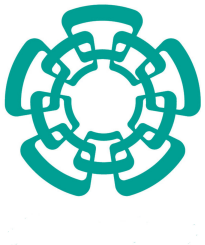


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Cinvestav Tamaulipas

Generalización de la Construcción de Vectores Inicializadores

Tesis que presenta:

Gildardo Maldonado Martínez

Para obtener el grado de:

Maestro en Ciencias en Ingeniería y Tecnologías Computacionales

Director de la Tesis:
Dr. José Torres Jiménez

Cd. Victoria, Tamaulipas, México. Octubre, 2017

La tesis presentada por Gildardo Maldonado Martínez fue aprobada por:

_____

Dr. Said Polanco Martagón

Dr. Ricardo Landa Becerra

Dr. José Torres Jiménez, Director

Cd. Victoria, Tamaulipas, México., 27 de Octubre de 2017

Para mamá y papá,
cada uno de mis logros es gracias a ustedes.

Agradecimientos

- Agradezco a Dios por permitirme culminar una etapa más.
- A la Unidad Cinvestav Tamaulipas por facilitar la infraestructura necesaria y al personal administrativo por todas sus atenciones.
- Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado para la realización de este trabajo de tesis.
- A mi asesor, el Dr. José Torres Jiménez por todos los conocimientos que pude aprender de él, su paciencia y consejos brindados.
- A mis amigos y compañeros de maestría, en especial a Carlos y Daniel, quienes hicieron más llevadera esta etapa en mi vida mediante anécdotas que recordaré siempre.
- A mis padres y hermanos por su apoyo incondicional en las decisiones que he tomado en mi vida, preocuparse por mi y estar al pendiente de que nada me falte.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Resumen

Generalización de la Construcción de Vectores Inicializadores

por

Gildardo Maldonado Martínez
Unidad Cinvestav Tamaulipas
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2017
Dr. José Torres Jiménez, Director

Hoy en día existe una fuerte dependencia de los sistemas computacionales para la toma de decisiones en distintos sectores (economía, salud, etc), dependiendo del sector en el que se incorpore, el costo de un error en la salida del sistema puede ser muy alto, y validar de manera exhaustiva todos los posibles escenarios no siempre es conveniente debido a la gran cantidad de recursos, como el tiempo y costo, que demandan. Los Covering Arrays (CAs) son objetos combinatorios que permiten construir un conjunto de N casos de prueba para componentes de software, con los que se asegura que todas las interacciones de tamaño t entre k parámetros se encuentran cubiertos, para cada parámetro que admite un valor dentro de un conjunto de v posibles valores. Si bien es cierto que las aplicaciones más comunes se pueden encontrar en el área de pruebas del software, recientemente han sido empleados para realizar pruebas en hardware y aspectos relacionados a la seguridad como detección de troyanos en hardware. Existe un conjunto de métodos para la construcción de CAs los cuales se clasifican como algebraicos, voraces, exactos y metaheuristicos. En la literatura se ha propuesto el construir un CA a partir de un Vector Inicializador (VI) aplicando operaciones de rotación y traslación, aunque la metodología para la obtención del VI no se encuentra muy desarrollada, salvo algunas construcciones con valores específicos de t, k y v. En este trabajo se presentan dos generalizaciones; una generalización de operadores que engloba las construcciones reportadas en el estado del arte con respecto a VIs y que además admite la introducción de nuevos operadores que permiten diversificar las construcciones actuales; y una generalización para la búsqueda de VIs en la que se propone una metodología original mediante la cual es posible obtener VIs trasladando el

problema al dominio de grafos.

Los resultados muestran que la generalización propuesta construye 14 casos para fuerza $t = 2$ los cuales son óptimos ya que el número de renglones del CA resultante es igual a las v^t interacciones posibles. Además 29 cotas fueron igualadas mediante construcciones obtenidas de forma directa y 34 cotas se igualaron mediante un proceso de postoptimización aplicado a un CA construido a partir de un VI. Aunque las cotas obtenidas sólo igualaron cotas reportadas actualmente en el estado del arte, esta generalización podría obtener resultados para casos de t, k y v mayores.

Abstract

Generalization of the Construction of Starter Vectors

by

Gildardo Maldonado Martínez
Information Technology Laboratory, CINVESTAV-Tamaulipas
Center for Research and Advanced Studies of the National Polytechnic Institute, 2017
Dr. José Torres Jiménez, Advisor

Nowadays there is a strong computer systems dependence for decision-making in different sectors (economy, healthcare, etc.), depending on the sector in which it is incorporated, the cost of an error in the output of the system can be very expensive and to validate exhaustively all the possible cases is not always convenient due to the amount of resources, such as time and cost, that demand. Covering Arrays (CAs) are combinatorial objects that allow to develop a set of N test cases for software components, which ensures that all the interactions of size t between k parameters are covered, for each parameter that accept a value within a set of v possible values. While it is true that the most common applications can be found in the software testing area, they have been used recently to perform hardware testing and aspects relating to the security such as hardware trojan detection. There are a set of methods for CAs construction which are classified as algebraic, exact, greedy and metaheuristic. In the literature, it has been proposed to construct a CA from a Starter Vector (SV) by applying rotation and translation operations, although the methodology for obtaining the SV is still poorly developed, except for some constructions with specific values of t, k and v. In this work two generalizations are presented; a generalization of operators that includes the constructions reported in the state of the art with regard to SVs and that also admits the introduction of new operators that allow to diversify the current constructions; and a generalization for the search of SVs in which an original methodology is proposed through which it is possible to obtain SVs by mapping the problem to the graph domain.

The results show that the proposed generalization constructs 14 cases for strength $t = 2$ which are optimal since the number of rows of the resulting CA is equal to the v^t possible interactions. In

addition 29 best-known results were matched by a directly construction and 34 best-known results were matched by a post-optimization process applied to a CA constructed from a SV. Although the results obtained only were matched with the currently reported in the state of the art, this generalization could obtain competitive results for greater cases of t, k and v.

Nomenclatura

OA	Arreglo Ortogonal
CA	Covering Array
CAN	Número del Covering Array
VI	Vector Inicializador
VIG	Vector Inicializador Generalizado
PCCA	Problema de Construcción de Covering Arrays
PBVI	Problema de Búsqueda de Vectores Inicializadores
PMC	Problema de Máximo Clique

1

Introducción

Dentro de este capítulo se introduce el concepto de covering arrays como objetos matemáticos y su impacto en el diseño de casos de prueba para la validación del correcto funcionamiento de componentes, haciendo énfasis en la importancia de contar con sistemas altamente funcionales especialmente en ambientes en los que el resultado arrojado por este sistema pueda influenciar en la toma de decisiones. A continuación se aborda el problema de construcción de estos objetos matemáticos y la introducción del concepto de vectores inicializadores como alternativa de solución para construir covering arrays, analizando sus ventajas y las limitaciones actuales de esta construcción. Finalmente, se presenta de forma breve como se encuentra estructurado este documento de tesis.

1.1 Antecedentes y motivación

El Instituto Nacional de Estándares y Tecnología (NIST, por sus siglas en inglés) reportó en el 2003 que los costos por realizar pruebas de software de manera inadecuada ascienden a $59.5 billones de dólares al año [31]. Diseñar un conjunto de pruebas destinadas a la validación del funcionamiento de

un componente resulta una tarea delicada, debido a que este conjunto de pruebas debe proporcionar la certeza de que han sido contempladas todas las interacciones entre los parámetros de entrada del componente, lo que resulta impráctico por cuestiones de tiempo, dinero. Por esta razón, el proceso de validación no siempre se realiza de forma adecuada y las consecuencias pueden variar dependiendo del sector en el que se desempeñe el sistema.

Según estudios reportados por Kuhn [30], la mayoría de los errores en componentes de software se debe a la interacción de a lo más seis parámetros, por lo que una alternativa a las pruebas exhaustivas es hacer uso de objetos combinatorios, como los covering arrays, que reduzcan el número de casos de pruebas manteniendo la cobertura de interacciones de cierto tamaño entre parámetros de entrada.

Definición 1: Covering Array (CA)

Sean $N, t, k,$ y v enteros positivos, un covering array $A = \mathsf{CA}(N; t, k, v)$ es un arreglo de tamaño $N \times k$, donde v es el alfabeto, t la fuerza del arreglo y $A = (a_{ij}),\ 0 \leq i < N,\ 0 \leq j < k$, sobre $\mathbb{Z}_v = \{0, 1, \ldots, v-1\}$ con la propiedad que para cualesquiera t distintas columnas $0 \leq c_0 < c_1 \ldots < c_{t-1} < k$, y cualquier miembro $(x_0, x_1, \ldots, x_{t-1})$ de $\mathbb{Z}_v^t$ exista al menos una fila r tal que $x_i = a_{r,c_i}$ para todo $0 \leq i < t$.

Los CAs son generalmente empleados para construir casos de prueba de componentes de software, los cuales tienen la propiedad de cubrir todas las interacciones de tamaño t entre parámetros o números de entradas de un componente (véase Figura 1.1.). Si bien es cierto que las aplicaciones más comunes se pueden encontrar en el área de pruebas del software, recientemente han sido empleados para realizar pruebas en hardware [22] y aspectos relacionados a la seguridad como detección de troyanos en hardware [28].

Definición 2: Problema de Construcción de Covering Arrays (PCCA)

El problema de construcción de covering arrays (PCCA) [52] consiste en construir un covering array $\mathsf{CA}(N; t, k, v)$ dado los parámetros t, k, v tal que número de filas N sea mínimo.

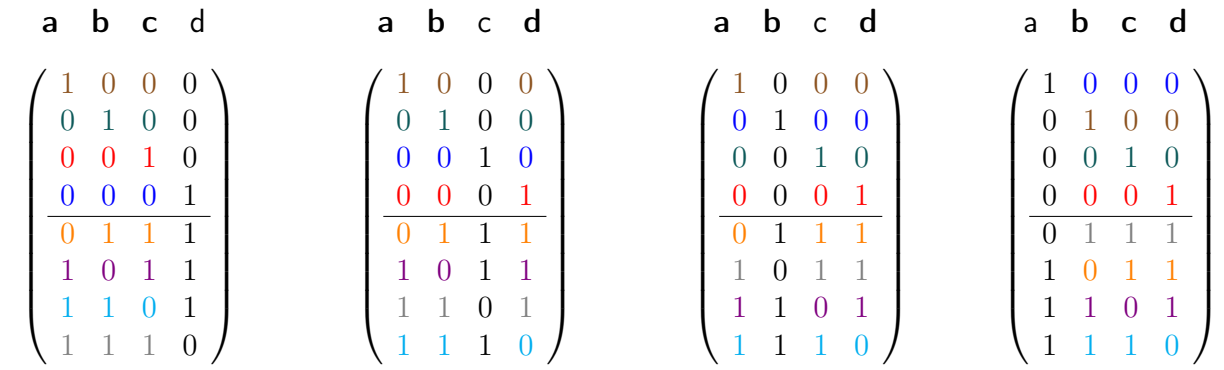
El espacio de búsqueda del PCCA denotado por χ satisface:

$$v^t \leq \chi \leq \binom{v^k}{N} \leq v^{kN} \tag{1.1}$$

Definición 3: Número del Covering Array (CAN)

El valor de N más pequeño para el cual existe un CA dado los valores t, k, v, se denomina número del covering array (CAN, por sus siglas en inglés) y se denota de la siguiente manera:

$$\text{CAN}(t,k,v) = min(N|\ \exists \text{CA}(N;t,k,v)) \tag{1.2}$$



Cobertura para columnas {a,b,c}. Cobertura para columnas {a,b,d}. Cobertura para columnas {a,c,d}. Cobertura para columnas {b,c,d}.

Figura 1.1: Proceso de validación de un CA$(8;3,4,2)$.

Suponga el caso en el que se requiere realizar pruebas de software en el que se tiene un panel compuesto de 34 interruptores, es impráctico validar $2^{34} = 1.7 \times 10^{10}$ configuraciones posibles, aunque empleando un CA de fuerza $t = 3$ (es decir que todas las interacciones entre 3 parámetros se encuentran cubiertas) se puede realizar el proceso de validación para el panel de interruptores con sólo 24 pruebas y para fuerza $t = 4$ con sólo 63 pruebas [31]. Es aquí donde la investigación en el desarrollo de CAs óptimos cobra mayor relevancia, ya que contar con un número reducido de pruebas que garantice que cierto elemento está completamente libre de errores, ocasionados por interacciones de cierto tamaño, se traduce en ahorros de tiempo y recursos dedicados para esta actividad los cuales

generalmente son limitados [23].

1.2 Descripción del problema

Aunque no se ha demostrado que el problema de construcción de covering arrays pertenezca a la clase de problemas NP-Completo, el espacio de búsqueda crece de manera exponencial a medida que crecen los parámetros de entrada k y sus valores v, por lo que han sido propuestos diferentes métodos [52] para obtener CAs óptimos o casi óptimo. De manera general, una gran parte de estos métodos buscan asignar cada elemento para la matriz de dimensiones $N \times k$ hasta que la propiedad de cobertura, según los parámetros especificados, ha sido satisfecha. Una alternativa resulta muy atractiva para la construcción de CAs es la de emplear un solo vector de tamaño k, denominado como Vector Inicializador (VI), para construir a partir de este los $N - 1$ renglones restantes. La manera en el que se construye un CA empleando un VI es aplicando operaciones de manipulación, comúnmente son operaciones de rotación y traslación (véase Figura 1.2.).

