp. 212–219, March 1984.

[49] G. B. Sherwood, S. S. Martirosyan, and C. J. Colbourn, "Covering arrays of higher strength from permutation vectors," *Journal of Combinatorial Designs*, vol. 14, no. 3, pp. 202–213, 2006.

[50] R. A. Walker II and C. J. Colbourn, "Tabu search for covering arrays using permutation vectors," *Journal of Statistical Planning and Inference*, vol. 139, pp. 69 – 80, January 2009.

[51] J. Bracho-Rios, J. Torres-Jimenez, and E. Rodriguez-Tello, "A new backtracking algorithm for constructing binary covering arrays of variable strength," in *MICAI 2009: Advances in Artificial Intelligence* (A. Aguirre, R. Borja, and C. Garciá, eds.), vol. 5845 of *Lecture Notes in Computer Science*, pp. 397–407, Springer Berlin Heidelberg, 2009.

[52] J. E. Bracho-Ríos, "A branch & bound algorithm to construct small instances of binary covering arrays of variable strength," Master's thesis, CINVESTAV-Laboratorio de Tecnologías de Información, 2010.

[53] J. Yan and J. Zhang, "Backtracking algorithms and search heuristics to generate test suites for combinatorial testing," in *30th Annual International Computer Software and Applications Conference (COMPSAC 2006)*, pp. 385–394, 2006.

[54] B. Hnich, S. D. Prestwich, E. Selensky, and B. M. Smith, "Constraint models for the covering test problem," *Constraints*, vol. 11, pp. 199–219, July 2006.

[55] B. M. W. Cheng, K. M. F. Choi, J. H. M. Lee, and J. C. K. Wu, "Increasing constraint propagation by redundant modeling: an experience report," *Constraints*, vol. 4, pp. 167 – 192, May 1999.

[56] D. López-Escogido, "Cálculo de covering arrays mixtos empleando el problema de satisfactibilidad proposicional.," Master's thesis, CINVESTAV-Laboratorio de Tecnologías de Información, 2008.

[57] D. Cohen, S. Dalal, A. Kajla, and G. C. Patton, "The automatic efficient test generator (aetg) system," in *Software Reliability Engineering, 1994. Proceedings., 5th International Symposium on*, pp. 303–309, 1994.

[58] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The aetg system: an approach to testing based on combinatorial design," *IEEE Transactions on Software Engineering,*, vol. 23, pp. 437–444, July 1997.

[59] R. C. Bryce and C. J. Colbourn, "The density algorithm for pairwise interaction testing," *Software Testing, Verification and Reliability.*, vol. 17, pp. 159 – 182, sep 2007.

[60] Y. Lei and K. Tai, "In-parameter-order: a test generation strategy for pairwise testing," in *High-Assurance Systems Engineering Symposium, 1998. Proceedings. Third IEEE International*, pp. 254–261, nov 1998.

[61] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "Ipog: A general strategy for t-way software testing," in *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, ECBS '07, (Washington, DC, USA), pp. 549–556, IEEE Computer Society, 2007.

[62] M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Refining the in-parameter-order strategy for constructing covering arrays," *Journal of Research of the National Institute of Standards and Technology*, vol. 113, no. 5, pp. 287 – 297, 2008.

[63] A. Calvagna and A. Gargantini, "T-wise combinatorial interaction test suites construction based on coverage inheritance," *Softw. Test. Verif. Reliab.*, vol. 22, pp. 507–526, nov 2012.

[64] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* A Bradford Book, 1992.

[65] J. Stardom, "Metaheuristics and the search for covering and packing arrays," Master's thesis, Simon Fraser University, 2001.

[66] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," in *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, pp. 72–77 vol.1, sept. 2004.

[67] F. Glover, "Tabu search - part 1," *Operations Research Society of America*, vol. 1, no. 3, pp. 190–206, 1989.

[68] K. J. Nurmela, "Upper bounds for covering arrays by tabu search," *Discrete Applied Mathematics*, vol. 138, pp. 143–152, March 2004.