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

Figura 1.2: CA$(10; 3, 5, 2)$ construido mediante el VI = {1,0,0,0} aplicando k rotaciones y v traslaciones.

1.3 Justificación

Aunque la construcción de CAs usando el concepto de VIs ha demostrado que puede producir construcciones competitivas por los trabajos presentados en [6][41][19][35][11], al parecer se han agotado los casos que pueden ser construidos mediante los operadores que han sido propuestos en el estado del arte. Por esta razón es importante un mecanismo general que permita la introducción de nuevos operadores, además de establecer una formula general que controle el número de filas N que se pueden obtener para un CA(t, k, v) y de esta manera aproximar el resultado de la construcción a una región factible tal que:

$$LB \leq N \leq UB \tag{1.3}$$

Donde:

- LB es la cota mínima para la cual pudiera existir solución.
- UB es la cota máxima para el cual se tiene un CA ya conocido.
- N es el número de filas del CA construido con el VI.

Este trabajo presenta una generalización de las construcciones reportadas en el estado del arte con respecto a Vectores Inicializadores. Además, hasta nuestro conocimiento, la metodología para la obtención de VIs solo se encuentra desarrollada para casos específicos en el que los parámetros k y v necesariamente deben ser números primos o una potencia de primo [11][45][60], variando dicha metodología dependiendo del parámetro de fuerza t. En cuanto a una metodología general sólo se tiene conocimiento de una metodología [19] que reduce el espacio de búsqueda de v^k a $\frac{v^k}{k}$ empleando el conocimiento que el número de combinaciones faltantes para un CA A construido mediante un VI S será el mismo que el número de combinaciones faltantes para un CA B mediante un VI S', siempre y cuando el vector S' pueda ser expresado como una rotación del vector S. Aunque esto

represente una reducción del espacio de búsqueda no se descartan soluciones que contienen un alto grado de redundancia, por esta razón se propone una nueva metodología mediante la cual es posible obtener VIs trasladando el problema al dominio de grafos. Trasladar el problema al dominio de grafos nos permite introducir un algoritmo general de búsqueda de VIs abordándolo como un problema de máximo clique, el cual ha sido ampliamente explorado en el estado del arte.

Definición 4: Grafo

Un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ es un par ordenado formado por un conjunto finito no vacío $\mathcal{V}$ que representa el conjunto de nodos y un conjunto $\mathcal{E}$ de aristas conformado por pares no ordenados de nodos del conjunto $\mathcal{V}$.

Definición 5: Máximo clique

Sea $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ un grafo, el máximo clique [62] se define como el máximo subconjunto de nodos de $\mathcal{G}$ que forman un grafo completo.

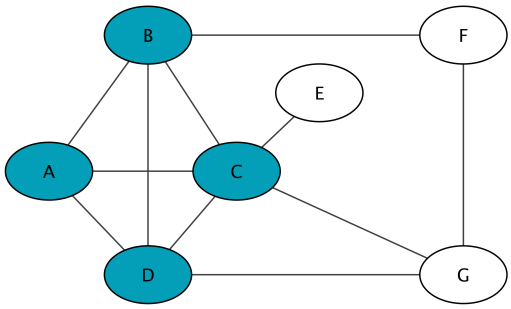


Figura 1.3: Máximo clique de tamaño cuatro formado por el subconjunto de nodos {A,B,C,D}.

La ventaja que representa trabajar el problema de búsqueda del máximo clique en el dominio de grafos es que nos permite modelar mediante enlaces aquellas combinaciones que pueden ser concentradas en un solo VI y descartar los enlaces existentes entre pares de combinaciones redundantes para enfocar la búsqueda en regiones que aseguren una alto nivel de cobertura, sin mencionar que actualmente existen algunas implementaciones estándar, como *Cliquer* [43] y *MaxCLQ* [34], las cuales se encuentran ampliamente documentadas y que resuelven el problema del máximo

clique dado un grafo cualquiera, por lo que únicamente es necesario modelar el grafo de forma adecuada para la obtención de VIs.

1.4 Organización del documento de tesis

Este documento de tesis se encuentra estructurado en seis capítulos. Los cuatro capítulos restantes se describen de manera breve a continuación:

- Capítulo 2: Estado del arte. Documenta los diversos métodos que existen actualmente para la construcción de covering arrays. Dichos métodos se encuentran organizados como algoritmos algebraicos, exactos, voraces y metaheurísticos.

- Capítulo 3: Generalización propuesta para la construcción de vectores inicializadores. Presenta de manera formal el concepto de vector inicializador y describe las propiedades que son de interés para el desarrollo de este trabajo de investigación. Para finalizar, se introduce el concepto de vectores inicializadores generalizados como resultado de la generalización propuesta para las operaciones que son aplicadas al vector inicializador en la construcción de covering arrays.

- Capítulo 4: Búsqueda de vectores inicializadores en el dominio de grafos. Describe los algoritmos desarrollados para este trabajo de investigación en el que se incluye el algoritmo para el mapeo del problema de la búsqueda de vectores inicializadores al dominio de grafos, el algoritmo de búsqueda del máximo clique adaptado para este problema en específico, y la interpretación del resultado obtenido en el dominio de grafos (máximo clique) a un vector inicializador.

- Capítulo 5: Experimentación y resultados. Presenta la experimentación empleada para la construcción de covering arrays a través del concepto de vectores inicializadores generalizados, así como la descripción de los resultados obtenidos.

- Capítulo 6: Conclusiones y trabajo futuro. Se discutirá acerca de las conclusiones y los

principales aportes en este trabajo de tesis, así como aspectos relevantes que se derivan como trabajo futuro.

1.5 Resumen

En este capítulo se mostró un estudio en el que se analizan las consecuencias económicas de un mal funcionamiento en los sistemas computacionales, así como evidenciar que los covering arrays son una opción práctica para el diseño de casos de prueba como alternativa a realizar pruebas exhaustivas. Se introdujo el concepto de vectores inicializadores señalando las ventajas que provee la construcción de covering arrays bajo este concepto, además de indicar una alternativa para la búsqueda de dichos vectores trasladando el dominio del problema al dominio de grafos. En el siguiente capítulo se presentan los trabajos con respecto a la construcción de covering arrays.

2

Estado del arte en la construcción de covering arrays

A continuación se presenta de manera general los métodos propuestos actualmente para la construcción de covering arrays. Estas construcciones son clasificadas como métodos algebraicos, exactos, voraces y metaheurísticos.

2.1 Métodos algebraicos

Los métodos algebraicos presentan como característica que construyen, de manera directa, covering arrays mediante el uso de fórmulas u operaciones. En este proceso se involucran el uso de objetos matemáticos como arreglos, grupos, campos finitos, etc.

2.1.1 Construcción de arreglos ortogonales de índice unitario

En 1952, Bush [4] propuso su propio modelo de construcción de arreglos ortogonales de índice unitario $\text{OA}_\lambda(v^t, t, v+1, v)$ donde $\lambda = 1$ para alfabetos $v = p^\alpha$ donde p es un número primo y $\alpha \geq 0$, mediante operaciones en los campos finitos de Galois $GF(p^\alpha)$.

Sea p un número primo, para $GP(p)$ los elementos que conforman el campo finito son el conjunto de todos los enteros *mod* p. Por ejemplo los elementos en $GP(3)$ son $\{1, 2, 3\}$ Para el caso de potencia de primos $GP(p^\alpha)$ los elementos pueden ser expresados como polinomios $P(x)$ de grado $n-1$ con coeficientes $a_i \in GP(p)$. Tomando cada combinación para los coeficientes:

$$P_i(x) = \sum_{r=0}^{\alpha-1} a_{ir} x^r \text{ para } i = 0, 1, \ldots, p^\alpha - 1 \tag{2.1}$$

Zero-sum: Forma un arreglo $v^t \times t$ a partir de todas las t-tuplas en base a $0, \ldots, v-1$, y añade una columna tal que el valor para cada elemento de la nueva columna sea el negativo de la suma de las t entradas en cada fila. El arreglo $v^t \times t+1$ es un $\text{OA}(v^t; t+1; v; t)$. Este arreglo tiene la propiedad de que la suma de todos sus elementos por fila es igual a cero.

En 2010, Torres-Jimenez [56] diseñó un método basado en prueba y error para encontrar polinomios primitivos que a la vez construía la tabla de logaritmos y anti-logaritmos necesarios para realizar las multiplicaciones en los campos finitos de Galois.

Para $GF(2^2)$:

	e_0	e_1	e_2	e_3	∞
$y_0(x) = e_0x + e_0$	0	0	0	0	0
$y_1(x) = e_0x + e_1$	1	1	1	1	0
$y_2(x) = e_0x + e_2$	2	2	2	2	0
$y_3(x) = e_0x + e_3$	3	3	3	3	0
$y_4(x) = e_1x + e_0$	0	1	2	3	1
$y_5(x) = e_1x + e_1$	1	0	3	2	1
$y_6(x) = e_1x + e_2$	2	3	0	1	1
$y_7(x) = e_1x + e_3$	3	2	1	0	1
$y_8(x) = e_2x + e_0$	0	2	3	1	2
$y_9(x) = e_2x + e_1$	1	3	2	0	2
$y_{10}(x) = e_2x + e_2$	2	0	1	3	2
$y_{11}(x) = e_2x + e_3$	3	1	0	2	2
$y_{12}(x) = e_3x + e_0$	0	3	1	2	3
$y_{13}(x) = e_3x + e_1$	1	2	0	3	3
$y_{14}(x) = e_3x + e_2$	2	1	3	0	3
$y_{15}(x) = e_3x + e_3$	3	0	2	1	3

Figura 2.1: Construcción de arreglos ortogonales de índice unitario para $v = 4$ y $t = 2$.

2.1.2 Construcción de CAs de fuerza dos y alfabeto binario

Rényi [46] propuso un algoritmo exacto para el caso particular de construir CAs óptimos de fuerza $t = 2$ y vocabulario $v = 2$ dado un valor N par. De manera independiente este algoritmo fue modificado para cualquier valor N, dicha modificación se encuentra documentada por Katona[26], así como por Kleitman y Spencer [29].

El número óptimo de columnas k para un covering array CA$(N; 2, k, 2)$ dado N filas, se encuentra delimitado por la Inecuación 2.2.

$$k \leq \binom{N-1}{\lceil \frac{N}{2} \rceil} \tag{2.2}$$

La Inecuación 2.2 describe el algoritmo para su construcción en el que inicialmente se define una matriz $N \times k$ en la que la primera fila se encuentra definida por un renglón de elementos en cero. Las $N - 1$ filas restantes son definidas columna por columna por las combinaciones de $\lceil \frac{N}{2} \rceil$ unos y $N - 1 - \lceil \frac{N}{2} \rceil$ ceros.

Por ejemplo para $N = 6$, el máximo valor para k con base en la Inecuación 2.2 es $k = 10$, por lo que se construye una matriz de 6×10. La primera fila de la matriz es inicializada con ceros y las filas restantes son definidas columna por columna con las $\binom{N-1}{\lceil \frac{N}{2} \rceil} = \binom{5}{3} = 10$ combinaciones de $\lceil \frac{N}{2} \rceil = 3$ unos y $N - 1 - \lceil \frac{N}{2} \rceil = 2$ ceros. El resultado se muestra en la Figura 2.2.

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

Figura 2.2: Ejemplo de la construcción para $t = v = 2$. CA$(6; 2, 10, 2)$.

2.1.3 Vectores de pesos constantes

En 1983, Tang and Woo [50] utilizaron vectores de pesos constantes para aplicarlos como conjunto de pruebas de entradas de un circuito lógico. Este tipo de construcción emplea vectores s con un peso específico w. El concepto de peso se define como:

> Sea s un vector de tamaño k con entradas de $\{0, 1, \ldots, v-1\}$. El peso de s denotado por w, es la suma de los valores en el vector $w = \sum_{i=0}^{k} s_i$.

Un conjunto de casos de prueba (equivalentes a un covering array) se construye concatenando todos los vectores s que poseen el mismo peso w. Para el caso alfabeto binario $v = 2$ la construcción de un CA esta definido mediante:

> Dado k y t, $k > t$ un conjunto T de vectores binarios cubre todas las interacciones de tamaño t, si este contiene todos los vectores binarios con peso w tal que $w \equiv c \ mod \ k - t + 1$ para una constante $c \in \{0, 1, \ldots, k - t\}$.

Para valores con $k > t$ y $v > 2$ tenemos que:

Dado $k > t$ y $v > 2$ un conjunto T de vectores de alfabeto v cubre todas las interacciones de tamaño t si este contiene todos los vectores binarios con peso w tal que $w \equiv c$ *mod* m para una constante c, donde $m = (k-t)(v-1)+1$ y $0 \leq c \leq k-t$ y $0 \leq w \leq k(v-1)$.

Para $k = 4$, $v = 3$ y $t = 2$, $w \equiv 1$ *mod* $5, w = 1, 6$.

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
	2 1 1 2
	1 2 2 1
	1 2 1 2
	1 1 2 2

Figura 2.3: CA(14; 2;4;3) construido por los conjuntos de vectores de peso 1 y 6.

2.1.4 Construcción por grupos

En 2002, Chateauneuf y Kreaher [7] presentaron un nuevo método para construir CAs para t=3. Este se basaba en construir un CA a partir de matrices pequeñas, un vector inicializador y un grupo G aunque no se definía una estrategia precisa que permita obtener el vector inicial y el grupo de G. Posteriormente en 2005, Meagher y Stevens [41] extendieron la idea presentada por Chateauneuf y Kreaher, presentando una estrategia para obtener el vector inicializador y la selección del grupo G.

La construcción de un CA por grupos consiste en seleccionar un grupo, $G < Sym_g$ y encontrar un vector $w \in \mathbb{Z}_g^t$. Este vector se le denomina vector inicializador y este depende del grupo G. El vector inicializador es empleado para producir una matriz circular por medio de sus k rotaciones y

con el grupo se crean varias matrices que se concatenan para completar las condiciones de cobertura de un CA. Además es necesario agregar un vector C de tamaño k y con elementos en 0.

Para ejemplificar se toma el caso $G(e,(1,2)) < Sym_3$ y $w =$ {0, 1, 1, 1, 2}. A partir del VI se construye la matriz M:

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 & 2 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 & 1 \\ 1 & 1 & 2 & 0 & 1 \\ 1 & 1 & 1 & 2 & 0 \end{pmatrix} \qquad M_e = \begin{pmatrix} 0 & 1 & 1 & 1 & 2 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 & 1 \\ 1 & 1 & 2 & 0 & 1 \\ 1 & 1 & 1 & 2 & 0 \end{pmatrix} \qquad M_{(1,2)} = \begin{pmatrix} 0 & 2 & 2 & 2 & 1 \\ 1 & 0 & 2 & 2 & 2 \\ 2 & 1 & 0 & 2 & 2 \\ 2 & 2 & 1 & 0 & 2 \\ 2 & 2 & 2 & 1 & 0 \end{pmatrix}$$

Los elementos de G en M producen M_e y $M_{(1,2)}$. Concatenamos ambas matrices y agregamos un vector C de ceros y obtenemos el siguiente CA:

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

Figura 2.4: CA$(11; 2, 5, 3)$ construido utilizando el VI= $\{0, 1, 1, 1, 2\}$, el grupo $G(e,(1,2))$ y el vector C.

2.1.5 Potenciación de CAs

En 2005, Hartman [22] presentó un método para elevar al cuadrado el número de columnas k^2 de un CA$(N; t, k, v)$. Para elevar al cuadrado un covering array A = CA(N;t;k;v), se requiere de un arreglo ortogonal B = OA$_\lambda(v^t; 2, T(v,t)+1, k)$ con índice unitario $\lambda = 1$. Cada elemento de B determina el 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 potenciación de CAs 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-esima 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$(N(T(v,t)+1); t, k^2, v)$.

2.1.6 Construcción de CAs utilizando construcciones tipo Roux

En el 2006, Colbourn [13] demostró para fuerzas 2 y 3 que de manera recursiva se puede duplicar las k columnas de un covering array CA$(3, 2k, 2)$ a partir de un CA$(3, k, 2)$ y un CA$(2, k, 2)$ tal que CAN$(3, 2k, 2) \leq$ CAN$(3, k, 2)$ + CAN$(2, k, 2)$.

Sea A = CA$(3, k, 2)$, B = CA$(3, k, 2)$, es posible obtener una matriz C tal que C sea un CA$(3, 2k, 2)$ duplicando la matriz A y concatenando en la parte inferior la matriz B junto con su complemento. Tal como se muestra a continuación:

$$C = \begin{pmatrix} A & A \\ B & \overline{B} \end{pmatrix}$$

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

$$B = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad \overline{B} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

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

Figura 2.5: CA$(13; 3, 8, 2)$ resultante de la construcción tipo Roux a partir de un CA$(3, 4, 2)$, y un CA$(3, 4, 2)$.

2.1.7 Producto de CAs de fuerza dos

En 2006, Colbourn [13] presenta un método con el cual se puede realizar una multiplicación de dos CAs únicamente para construcciones de fuerza $t = 2$. Sean A y B covering arrays, tal que $A =$ CA$(N; 2, k, v)$ y $B =$ CA$(M; 2, l, v)$. El producto de $A \otimes B$ es una matriz $C =$ CA$(N \times M; 2, kl, v)$.

- $c_{i,(f-1)k+g} = a_{i,g}$ para $1 \leq i \leq N, 1 \leq f \leq l$ y $1 \leq g \leq k$
- $c_{N+i,(f1)k+g} = b_{i,f}$ para $1 \leq i \leq M, 1 \leq f \leq l$ y $1 \leq g \leq k$

Se realizan k copias de B y estas se agregan a las l copias de A.

N filas	a_{11}	a_{12}	$\cdots$	a_{1k}	a_{11}	a_{12}	$\cdots$	a_{1k}	$\cdots$	a_{11}	a_{12}	$\cdots$	a_{1k}
	a_{21}	a_{22}	$\cdots$	a_{2k}	a_{21}	a_{22}	$\cdots$	a_{2k}	$\cdots$	a_{21}	a_{22}	$\cdots$	a_{2k}
	$\vdots$				$\vdots$				$\cdots$	$\vdots$			
	a_{N1}	a_{N2}	$\cdots$	a_{Nk}	a_{N1}	a_{N2}	$\cdots$	a_{Nk}	$\cdots$	a_{N1}	a_{N2}	$\cdots$	a_{Nk}
M filas	b_{11}	b_{11}	$\cdots$	b_{11}	b_{12}	b_{12}	$\cdots$	b_{12}	$\cdots$	b_{1l}	b_{1l}	$\cdots$	b_{1l}
	b_{21}	b_{21}	$\cdots$	b_{21}	b_{22}	b_{22}	$\cdots$	b_{22}	$\cdots$	b_{2l}	b_{2l}	$\cdots$	b_{2l}
	$\vdots$				$\vdots$				$\cdots$	$\vdots$			
	b_{M1}	b_{M1}	$\cdots$	b_{M1}	b_{M2}	b_{M2}	$\cdots$	b_{22}	$\cdots$	b_{Ml}	b_{Ml}	$\cdots$	b_{Ml}

Figura 2.6: Estructura del producto $C = A \otimes B$. Donde $A = \mathsf{CA}(N; 2, k, v)$, $B = \mathsf{CA}(M; 2, l, v)$ y $C = \mathsf{CA}(N \times M; 2, kl, v)$

2.1.8 Vectores de permutación

En el 2006, Sherwood et al. ([47] y [61]) presentaron un método para construir CAs utilizando vectores de permutación.

Definimos $(\beta_0^{(i)}, \beta_1^{(i)}, \ldots, \beta_{t-1}^{i})$ como una representación de base v de los símbolos $i \in \{0, 1, \ldots, v-1\}$; tal que $i = \beta_0^{(i)} + v^1\beta_1^{(i)} + \ldots + v^{t-1}\beta_{t-1}^{(i)}$ donde $\beta_j^{(i)} \in \{0, 1, \ldots, v-1\}$ para $0 \leq j \leq t-1$. Para cada tupla $(t-1)$ $(h_1, h_2, \ldots, h_{t-1})$ con $h_j \in \{0, 1, \ldots, v-1\}$ para $1 \leq j \leq t-1$. Un vector de permutación $(\overrightarrow{h_1, h_2, \ldots, h_{t-1}})$ de tamaño v^t es un vector que tiene el símbolo $\beta_0^{(i)} + (h_1 \times \beta_1^{(i)}) + (h_2 \times \beta_2^{(i)}) + \ldots + (h_{t-1} \times \beta_{t-1}^{(i)})$ en la posición i para $0 \leq i \leq v^t - 1$.

Para realizar la construcción de coverings arrays se realiza una expansión de objetos matemáticos conocidos como Familias Hash de Cobertura Perfecta (CPHF, por sus siglas en inglés), en el que cada uno de sus elementos es un vector de permutación que al ser expandido producen un CA reduciendo el tiempo de computo. En este caso se presenta el ejemplo con $\mathsf{CPHF}(2; 10, 3, 3)$.

$$\begin{array}{cccccccccc} 11 & 00 & 22 & 21 & 01 & 02 & 10 & 11 & 02 & 12 \\ 10 & 01 & 11 & 11 & 00 & 22 & 01 & 02 & 20 & 12 \end{array}$$

Para construir el covering array se requiere tomar cada elemento como $t-1$ tupla de los v^{t-1}

símbolos (para este caso son 3^2 símbolos como tupla de 2 elementos sobre 3 símbolos). Para el primer elemento 11 $(h_1 = 1, h_2 = 1)$ se transforma como un vector de 3^3 elementos donde cada fila i es escrito como una tupla v^t. Para la fila i se define un su valor en base v en un vector de tamaño t. Es decir para $i = 0$ le corresponde 000 y para $i = 17$ le corresponde 122. Para la fila i = 000, el valor que se asigna al vector de v^t elementos se calcula como $0 \cdot 1 + 0 \cdot 1 + 0 = 0$. Continuado con:

$$i = 001 : 0 \cdot 1 + 0 \cdot 1 + 1 = 1$$

$$i = 002 : 0 \cdot 1 + 0 \cdot 1 + 2 = 2$$

$$\vdots$$

$$i = 001 : 2 \cdot 1 + 2 \cdot 1 + 2 = 0$$

Al unir todos estos resultados se obtiene el siguiente CA(51,3,10,3):

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

Figura 2.7: CA$(51, 3, 10, 3)^T$ construido mediante la expansión de los vectores de permutación de un CPHF(2;10,3,3).

2.1.9 Ciclotomía

En 2010, Coulbourn [11] propuso la ciclotomía para la construcción de CAs, este enfoque se basa en el uso de un vector inicializador $VI_{q,v}$ de tamaño q tal que q sea primo o potencia de primo y alfabeto v. Este vector es derivado a través de los campos finitos de Galois a través de la tabla de logaritmos presentada por Torres-Jimenez [56] en el 2010. Teniendo a T como la tabla de logaritmos, donde una celda en particular se denota por T_i, $0 \leq i \leq k-1$, el vector inicializador VI se construye por $VI_i = T_i$ *mod* v y $VI_0 = 0$.

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

Tabla 2.1: Tabla de logaritmos para GF(7).

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

Figura 2.8: CA$(7; 2, 7, 2)$ creado con el VI$= \{0, 0, 0, 1, 0, 1, 1\}$ $q = 7$ y $v = 2$.

Para valores con $k = 7$ y $v = 2$ se construye la tabla de logaritmos que se muestra en la tabla 2.1. y se crea el VI mediante la operación $VI_i = T_i$ *mod* v. Como resultado obtenemos el VI = {0, 0, 0, 1, 0, 1, 1} con el que es posible obtener un CA$(7; 2, 7, 2)$ (Figura 2.8).

2.1.10 Construcción a través de coeficientes binomiales

En 2015, Torres-Jimenez et. al. presentaron en [54] una construcción para covering arrays de alfabeto $v = 2$ usando coeficientes binomiales. Para construir un covering array binario CA$(N; t, k, 2)$ empleando coeficientes binomiales se plantea un algoritmo de ramificación y poda que encuentra una subcolección $\mathscr{C} = \binom{k}{r_1}, \binom{k}{r_2}, \ldots, \binom{k}{r_i}$ para $r = 0, \ldots, k$ tal que su yuxtaposición de sus filas

proporcione cobertura total con la menor cantidad de filas posible.

$\mathscr{C}$	
$\binom{4}{0}$	0 0 0 0
$\binom{4}{3}$	1 1 1 0
	1 1 0 1
	1 0 1 1
	0 1 1 1

Figura 2.9: CA(5;2,4,2) construido mediante la yuxtaposición de las filas formadas por los coeficientes $\binom{4}{0}$ y $\binom{4}{3}$.

2.1.11 Construcción a través de coeficientes trinomiales

En 2010, Martinez-Pena et. al. presentaron en [37] y [38] una construcción para covering arrays de alfabeto $v = 3$ mediante un conjunto de filas obtenidas de los coeficientes trinomiales.

Definimos los coeficientes trinomiales como:

Sean a, b, c, k enteros positivos tales que $0 \leq a, b, c \leq k$ un coeficiente trinomial (CT) denotado por $\binom{k}{a,b,c}$, es el coeficiente obtenido por la Ecuación 2.3:

$$\binom{k}{a, b, c} = \frac{(a+b+c)!}{a!\ b!\ c!} \tag{2.3}$$

Para la construcción de CA ternario se realizan los siguientes pasos:

1. Generar todos los coeficientes trinomiales de orden k.
2. Se busca el conjunto de coeficientes trinomiales que construyan un CA $\mathscr{C}$ con el menor número de renglones mediante un algoritmo de retroceso se implementa una técnica de ramificación y poda para recorrer el espacio de búsqueda de manera más eficiente (Figura 2.10(A)).
3. Transforma los coeficientes trinomiales $\mathscr{A}$ encontrados en el paso anterior en un CA (Figura 2.10(B)(C)).