[69] L. González-Hernández, N. Rangel-Valdez, and J. Torres-Jimenez, "Construction of mixed covering arrays of variable strength using a tabu search approach," *Lecture Notes in Computer Science*, vol. 6508, pp. 51–64, 2010.

[70] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[71] E. Covarrubias-Flores, "Cálculo de covering arrays binarios de fuerza variable, usando un algoritmo de recocido simulado," Master's thesis, CINVESTAV-Laboratorio de Tecnologías de Informacón, 2008.

[72] M. B. Cohen, C. J. Colbourn, and A. A. C. H. Ling, "Constructing strength three covering arrays with augmented annealing," *Discrete Mathematics*, vol. 308, no. 13, pp. 2709–2722, 2003.

[73] J. Torres-Jimenez and E. Rodriguez-Tello, "New bounds for binary covering arrays using simulated annealing," *Information Sciences: an International Journal*, vol. 185, pp. 137–152, February 2012.

[74] H. Avila-George, J. Torres-Jimenez, and V. Hernández, "New bounds for ternary covering arrays using a parallel simulated annealing," *Mathematical Problems in Engineering*, vol. 2012, p. 19, 1 2012.

[75] L. González-Hernández, J. Torres-Jimenez, and N. Rangel-Valdez, "An exact approach to maximize the number of wild cards in a covering array," in *Proceedings of 10th Mexican International Conference on Artificial Intelligence (MICAI 2011)*, vol. 7094, (In Puebla,Mexico. At Complejo Cultural Universitario), pp. 210–221, November 26-December 4 2011.

[76] P. Quiz-Ramos, "Maximización de renglones constantes para covering arrays," Master's thesis, CINVESTAV-Laboratorio de Tecnologías de Información, 2010.

[77] J. Torres-Jiménez, N. Rangel-Valdez, H. Avila-George, and A. L. González-Hernández, "Maxclique problem solved using sql," in *Proceedings of The Third International Conference on Advances in Databases, Knowledge and Data Applications (IARIA 2011)*, pp. 83–88, January 2011.

[78] C. J. Colbourn, G. Kéri, P. P. Rivas Soriano, and J. C. Schlage-Puchta, "Covering and radius-covering arrays: Constructions and classification," *Discrete Applied Mathematics*, vol. 158, pp. 1158–1180, June 2010.

[79] F. Gray, "Pulse code communication," March 1953.

[80] D. Guan, "Generalized gray codes with applications," *Proceedings of the National Science Council ROC(A)*, vol. 22, pp. 841–848, 1998.

[81] C. J. Colbourn, "Covering array tables for $t = 2, 3, 4, 5, 6$. last time accessed on november 21, 2013." http://www.public.asu.edu/ ccolbou/src/tabby/catable.html, Nov 2013.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL IPN

UNIDAD TAMAULIPAS

Cd. Victoria, Tamaulipas, a 6 de diciembre de 2013.

Los abajo firmantes, integrantes del jurado para el examen de grado que sustentará el C. Aldo Josimar González Gómez, declaramos que hemos revisado la tesis titulada:

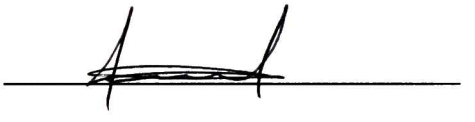
"Construcción de Covering Arrays Utilizando Vectores Inicializadores"

Y consideramos que cumple con los requisitos para obtener el grado de Maestro en Ciencias en Computación.

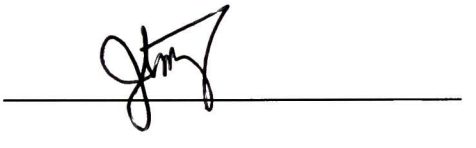
Atentamente,

Dr. Nelson Rangel Valdez

Dr. José Juan García Hernández



Dr. José Torres Jiménez



Km. 5.5 Carretera a Soto la Marina. Ciudad Victoria, Tamaulipas. CP 87130
Parque Científico y Tecnológico TECNOTAM
Tels.: (834) 1070220 y 316 6600
E-mail: admin@tamps.cinvestav.mx