$\mathscr{A} = \{\binom{3}{3,0,0}, \binom{3}{0,3,0}, \binom{3}{0,0,3}, \binom{3}{1,1,1}\}$
$\mathscr{C} = (\mathscr{R}^3_{3,0,0} + \mathscr{R}^3_{0,3,0} + \mathscr{R}^3_{0,0,3} + \mathscr{R}^3_{1,1,1})$
$N = \sum \mathscr{A} = 9$

A)

$\mathscr{A}$	$\mathscr{R}^k_{a,b,c}$
$\binom{3}{3,0,0}$	0 0 0
$\binom{3}{0,3,0}$	1 1 1
$\binom{3}{0,0,3}$	2 2 2
$\binom{3}{1,1,1}$	0 1 2
	1 2 0
	2 0 1
	0 2 1
	1 0 2
	2 1 0

B)

$\mathscr{C}$
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

C)

Figura 2.10: Proceso de construcción de un CA(9;2,3,3) mediante coeficientes trinomiales.

2.1.12 Torres de CAs

En 2013, Torres-Jimenez et al. [55] propone una metodología para construir CAs competitivos denominado Torres de Covering Arrays (TCA). La metodología se describe a continuación:

- Se generan todos los CAs no isomorfos de mínimo rango CA$(N; t, k, v)$ con el objetivo de usarlas como base de la torre de CAs.
- Se aplica iterativamente la construcción ε para los CAs no isomorfos de mínimo rango.
- Se reporta la torre de CAs construida con la mayor altura,

Dado un covering array CA$(N; t, k, v)$ denominado como A, el cual es tomado como base de la torre se pretende generar un CA B de fuerza mayor tal que $B = (M, t+1, k+1, v)$. En el caso de que A sea óptimo, el mínimo número de filas M de B es al menos $M = Nv$. Para generar un CA(M,t+1,k+1,v) se yuxtaponen v copias verticales de A y se traslada la j-ésima columna de la i-ésima copia añadiendo un valor c y aplicando módulo v a cada valor de la columna. La columna $k+1$ es construida colocando N símbolos cero y N símbolos uno hasta añadir N elementos $v-1$.

$$\underset{\text{CA(4,2,4,2)}}{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}} \qquad \delta = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad X = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \qquad \underset{\text{CA(8,3,4,2)}}{\left(\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{array}\right)}$$

Figura 2.11: CA(8,3,4,2) obtenido mediante un CA(4,2,4,2) de fuerza $t-1$ y la matriz δ, donde δ_i se emplea para realizar las traslaciones de la i-ésima copia del CA.

2.1.13 Registro de desplazamiento con retroalimentación lineal

En 2016, Tzanakis [60] introduce un nuevo método usando secuencias de registro de desplazamiento con retroalimentación lineal (LFSR, por sus siglas en inglés) para construir covering arrays basados en los resultados teóricos establecidos en el trabajo de Raaphorst. Esto produce covering arrays sobre los campos finitos con q elementos, fuerza t y tamaño $l(q^t-1)+1$, donde q es una potencia de primo y l, t son enteros tal que $l \geq 1$, $t \geq 2$.

Un LFSR con un polinomio característico f y valores iniciales $T = (b_0, \ldots, b_{m-1}) \in \mathbb{F}_q^m$ es una secuencia $S(f,T) = (a_0, a_1, a_2, \ldots)$ sobre $\mathbb{F}_q$ definido como:

$$a_i = \begin{cases} b_i & \text{si } 0 \leq i < m \\ -c_{m-1}a_{i-1} - c_{m-2}a_{i-2} - \cdots - c_1 a_{i-(m-1)} - c_0 a_{i-m} & \textbf{si } i \geq m \end{cases}$$

El resultado de este trabajo es una generalización de las 2 anteriores construcciones anteriormente propuestas empleando LFSR en [45]. En particular se hace uso de un algoritmo de retroceso para la generación y reducción del espacio de búsqueda que es definido por las propiedades de los campos finitos y generando collares binarios

$S(x^3 + x + 1, (1, 0, 0))$

i	s_i	LFSR
0	1	1
1	0	10
2	0	100
3	0(0) + 1(0) + 1(1) = 0 + 0 + 1 = 1	1001
4	0(1) + 1(0) + 1(0) = 0 + 0 + 0 = 0	10010
5	0(0) + 1(1) + 1(0) = 0 + 1 + 0 = 1	100101
6	0(1) + 1(0) + 1(1) = 0 + 0 + 1 = 1	1001011

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

Figura 2.12: CA$(15; 3, 7, 2)$ construido a partir de la secuencia LFSR $S = \{1, 0, 0, 1, 0, 1, 1\}$.

2.2 Métodos exactos

Se emplea búsqueda exhaustiva para encontrar covering arrays óptimos. Debido al enorme espacio de búsqueda se suele implementar técnicas de ramificación y poda con el objetivo de acotar dicho espacio de búsqueda.

2.2.1 Programación con restricciones

En 2006, Hinch [24] implementó una estrategia que comienza por restringir que cada combinación de parámetros debe aparecer una vez, sin embargo, ellos reportan que su implementación fue ineficiente, para lo cual diseñaron tres nuevas mejoras para el algoritmo. En su primer mejora, especifican que cada número en el rango 0 a $2^t - 1$ debe aparecer al menos una vez y a lo más $b-2^t+1$ veces en los b vectores de prueba en la columna correspondiente. En la segunda mejora, ellos usaron las variables de ambos enfoques para ser vinculadas. Finalmente, incorporaron un algoritmo de búsqueda local SAT. Demostraron que para problemas de tamaño moderado su enfoque puede alcanzar una solución óptima.

2.2.2 Algoritmo EXACT

En 2006, Yan y Zhang [63] desarrollaron una herramienta llamada EXACT (por sus siglas en inglés EXhaustive seArch of Combinatorial Test suites). El algoritmo fue diseñado para trabajar con Covering Arrays Mixtos (MCA), los cuales poseen un número heterogéneo de valores posibles para sus parámetros, por lo que el termino de cobertura indica que se deben encontrar al menos una vez las t-tuplas con el producto cartesiano de cada uno de los valores por parámetro $\{0, 1, \ldots, v_{j_0}\} \times \{0, 1, \ldots, v_{j_1}\} \times \ldots \times \{0, 1, \ldots, v_{j_{t-1}}\}$. Éste generador funciona haciendo asignaciones a los elementos del MCA cuando sea posible y retrocediendo cuando no es posible completar el MCA. No explora arrays isomorfos que ya han sido explorados, para ello solo considera arreglos que están ordenados lexicográficamente en filas y columnas. El reetiquetado de una columna no es considerado por el generador porque solo genera un MCA para los parámetros dados.

2.2.3 Construcción de CAs usando una transformación al problema de satisfactibilidad (SAT)

En 2008, López-Escogido [36] propuso la codificación SAT para generar CA de fuerza $t = 2$. Para cada elemento $m_{i,j}$ de la matriz M asociada con la instancia se introducen v variables booleanas $m_{i,j,x},\ 0 \leq x < v$. El modelo de transformación utiliza dos conjuntos de clausulas en forma de conjunción (z_1 y z_2) para asegurar que cada elemento de la matriz M toma solamente un valor del alfabeto y utiliza otro conjunto de clausulas (z_3) no conjuntiva para garantizar que las interacciones de las t-tuplas del CA sean satisfechas.

2.2.4 Algoritmo de retroceso para covering arrays binarios

En 2009, Bracho-Ríos [2] desarrolló un algoritmo de búsqueda para construir CAs binarios $v = 2$, de fuerza t y dimensiones $N \times k$. El algoritmo construye el CA columna por columna

en orden lexicográfico de las columnas para evitar la simetría de filas y columnas. Las columnas están compuestas por $\lceil N/2 \rceil$ ceros y $N - \lceil N/2 \rceil$ unos para construir CAs con símbolos balanceados. Antes de comenzar la búsqueda un bloque de t columnas se queda fijo y las primeras $N - 2^t$ filas del bloque se rellenan con ceros y las 2^t filas restantes se rellenan con las 2^t tuplas de tamaño t sobre el conjunto de símbolos {0, 1}. Supone una solución parcial con r columnas $(t \leq r < k)$ y sea l la ultima columna de la solución parcial. Para construir un CA de fuerza t y $r+1$ columnas se busca en todas las columnas mayores que l lexicográficamente hasta que se encuentre una que genere un CA con filas y columnas ordenadas lexicográficamente y si dicha columna no es encontrada el algoritmo regresa a la columna $r - 1$.

2.2.5 Construcción de CAs utilizando vectores inicializadores

En 2013, Gonzalez-Gomez [19], presenta una extensión de la construcción de CAs utilizando VIs con base en los trabajos presentados por Lobb et al. (2012) [35] y Colbourn (2009) [11]. En este trabajo se generan todos los posibles v^k vectores inicializadores empleando el código gray v-ario descrito en [20] aunque únicamente se validan aquellos que sean vectores canónicos (VC), el cual es un vector de k con una estructura definida por:

$$\mathsf{VC}_i = 0 \ \{\forall i |\ 0 \leq i < \mu\}$$

$$\mathsf{VC}_\mu = 1$$

$$\sum_{\forall i | \mathsf{VC}_i = 0} 1 \leq \mu \text{ para } \mu + 1 \leq i < k - 1$$

$$\mathsf{VC}_{k-1} \neq 0$$

Los vectores canónicos permiten reducir el espacio de búsqueda del VI de v^t a $\frac{v^t}{vk}$. Los operadores con los que trabaja esta extensión son los operadores de rotación, traslación y símbolos fijos.

2.2.6 Construcción de CAs no isomorfos

En 2002, Meagher [40] implementó la generación de covering array no isomorfos de fuerza $t = 2$ y $v = 2$. En los covering arrays existen tres tipos de isomorfismos que se obtienen al permutar filas, columnas y alfabeto. Este algoritmo tiene como objetivos realizar una búsqueda descartando CAs isomorfos. Primeramente se genera una columna a la vez el covering array hasta lograr las k columnas rechazando los covering Arrays que no son de rango mínimo.

En 2016, Torres-Jimenez e Izquierdo-Marquez [53], definen un algoritmo para la construcción de todos los CAs no isomorfos con orden lexicográfico mínimo que existen para una combinación particular de los parámetros N, t, k, y v. La estrategia del algoritmo se basa en extender los subarreglos de 0 a k columnas, una columna a la vez, validando que la nueva columna forma un CA de orden lexicográfico mínimo con las columnas anteriores. Si la nueva columna no forma un CA con el orden lexicográfico mínimo se rechaza. Cuando el algoritmo comprueba todas las opciones posibles para una columna particular se da marcha atrás a la columna anterior. La optimalidad para un CA$(N; t, k, v)$ es demostrada si el conjunto de CAs no isomorfos para un CA$(N - 1; t, k, v)$ se encuentra vacío. En este trabajo se construyeron los CAs no isomorfos para 70 combinaciones para los parámetros N, t, k y v, y se determinó optimalidad para CAN$(3, 13, 2) = 16$, CAN$(3, 14, 2) = 16$, CAN$(3, 15, 2) = 17$, CAN$(3, 16, 2) = 17$, y CAN$(2, 10, 3) = 14$.

2.3 Métodos voraces

Los metódos voraces se caracterizan por construir soluciones aceptables en poco tiempo de cómputo. Son una buena opción para aquellos parámetros de t, k y v que resulta impráctico resolverse de forma exacta.

2.3.1 Generador automático de pruebas eficiente

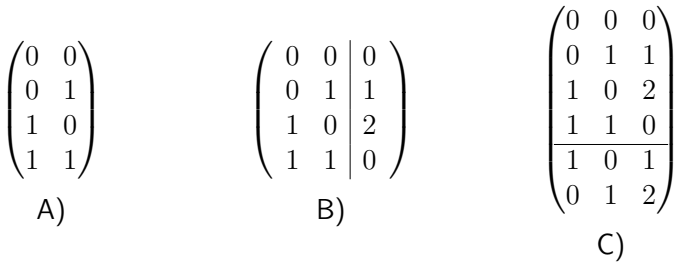
En 1997, Cohen et al. [8] propuso el generador automático de pruebas eficiente (sistema AETG, por sus siglas en inglés) es un generador de casos de prueba que son definidos por el usuario. El AETG es una herramienta que genera CA y MCA para $v = 2$ y $t = 2$, aunque sólo se usó para generar casos de $t = 2$ y $t = 3$. Comienza con un caso de prueba vacío y genera un renglón a la vez hasta que todas las interacciones de tamaño t entre los parámetros hayan sido cubiertas. Para generar el siguiente caso el algoritmo genera varios casos de prueba candidatos y selecciona el candidato que cubra la mayor cantidad de tuplas faltantes.

2.3.2 Algoritmo de orden por parámetro

En 2002, Lei y Tai [33] presentaron el algoritmo de orden por parámetro (IPO, por sus siglas en inglés) para generar casos de prueba en el que todas las interacciones de tamaño t entre los k factores son cubiertas. Como característica expande el CA por filas y por columnas. Posteriormente en 2007, Tai y Lei presentaron IPOG [32] como una generalización de este algoritmo para $t > 2$, así también en el 2008 Forbes [15] presentó IPOG-F como una mejora del algoritmo IPOG.

Los algoritmos IPO, IPOG e IPOG-F comparten la estrategia de utilizar un CA de $k - 1$ factores para construir el CA de k factores. La base de este proceso recursivo es el CA con $k = t$ factores. Posteriormente se añaden los factores $t + 1, t + 2, \ldots, k$ de uno en uno con base en lo siguiente:

- Crecimiento horizontal. Se añade una columna correspondiente al nuevo factor, las celdas son llenadas de tal manera que algunas tuplas del CA son cubiertas (Figura 2.13(B)).
- Crecimiento vertical. Nuevas filas se añaden para cubrir las tuplas faltantes del CA (Figura 2.13(C)).



A) B) C)

Figura 2.13: Construcción de un MCA$(6; 2, 3, 2^2 3^1)$ mediante el algoritmo de orden por parámetro.

2.3.3 Algoritmo de densidad determinista

En 2007, Bryce y Colbourn [3] propusieron el Algoritmo de Densidad Determinista (DDA, por sus siglas en inglés) como un generador de casos de prueba de fuerza $t = 2$. Este algoritmo como característica genera un caso de prueba a la vez. Para construir cada caso de prueba el algoritmo repetidamente fija un valor para cada factor basado en la densidad y actualiza las densidades locales y globales. La densidad es calculada de acuerdo al número de interacciones no cubiertas entre un par de factores i, j y la mayor cardinalidad de los dos factores. Cuando todos los factores han sido fijados, se emite el caso de prueba y el algoritmo continua con la construcción de los casos de prueba faltantes hasta que todas las interacciones son cubiertas.

2.3.4 Herencia de cobertura

En 2011, Calvagna y Gargantini [5] desarrollaron un algoritmo para generar MCA de cualquier fuerza. Se comienza con la creación de un MCA de fuerza t para los primeros t parámetros (Figura 2.14(A)), se asume que las cardinalidades de los k parámetros están en orden descendente, entonces tomando el caso base se agrega una nueva columna copiando la columna anterior. Todos los símbolos copiados que sean mayores al vocabulario de la columna se le asigna un símbolo x en la columna actual para indicar que están libres para ser asignados (Figura 2.14(B)(D)).

Como la nueva columna j es una copia de la columna $i < j$ cada subarreglo de t columnas

satisface los requerimientos para ser un MCA de fuerza t (propiedad de herencia de cobertura), excepto los subarreglos que contienen ambas columnas i y j. Las tuplas que aún no han sido cubiertas se agregan de la siguiente manera (Figura 2.14(C)(E)):

- Inicializar un conjunto de banderas que indiquen que celdas de la columna j puede ser modificada libremente.
- Modificar las celdas libres en la columna j para incrementar la cobertura con respecto a la columna i.
- Si la operación destruye tuplas heredadas, modifica las celdas libres con el fin de restaurar la cobertura original con la columna i.
- Si no hay celdas libres se agregan filas para cubrir las tuplas faltantes.

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

A) B) C) D) E)

Figura 2.14: MCA$(9, 2, 4, 3^3 2^1)$ construido bajo el algoritmo de herencia de cobertura.

2.4 Métodos metaheurísticos

Los algoritmos metaheurísticos son más completos y complejos que los algoritmos voraces, ya que aplican distintas estrategias para diversificar la búsqueda e intensificarla cuando se considere necesario, por lo que generalmente entregan mejores soluciones sacrificando tiempo de cómputo, aunque en la práctica obtienen buenos resultados en tiempos razonables.

2.4.1 Algoritmos genéticos

El concepto de algoritmos genéticos fue inventado por John Holland en el año de 1960 y desarrollada hasta 1970. En 1975, Holland presentó el libro "*Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control and artificial intelligence*"[25] donde mostraba a los algoritmos genéticos como una abstracción de la evolución biológica donde se contaba con una población que era sometida a cruzas y mutaciones genéticas bajo un mecanismo de selección natural.

En 2001, Stardom [49] presentó un algoritmo genético para construir CAs. Iniciando con una población S conformada por un conjunto de matrices con combinaciones faltantes donde el número de estas combinaciones faltantes define la aptitud del individuo. Bajo este concepto, el problema se convierte en un problema de minimización. Para seleccionar a los padres de los nuevos individuos se divide la población S en dos grupos de tamaño $\frac{|S|}{2}$, en cada grupo los individuos se ordenan aleatoriamente y los i-ésimos miembros de cada grupo son combinados con el operador de cruza para producir S nuevos individuos. Los resultados obtenidos a partir del algoritmo genético no resultaron tan competitivos en comparación con el recocido simulado y la búsqueda tabú [51].

En 2004 Shiba presenta su algoritmo genético [48] como una modificación del AETG [9]. Inicialmente se crea una población de m casos de prueba en el que la aptitud de cada caso de prueba se evalúa como el número de combinaciones nuevas que aporta S con respecto a la población. Se implementa una estrategia de elitismo donde se conserva una cantidad n de individuos con la mejor aptitud y se aplica cruza y mutación a los $m-n$ individuos restantes para definir a la nueva población junto a los n individuos de élite.

2.4.2 Búsqueda tabú

La búsqueda tabú fue propuesta en 1986 [16] y formalizada en 1989 [17] [18] como una forma de incorporar una memoria adaptativa mediante una lista que almacena los últimos movimientos del

algoritmo con la finalidad de evitar regresar a zonas ya exploradas y quedar atrapado en óptimos locales.

En 2004, Nurmela [42] propuso un algoritmo para la construcción de CAs mediante búsqueda tabú. Inicialmente se crea una matriz $k \times n$ donde el número de combinaciones faltantes es el costo de la matriz. A continuación se selecciona una combinación faltante y se busca el renglón que requiere el menor cambio para cubrir dicha combinación. Cada uno de estos cambios corresponde a un movimiento y este se realiza siempre y cuando el movimiento a efectuarse no esté almacenado en la lista tabú.

En 2009, Walker y Colbourn [61] propusieron un algoritmo de búsqueda tabú en el que con la ayuda de Familias Hash de Cobertura Perfecta (CPHF). Se mantiene una lista de estados recientes como lista tabú. Después se generan un vecindario de movimientos. De esos movimientos se elige el mejor movimiento con el mayor puntaje que no esté dentro de la lista tabú. La puntuación obtenida de un arreglo candidato S está dada por el número de combinaciones de t columnas que no tienen cobertura por alguna fila, si un conjunto de columnas no está cubierto, cada columna del conjunto es deficiente. Los valores de la puntuación de un candidato S están entre $0 \leq \binom{k}{t}$ donde una puntuación 0 indica un CPHF. Con esta implementación lograron definir nuevas cotas para CAs con fuerzas 3, 4 y 5.

2.4.3 Recocido simulado

El algoritmo de recocido simulado fue propuesto por Scott Kirkpatrick, C. Daniel Gelatt y Mario P. Vecchi en [27]. Este enfoque esta inspirado en el proceso de templado de metales que consiste en calentar, sostener la temperatura alta y posteriormente enfriar lentamente algún metal para variar sus propiedades físicas y reducir sus defectos.

En 2008 Covarrubias-Flores presenta una implementación de un algoritmo de recocido simulado para construir covering arrays binarios de fuerza variable [14]. Iniciando con una matriz M inicializada aleatoriamente en el que el número de ceros y unos se encuentran balanceados. La función de

evaluación se determina por el número de combinaciones que no están cubiertas por la matriz M.

En 2010 Martinez-Pena presenta en [39] un algoritmo de recocido simulado para la construcción de covering arrays de alfabeto $v = 3$ usando coeficientes trinomiales para reducir el espacio de búsqueda del problema. Una solución factible S esta formada por la yuxtaposición de algunos coeficientes trinomiales de grado k.

En 2010 Torres-Jimenez y Rodriguez-Tello presentan en [58] un algoritmo de recocido simulado para construir covering arrays binarios, posteriormente en 2012 presentaron una mejora de este algoritmo en [59]. Para generar una solución inicial se define aleatoriamente cada columna con $\frac{N}{2}$ unos y $\frac{N}{2}$ ceros de tal manera que se encuentren balanceados o que la diferencia sea como máximo 1. Para generar una nueva solución A' a partir de A se utilizan dos funciones de vecindad, $cambio(A, i, j)$ el cual cambia el valor de la celda (i, j) por un valor diferente del alfabeto y la segunda $intercambio(A, i, j, l)$ intercambia el valor de la celdas (i, j) y (l, j), el costo de una solución A es el número de combinaciones faltantes en A.

En 2012 Avila-George et al. presentan en [21] y [1] tres enfoques de un algoritmo de recocido simulado paralelo. Los enfoque propuestos están basados en búsquedas independientes, semi-independientes y cooperativas. En la búsqueda independiente cada procesador ejecuta una versión secuencial del algoritmo de recocido simulado y al final cada procesador entrega su mejor resultado a un procesador maestro el cual selecciona el mejor resultado entregado; La búsqueda semi-independiente divide el esfuerzo de generar una cadena de Markov en p procesadores, de esta manera cada procesador intercambia sus soluciones intermedias y sus costos; La búsqueda cooperativa emplea una comunicación asíncrona en la que los procesadores acceden al estado global y su costo y lo actualizan si es necesario. En este trabajo se remarca que la búsqueda cooperativa ofrece mejores tiempos de ejecución con los mismos resultados obtenidos en comparación con la búsqueda independiente y semi-independiente.

2.5 Resumen

A continuación se presenta una comparativa de los diferentes enfoques para la construcción de covering arrays en el que se destacan tanto ventajas y desventajas que conllevan su implementación. Tomando en cuenta que el algoritmo que se plantea en este trabajo para encontrar el VI es mediante búsqueda se desarrollarán tres algoritmos siguiendo los enfoques: exacto, metaheurístico y voraz.

Clasificación	Ventaja	Desventaja
Algebraicos	Construyen CAs en tiempo polinomial mediante formulas u operaciones definidas.	Se aplican a casos particulares de valores t, k y v.
Exactos	Garantizan encontrar la mejor solución.	El tiempo de búsqueda es alto porque se realiza de manera exhaustiva.
Voraces	Requieren poco tiempo de cómputo y construyen soluciones aceptables.	No garantizan que la solución obtenida es óptima.
Metaheurísticos	Son más complejos que los algoritmos voraces, por lo tanto obtienen mejores soluciones.	No garantizan que la solución obtenida es óptima y el tiempo de cómputo suele ser largo.

Tabla 2.2: Tabla comparativa de enfoques para la construcción de CAs.

En el siguiente capítulo se presentan algunos conceptos empleados en la construcción por vectores inicializadores y se define de manera formal la generalización propuesta en este trabajo de tesis.

3

Generalización propuesta para la construcción de vectores inicializadores

En este capítulo se presenta la definición formal de un vector inicializador y algunos conceptos de interés relacionados a la construcción de vectores inicializadores que serán empleados en este documento. Para finalizar se introduce el concepto de vectores inicializadores generalizados el cual representa la definición formal de la generalización propuesta en esta investigación.

3.1 Vectores Inicializadores

En la literatura se ha propuesto el construir un CA usando el concepto de vector inicializador aplicando operaciones de rotación y traslación [6][35][11][19].

Definición 6: Vector Inicializador (VI)

Un vector inicializador es un vector S de tamaño k tal que $\{\forall j | S_j \in \mathbb{Z}_v \ \wedge \ 0 \leq j < k\}$.

Empleando el concepto de vector inicializador se reduce el espacio de búsqueda ya que el costo de encontrar un vector de k elementos con v valores se encuentra acotado por v^k. Dado un vector inicializador S se pueden producir kv vectores diferentes aplicando operaciones de rotación y traslación. Denotamos como $S^{l,r}$ a un vector obtenido por r rotaciones del l-ésimo vector trasladado. Por la propiedad del elemento neutro podemos concluir que $S = S^{0,0}$.

La operación de `traslación` sobre el vector S se denota como:

$$S_j^{l,0} = (S_j + l) \mod v, \text{ para } 0 \leq l < v \tag{3.1}$$

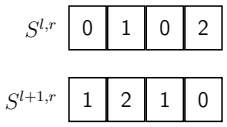


Figura 3.1: Ejemplo de vector $S^{l+1,r}$ de un CA$(t, 4, 3)$ obtenido mediante la operación de traslación.

La operación de `rotación` sobre el vector $S_j^{l,0}$ se denota como:

$$S_j^{l,r} = S_{(j+r) \mod k}^{l,0}, \text{ para } 0 \leq r < k \tag{3.2}$$

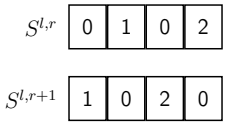


Figura 3.2: Ejemplo de vector $S^{l,r+1}$ de un CA$(t, 4, 3)$ obtenido mediante la operación de rotación.

donde $S_j^{l,r}$ denota el j-ésimo elemento del vector $S^{l,r}$.

3.1.1 Grupos Cíclicos

Dado que los $N-1$ renglones restantes son obtenidos a partir de un solo vector o VI, el proceso de validación para un CA, obtenido mediante este tipo de construcción, puede ser agilizado identificando los diferentes grupos cíclicos presentes en un CA(t, k, v).

El proceso de validación para un CA, obtenido a partir de un vector inicializador que permite construir N renglones, puede ser agilizado identificando los diferentes grupos cíclicos presentes en las t maneras de tomar k columnas.

Definición 7: Grupo cíclico

Sea ($\mathbb{G}$, $*$) un grupo, se dice que $\mathbb{G}$ es cíclico si existe un elemento $\alpha \in \mathbb{G}$ tal que $\alpha^n \in \mathbb{G}$.

Definición 8: Órbita

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

Definición 9: Vector de distancia

Un vector de distancia es un vector X de tamaño t que representa un conjunto de órbitas. Cada elemento X_i representa la distancia entre un par de los elementos de la órbita de un vector de tamaño k. El vector de distancias X puede ser representado mediante la siguiente ecuación diofántica, con la restricción de que todos los elementos deben ser mayores a cero.

$$\sum_{i=0}^{t-1} X_i = k \wedge min(X_0, \ldots, X_{t-1}) > 0 \tag{3.3}$$

Definición 10: Vectores de distancia equivalentes

Se dice que dos vectores de distancia (X y Y) son equivalentes siempre y cuando se cumpla que:

$$X_i = Y_{(i+l) \mod t}, \text{ cuando } 1 \leq l \leq t-1 \tag{3.4}$$

$X^1 = \{1,1,4\}$	$X^4 = \{1,2,3\}$	$X^7 = \{1,3,2\}$	$X^{10} = \{2,2,2\}$
0 1 2	0 1 3	0 1 4	0 2 4
1 2 3	1 2 4	1 2 5	1 3 5
2 3 4	2 3 5	$X^8 = \{2,1,3\}$	
3 4 5	$X^5 = \{3,1,2\}$	0 2 3	
$X^2 = \{4,1,1\}$	0 3 4	1 3 4	
0 4 5	1 4 5	2 4 5	
$X^3 = \{1,4,1\}$	$X^6 = \{2,3,1\}$	$X^9 = \{3,2,1\}$	
0 1 5	0 2 5	0 3 5	

Figura 3.3: Para el caso $t = 3$, $k = 6$, existen $\binom{k}{t} = 20$ distintas maneras de tomar t elementos de k columnas. Dichos elementos se encuentran agrupados en 4 grupos cíclicos representados por los vectores de distancia $X^1 = \{1,1,4\}, X^4 = \{1,2,3\}, X^7 = \{1,3,2\}, X^{10} = \{2,2,2\}$ y sus respectivos vectores equivalentes.

Es importante tener en cuenta el número de grupos cíclicos y el número de vectores de distancia contenidos en un CA(t,k,v), ya que éstos nos ayudan a identificar cómo se distribuye una combinación en particular, que ha sido asignada a un conjunto de t columnas, a través de los $N-1$ renglones generados a partir del VI. Siguiendo el caso del ejemplo que se ilustra en la Figura 3.3, una combinación del tipo $\{A,B,C\}$ (donde $A,B,C \in \mathbb{Z}_v \wedge A \neq B \neq C$) asignada a la órbita {0,1,2} (representada por el vector de distancia X^1 = {1,1,4}) estará presente además en el conjunto de órbitas que pueden ser representadas por el mismo vector de distancia X^1 ({{1,2,3},{2,3,4},{3,4,5}}). Cada vector de equivalencia dentro de un grupo cíclico denota una rotación en el que un elemento del VI es desplazado hacia el otro extremo del vector, esto significa que para el conjunto de órbitas representadas por el vector de distancia equivalente a X^1, el vector de distancia X^2, comenzará a cubrir combinaciones del tipo $\{C,A,B\}$ dentro de su conjunto de

órbitas ({{0,4,5}}), de la misma manera, las órbitas representadas por el vector de distancia X^3 comenzará a cubrir combinaciones del tipo $\{B, C, A\}$ dentro de su conjunto de órbitas ({{0,1,5}}) tal y como se muestra en la Figura 3.4.

$$\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} A & B & C & - & - & - \\ - & A & B & C & - & - \\ - & - & A & B & C & - \\ - & - & - & A & B & C \\ C & - & - & - & A & B \\ B & C & - & - & - & A \end{pmatrix} \end{array}$$

Figura 3.4: Distribución de combinaciones en órbitas dentro de un grupo cíclico.

3.1.2 Identificación de combinaciones redundates

Tal y como se mostró en la subsección anterior, seleccionar una combinación para una órbita en específico puede proporcionar cobertura para otras órbitas que pertenecen al mismo grupo cíclico. Dado que únicamente se tiene un VI, lo deseable sería agrupar aquellas combinaciones que aporten elementos nuevos a la cobertura total. Con la finalidad de evitar seleccionar elementos redundantes, es posible asignar un identificador (o color) a cada elemento de la matriz de cobertura completa, de tal manera que se asegure que un conjunto de combinaciones de un mismo color pueden ser cubiertos siempre y cuando se incluya un elemento representativo de este color en el VI.

Definición 11: Matriz de cobertura completa

Una matriz de cobertura completa es un arreglo P de dimensiones $v^t \times \binom{k}{t}$ que indica todos los elementos que deben ser cubiertos por un $CA(N; t, k, v)$, donde cada columna representa un conjunto de t columnas distinto y cada uno de los elementos presentes por columna representa una combinación de las v^t posibles.

En la Figura 3.5 se puede visualizar el proceso de coloreo de la matriz de cobertura completa para un $CA(8; 3, 4, 2)$ aplicando k rotaciones y v traslaciones. Inicialmente se recorre la matriz en

busca de un elemento que no haya sido coloreado hasta el momento, una vez identificado se asigna la combinación a su correspondiente conjunto de t columnas en el VI y se simula el procedimiento de construcción identificando aquellas combinaciones que son cubiertas bajo este procedimiento, a dicho conjunto de combinaciones se les asigna el siguiente color disponible. Este procedimiento se repite hasta que todos los elementos de la matriz de cobertura completa tengan un color asignado.

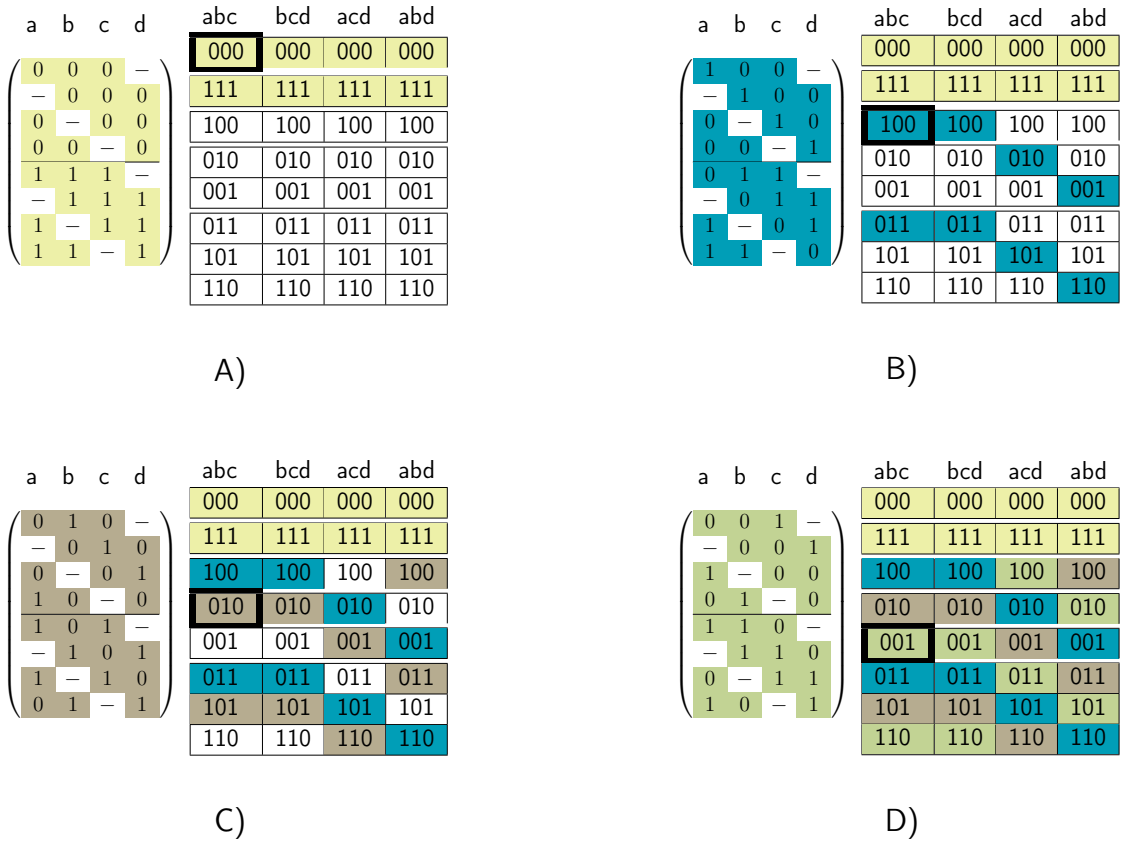


Figura 3.5: Proceso de coloreo de la matriz de cobertura completa para un $CA(8;3,4,2)$, usando un vector inicializador y aplicando k rotaciones y v traslaciones. En la parte superior de cada matriz se encuentra denotado mediante las letras {a, b, c, d} el conjunto de k columnas tomados en conjuntos de t elementos.

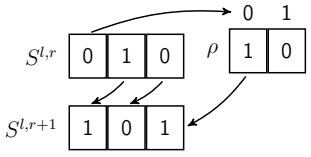
3.2 Vectores inicializadores generalizados

A continuación se describe de manera formal la generalización propuesta para la construcción de vectores inicializadores, la cual definiremos con el nombre de vectores inicializadores generalizados (VIG).

Definición 12: Vector Inicializador Generalizado (VIG)

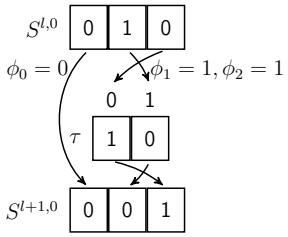
Un vector inicializador generalizado denotado como $VIG(\rho, \tau, \phi, \pi, \sigma)$, es un objeto empleado como auxiliar en la construcción de covering arrays mediante un vector inicializador. Un VIG se encuentra compuesto por cinco vectores que codifican las operaciones que deberán ser aplicadas al vector inicializador. La descripción de dichos vectores se presenta a continuación:

- ρ: vector de tamaño v que indica el valor al que debe ser cambiado el elemento que se desplaza al otro extremo al efectuarse una rotación.

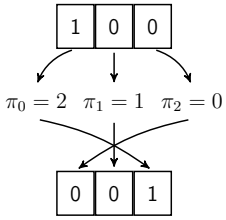


- τ: vector de tamaño v que indica el valor al que debe ser cambiado cada elemento del VIG para producir el vector l-ésimo.

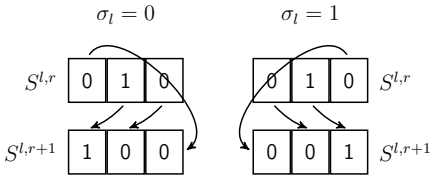
- ϕ: vector de tamaño k que indica los elementos $S_j^{l,0}$ que no son afectados por la traslación.



- π: vector de tamaño k que define una permutación de las columnas para ser las columnas del vector trasladado.



- σ: vector de tamaño δ que indica el sentido de las rotaciones por cada VIG trasladado, donde δ es el número de traslaciones.



De los cuales ρ, τ permiten controlar el número de filas resultantes y ϕ, π, σ se enfocan en cubrir las v^t tuplas del CA a través de la modificación de los valores de $S^{l,r}$.

3.2.1 Generalización I: Retroalimentación

El operador de rotación, de manera convencional, produce una matriz $k \times k$ tal que S sea un VI y para $1 \leq r < k$, $0 \leq j < k$:

$$S_j^{l,r} = S_{(j+r) \mod k}^{l,0}$$

En este contexto se propone el uso de un vector ρ de tamaño v que contendrá una permutación de los elementos $\{0, \ldots, v-1\}$. El vector ρ permite controlar el número de filas a través de θk rotaciones, donde:

$$\theta = \begin{cases} 1 & \textbf{si } \sum_{i=0}^{v-1}(\rho_i = i) = v \\ v - \sum_{i=0}^{v-1}(\rho_i = i) & \textbf{de otra manera} \end{cases}$$

A partir de esta construcción se obtiene una matriz $(\theta k) \times k$ tal que S sea un VI y para $1 \leq r < \theta k$, $0 \leq j < k$:

$$S_j^{l,r} = \begin{cases} \rho_{S_0^{l,r-1}} & \textbf{si } j = k-1 \\ S_{j+1}^{l,r-1} & \textbf{de otra manera} \end{cases}$$

De esta manera cada elemento del vector ρ que cumple la condición $\{\exists i |\ \rho_i \neq i\ \wedge\ 0 \leq i < v\}$ define un nuevo conjunto de k rotaciones diferentes. Cuando este vector cumple $\{\forall i |\ \rho_i = i\ \wedge\ 0 \leq i < v\}$ se tendrá un total de k rotaciones con la que se obtiene una matriz $k \times k$ equivalente a la construcción convencional. Para el caso contrario en el que $\{\forall i |\ \rho_i \neq i\ \wedge\ 0 \leq i < v\}$ es posible producir un total de kv rotaciones diferentes extendiendo la construcción a una matriz de tamaño $(kv) \times k$.

3.2.2 Generalización II: Traslación

En cuanto a la operación de traslación, actualmente el número de vectores trasladados y sujetos a la realización de rotaciones es $v - F$ donde F define los elementos que no son afectados por la operación de traslación. Sea S un VI y para $0 \leq l < v - F$.

$$S_j^{l,0} = (S_j + l) \mod v + F$$

En este contexto se propone el uso de un vector τ de tamaño v que contendrá una permutación de los elementos $\{0, \ldots, v-1\}$ que indica el valor al que debe ser cambiado cada elemento del VIG para producir el vector i-ésimo.

Al asignar una permutación al vector τ se puede definir una partición del alfabeto en λ grupos disjuntos $G_1, \ldots, G_\lambda$ tal que $1 \leq \lambda \leq v$, $min(|G_1|, \ldots, |G_\lambda|) \geq 1$, $\sum_{i=1}^{\lambda} |G_i| = v$ y $\bigcap_{i=1}^{\lambda} G_i = \{\emptyset\}$. Cuando este vector cumple $\{\exists i | \tau_i = i \ \wedge \ 0 \leq i < v\}$ se obtiene una construcción equivalente a usar símbolos fijos y debido a que esta construcción implica concatenar un CA$(t, k, \sum_{i=0}^{v-1}(\tau_i = i))$, la experimentación sólo tiene sentido para $\sum_{i=0}^{v-1}(\tau_i = i) \leq v/2$, de lo contrario el resultado estará compuesto en mayor porcentaje por el CA ingrediente.

Definimos δ como el número de traslaciones necesarias considerando los siguientes casos:

$$\delta = \begin{cases} 1 & \textbf{si } \sum_{i=0}^{v-1}(\tau_i = i) = v \\ max(|G_1|, \ldots, |G_\lambda|) & \textbf{de otra manera} \end{cases}$$

Sea S un VI y para $1 \leq l < \delta$, $0 \leq j < k$.

$$S_j^{l,0} = \tau_{S_j^{l-1,0}}$$

Con el propósito de definir los elementos en las posiciones j que no son afectados por la operación de traslación, se propone el uso de un vector ϕ de tamaño k y $\phi_j \in \mathbb{B}$ para $0 \leq j < k$, donde:

$$S_j^{l,0} = \begin{cases} \tau_{S_j^{l-1,0}} & \textbf{si } \phi_j = 1 \\ S_j^{l-1,0} & \textbf{si } \phi_j = 0 \end{cases}$$

El vector τ permite controlar el número de filas en conjunción con el vector ρ produciendo una matriz $(\delta \ \theta \ k) \times k$.

3.2.3 Generalización III: Orden

En el caso concreto de los vectores trasladados se cumple que las columnas son trasladadas en el mismo orden que el vector anterior:

$$S_j^{l,0} = \begin{cases} \tau_{S_j^{l-1,0}} & \textbf{si } \phi_j = 1 \\ S_j^{l-1,0} & \textbf{si } \phi_j = 0 \end{cases}$$

Se propone el poder definir una permutación, controlada a través de un vector π_l de tamaño k para $0 \leq l < \delta$, que establecerá el orden en que son trasladadas las columnas para obtener el vector $S^{l+1,0}$. Asociado a cada vector trasladado se tendrá un vector π_l tal que:

$$S_{\pi_j}^{l,0} = \begin{cases} \tau_{S_j^{l-1,0}} & \textbf{si } \phi_j = 1 \\ S_j^{l-1,0} & \textbf{si } \phi_j = 0 \end{cases}$$

Se plantea definir un elemento $\sigma_l \in \mathbb{B}$ para $0 \leq i < \delta$ que indica el sentido de las rotaciones:

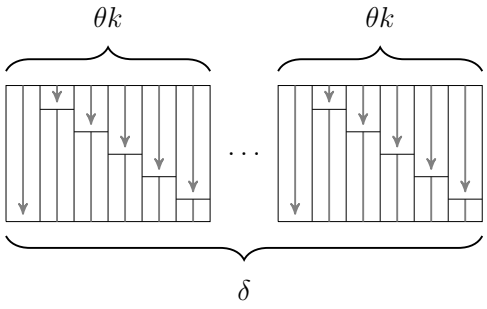
$$S_j^{l,r} = \begin{cases} \begin{cases} \rho_{S_0^{l,r-1}} & \textbf{si } j = k-1 \\ S_{j+1}^{l,r-1} & \textbf{de otra manera} \end{cases} & \textbf{si } \sigma = 0 \\ \begin{cases} \rho_{S_{k-1}^{l,r-1}} & \textbf{si } j = 0 \\ S_{j-1}^{l,r-1} & \textbf{de otra manera} \end{cases} & \textbf{si } \sigma = 1 \end{cases}$$

3.2.4 Modularización del parámetro N

A través de la construcción por VIG es posible modular el tamaño del número de filas de un CA, dado los valores t, k y v. El número de filas N esta definido por la ecuación:

$$N = \delta \ \theta \ k$$

En la Figura 3.6 se presenta la relación entre el tamaño del número de filas del CA resultado de la construcción por VIG y los vectores ρ y τ.



$$\theta = \begin{cases} 1 & \textbf{si } \sum_{i=0}^{v-1}(\rho_i = i) = v \\ v - \sum_{i=0}^{k-1}(\rho_i = i) & \textbf{de otra manera} \end{cases}$$

$$\delta = \begin{cases} 1 & \textbf{si } \sum_{i=0}^{v-1}(\tau_i = i) = v \\ max(|G_1|, \ldots, |G_\lambda|) & \textbf{de otra manera} \end{cases}$$

Figura 3.6: El CA resultante se encuentra compuesto de δ bloques donde la primera fila es un VIG trasladado seguido de sus θk rotaciones. En la parte derecha de la figura se definen los valores para θ y δ los cuales pueden ser controlados mediante los vectores ρ y τ respectivamente.

3.3 Resumen

En este capítulo se presentó de manera formal el concepto vectores inicializadores además de los conceptos de grupos cíclicos, órbitas y vectores de distancia que facilitan el proceso de validación de un CA que ha sido construido empleando VIs. Dentro de este capítulo se incluye también un procedimiento que ayuda a identificar combinaciones redundantes que se desea descartar dentro del proceso de búsqueda del VI. Para finalizar se presentó de manera formal la generalización propuesta en este trabajo de tesis, mediante la cual es posible reproducir las operaciones propuestas en el estado del arte, además de introducir nuevos operadores como la partición del alfabeto y la retro-alimentación de símbolos al efectuar el operador de rotación. En el siguiente capítulo se presenta el cambio de la representación del problema de búsqueda de VIs al dominio de grafos, el cual representa la unificación del método de obtención del VI para cualquier VIG dado.

4

Búsqueda de vectores inicializadores en el dominio de grafos

En este capítulo se presenta un enfoque original en el que se propone trasladar el problema de la búsqueda de vectores inicializadores al dominio de grafos. Este cambio de enfoque permite aprovechar el conocimiento que proporcionan los conceptos de grupos cíclicos y vectores de distancia y que además es una generalización de las construcciones propuestas, debido a que es un método de búsqueda con enfoque exacto que emplea una heurística para orientar la búsqueda a regiones prometedoras, y no está restringido a ciertos parámetros de t, k y v como es el caso de las construcciones algebraicas. Además se presenta un análisis del número de nodos, aristas y el máximo clique que es posible obtener empleando esta representación.

4.1 Definición de la estructura del grafo

En [44] se presenta un algoritmo de postoptimización en el que se muestra un mapeo del problema de construcción de covering arrays al dominio de grafos. Bajo este enfoque se modelan cada una de las v^t combinaciones que se requieren para cualquier t distintas columnas de k parámetros como nodos de un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, de esta manera cada elemento presente dentro de una matriz de cobertura completa es representado como un nodo que pertenece al conjunto de vértices $\mathcal{V}$ del grafo $\mathcal{G}$.

Dado que cada nodo representa un elemento dentro de la matriz de cobertura completa, un nodo x, tal que $x \in \mathcal{V}$, codifica una combinación particular que será asignada a un conjunto de t columnas (representados por los vectores α_i^x y β_i^x para $0 \leq i < t$ respectivamente). El conjunto de arcos que conforma el grafo se encuentra definido entre aquellos pares de nodos que sean compatibles entre sí ya que sólo se tiene un vector inicializador en el que cada columna puede aceptar un solo valor dentro de los $\mathbb{Z}_v$ valores posibles.

Definición 13: Nodos compatibles

Se dice que un par de nodos $x, y \in \mathcal{V}$ son compatibles si se cumple la siguiente función:

$$\Lambda(x,y) = \begin{cases} 1 & \textbf{si } \forall i,j \mid \beta_i^x = \beta_j^y \rightarrow \alpha_i^x = \alpha_j^y \vee \forall i,j | (\beta_i^x \neq \beta_j^y) \text{ para } 0 \leq i,j < t \\ 0 & \textbf{de lo contrario} \end{cases} \tag{4.1}$$

La compatibilidad entre un par de nodos $x, y \in \mathcal{V}$ definida mediante la función $\Lambda(x, y)$ en la Ecuación 4.1, asegura que no exista una sobreescritura en la asignación de un valor para cierta columna, por lo que únicamente se establece un arco en el caso de que ambos nodos no compartan ninguna columna $\forall i, j | (\beta_i^x \neq \beta_j^y)$ o en caso contrario, cuando se cumpla que $\beta_i^x = \beta_j^y$ se asegure que dicha columna tenga el mismo valor asignado $\alpha_i^x = \alpha_j^y$. Una vez modelado el conjunto de enlaces

con los pares de nodos que son compatibles entre sí, es posible comenzar a realizar adecuaciones basadas en el problema de búsqueda de vectores inicializadores descartando los enlaces entre aquellas combinaciones que resultan redundantes.

Definición 14: Nodos redundantes

Se dice que un par de nodos $x, y \in \mathcal{V}$ son redundantes si el nodo y puede ser producido por el nodo x aplicando las operaciones descritas en el $VIG(\rho, \tau, \pi, \phi, \sigma)$.

En el Algoritmo 1 se presenta el pseudocódigo para el coloreo de la matriz de cobertura completa empleado en la detección de combinaciones redundantes (descrito en el Capítulo 3) adaptado al dominio de grafos. Dicho algoritmo recibe como entrada tres enteros no negativos t, k y v que conforman los parámetros del $CA(t, k, v)$ y un $VIG(\rho, \tau, \pi, \phi, \sigma)$ en el que se encuentran codificadas las operaciones que le serán aplicadas al vector inicializador para la construcción de un covering array. Inicialmente se define el valor $\zeta(x) = -1$ para cada nodo $x \in \mathcal{V}$ del grafo (Línea 4) en el que se indica que no ha sido asignado un color al nodo x. Posteriormente se procede a recorrer cada nodo x del conjunto $\mathcal{V}$ para detectar nodos a los cuales no se le ha asignado un color $\zeta(x) = -1$ (Línea 7). Una vez detectado, se indica que se requiere un nuevo color para cubrir un nuevo subconjunto de nodos $\mathcal{V}' \subseteq \mathcal{V}$ definido por la función $\Gamma(x, VIG)$ (Línea 9) la cual recibe como entrada un nodo x y un VIG y retorna como salida el subconjunto de nodos que pueden ser producidos por el nodo x aplicando las operaciones descritas en el VIG, a continuación dicho color es asignado a todos los nodos y tal que $y \in \mathcal{V}'$ y se vuelve a iniciar el procedimiento de búsqueda de aquellos nodos que cumplan con $\zeta(x) = -1$. Como salida, se obtiene el conjunto de nodos $\mathcal{V}$ tal que cada nodo $x \in \mathcal{V}$ tiene asignado un color denotado por $\zeta(x)$.

Algoritmo 1: Pseudocódigo de coloreo para detección de nodos redundantes.

```
1  función coloreo (t, k, v, VIG);
   Entrada: tres enteros no negativos t, k, y v y un VIG(ρ, τ, π, φ, σ).
   Salida  : conjunto de vertices V con un color ζ asignado.
2  V ← {0, ..., (k choose t) v^t − 1};
3  color ← 0;
4  para x ∈ V hacer
5      ζ(x) ← −1
6  para x ∈ V hacer
7      si ζ(x) ≠ −1 entonces
8          color ← color + 1;
9          V' ← Γ(x, VIG) ;        // conjunto de nodos que son producidos por el nodo x de
           acuerdo al VIG
10         para y ∈ V' hacer
11             ζ(y) ← color;
```

Ejemplo del uso de k rotaciones y v traslaciones para construir un CA$(8; 3, 4, 2)$

A continuación se presenta un ejemplo en el que se muestra como es posible obtener un vector inicializador trasladando el problema de búsqueda al dominio de grafos. Para este caso el covering array a construir es un CA$(8; 3, 4, 2)$ mediante un vector inicializador al que se le aplicarán operaciones de rotación y traslación bajo los siguientes parámetros del VIG$(\rho, \tau, \pi, \phi, \sigma)$:

$$\rho = \{0, 1, 2\};\ \tau = \{1, 2, 0\}$$
$$\pi = \{0, 1, 2\};\ \phi = \{1, 1\};\ \sigma = \{1, 1\}$$

Para la construcción del grafo inicialmente se requiere definir los $\binom{k}{t}v^t$ nodos que representa la cobertura total que debe proporcionar un CA$(N; t, k, v)$ por definición. Con la finalidad de identificar cada nodo del grafo un nodo $x \in \mathcal{V}$, para un caso de fuerza t, se representará mediante la siguiente nomenclatura: $\beta_0^x\alpha_0^x \ldots \beta_{t-1}^x\alpha_{t-1}^x$ (los elementos presentes en β serán denotados mediante letras del alfabeto). Posteriormente se establece un arco para cada par de nodos $x, y \in \mathcal{V}$ que satisfacen la

función $\Lambda(x, y) = 1$. Como resultado se obtiene un grafo en el que existen v^k máximos cliques de tamaño $\binom{k}{t}$ que representan todos los posibles vectores inicializadores que es posible obtener pero no todos estos vectores proveen cobertura completa empleándolo como vector inicializador y aplicando las operaciones correspondientes. Esta primera versión del grafo puede ser visualizado en la Figura 4.1.

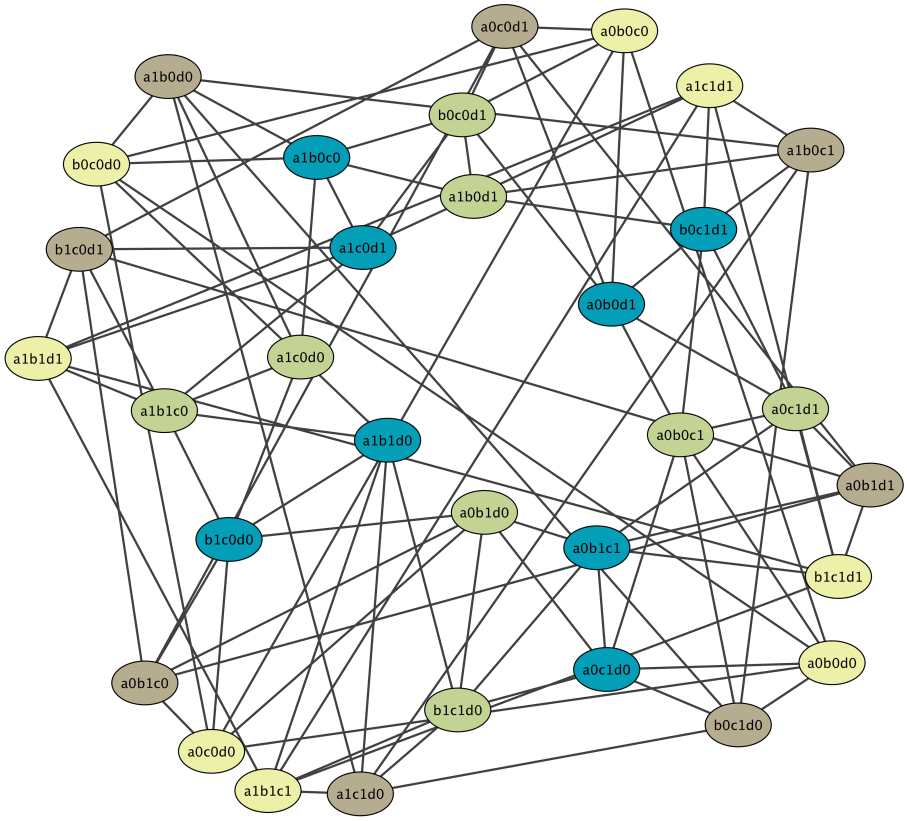


Figura 4.1: Modelado del problema de búsqueda de VIs en el dominio de grafos (1° criterio de compatibilidad: $\Lambda(x, y) = 1 \ \forall x, y | x, y \in \mathcal{V}$).

Debido a la propiedad de los vectores inicializadores de distribuir combinaciones a través de cada elemento contenido dentro de su mismo grupo cíclico, es deseable eliminar la redundancia removiendo los enlaces entre aquellos pares de nodos $x, y \in V$ tal que $\zeta(x) = \zeta(y)$. Una vez que hemos descartado los enlaces que contienen nodos redundantes, obtenemos un grafo de menor densidad (Figura 4.2) en el que el máximo clique (Figura 4.3) puede ser interpretado como un vector inicializador en el que

se garantiza que aporta la máxima cobertura al ser operado mediante un VIG.

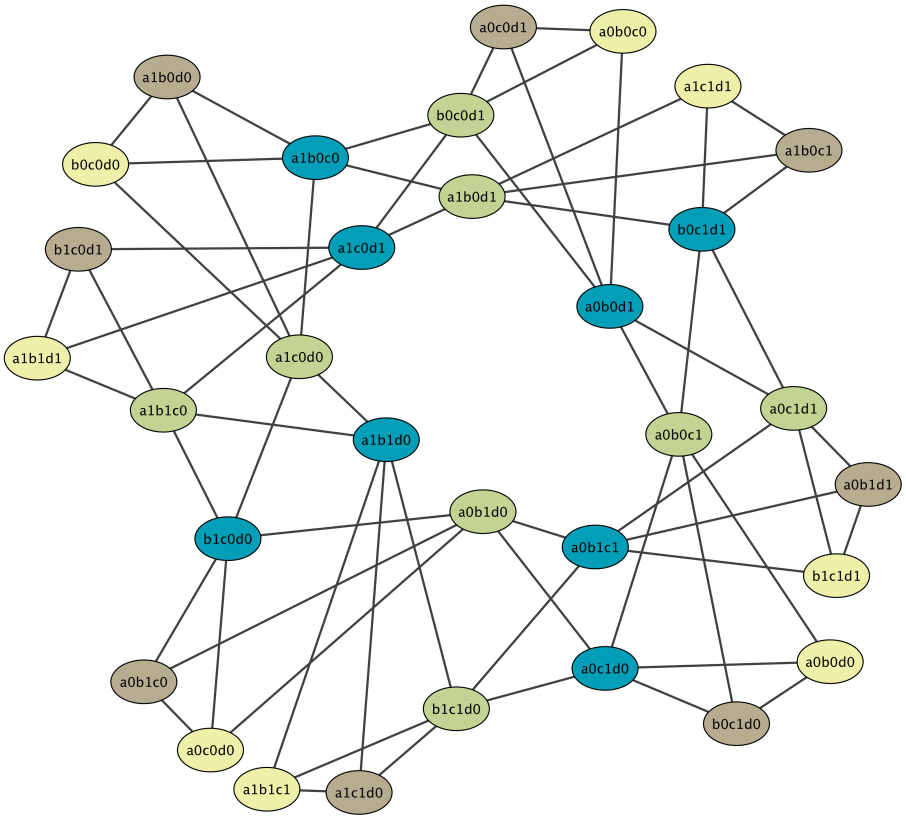


Figura 4.2: VIs en el dominio de grafos (2° criterio de compatibilidad: remover enlaces donde $\zeta(x) = \zeta(y)\ \forall x, y | x, y \in \mathcal{V}$).

Para este caso el máximo clique obtenido es de 4. Empleando esta representación es posible obtener 8 máximos cliques (Figura 4.3) que corresponden a soluciones isomorfas ya que cada máximo clique es capaz de formar un renglón distinto del covering array resultante (Figura 4.4), y debido a que la construcción realiza k rotaciones y v traslaciones para generar los renglones faltantes, resulta indistinto cual de estos se establezca como vector inicializador. El proceso para traducir el máximo clique encontrado en el dominio de grafos a vector inicializador únicamente consiste en asignar $S_{\beta_j^x} = \alpha_j^x$ para cada nodo x tal que x pertenezca al máximo clique.

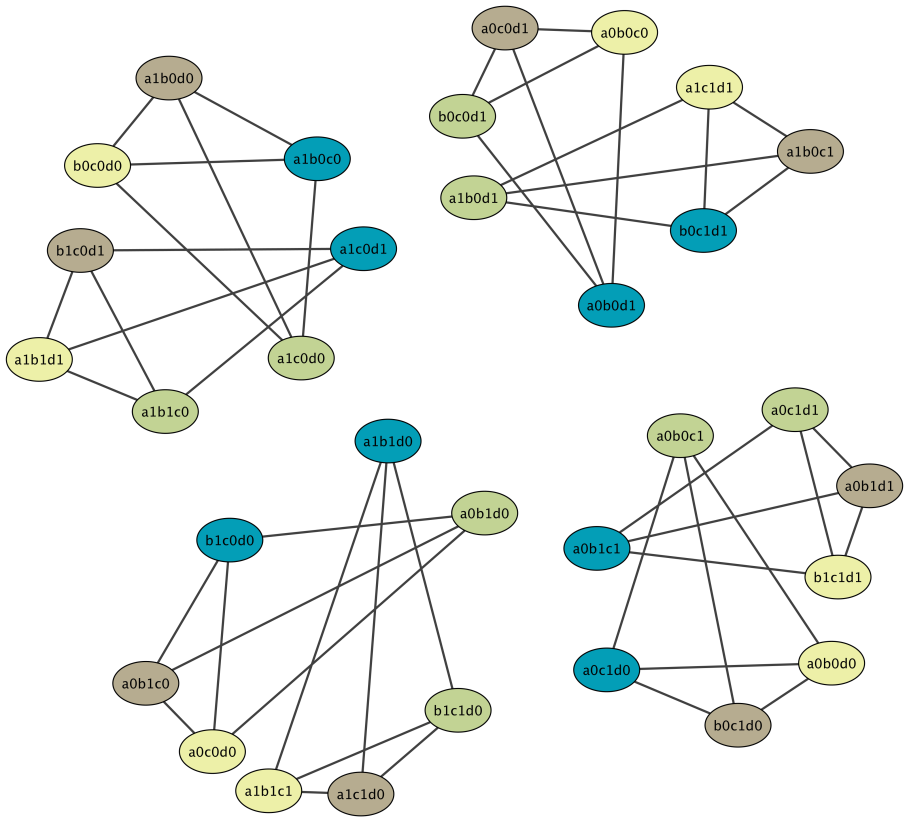


Figura 4.3: Soluciones encontradas por el algoritmo de máximo clique en el que cada solución equivale a un VI.

a b c d

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

a1b0c0	a1b0d0	a1c0d0	b0c0d0
a0b1c0	a0b1d0	a0c0d0	b1c0d0
a0b0c1	a0b0d0	a0c1d0	b0c1d0
a0b0c0	a0b0d1	a0c0d1	b0c0d1
a0b1c1	a0b1d1	a0c1d1	b1c1d11
a1b0c1	a1b0d1	a1c1d1	b0c1d1
a1b1c0	a1b1d1	a1c0d1	b1c0d1
a1b1c1	a1b1d0	a1c1d0	b1c1d0

Figura 4.4: Traducción de soluciones encontradas como máximos cliques en la Figura 4.3 a un CA(8;3,4,2).

4.2 Algoritmo de búsqueda de máximo clique para vectores inicializadores

A continuación se presenta un algoritmo recursivo con enfoque exacto (Algoritmo 2) para la búsqueda del máximo clique en un grafo que ha sido construido pensando en el modelo propuesto para este trabajo de investigación. Aunque se trata de un algoritmo exacto, contiene una heurística que permite evaluar primero aquellos elementos que tienen más potencial de pertenecer al máximo clique, por tal motivo desde el momento que se alcanza el máximo nivel de recursión, este algoritmo entrega soluciones competitivas (es decir que proveen un alto nivel de cobertura para el CA resultante) hasta alcanzar la solución tal que la cardinalidad del conjunto del máximo clique sea igual al número de colores encontrados después de aplicar el algoritmo de coloreo a la matriz de cobertura completa.

Para analizar el funcionamiento de este algoritmo propuesto se desarrollará la búsqueda del máximo clique para el siguiente VIG:

$$\rho = \{0, 2, 1\};\ \tau = \{0, 1, 2\}$$
$$\pi = \{0, 1, 2\};\ \phi = \{1, 1\};\ \sigma = \{1, 1\}$$

Inicialmente se definen tres estructuras de datos las cuales se describen brevemente:

- Definimos el conjunto Δ para denotar las todas las t maneras de tomar k elementos, para este caso dicho conjunto estará conformado de la siguiente manera:

$$\Delta = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}\}$$

- Posteriormente se define el orden en el que serán explorados cada uno de los k elementos del VI que se pretende obtener. Definimos una búsqueda que se encargue de explorar en cada nivel de recursión t nuevos elementos en orden ascendente. Los t elementos que serán explorados

en el nivel de recursión i se encuentran descritos dentro del conjunto L_i para $0 \leq i < \lceil k/t \rceil$:

$$L_0 = \{a, b\}$$

$$L_1 = \{c, d\}$$

- Dado que hasta el nivel de recursión i solo se han explorado los elementos contenidos en $\bigcup_{j=0}^{i} L_j$ para $0 \leq i < \lceil k/t \rceil$, es conveniente definir un conjunto B que almacene todos las t maneras de tomar k elementos contenidas en Δ que cumplan con la restricción $\bigcup_{\forall b | \Delta_b \subseteq \bigcup_{j=0}^{i} L_j} \Delta_b$:

$$B_0 = \{\{a, b\}\}$$

$$B_1 = \{\{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}\}$$

  La cardinalidad $|B_i|$ definirá el número máximo de nodos que es posible añadir al clique dentro del nivel de recursión i.

Para este caso, el grafo obtenido modelando la función de compatibilidad y restricciones de color descritas anteriormente puede ser visualizado en la Figura 4.1. Inicialmente se definen dos conjuntos: un clique actual AC que almacenará el conjunto de nodos que conforman el clique encontrado hasta el momento en el nivel de recursión i y un máximo clique MC en el que se almacenará el conjunto de nodos que definen el máximo clique encontrado hasta ese punto de ejecución del programa. Posteriormente se realiza el llamado a la función recursiva $expandir(0, AC)$ indicándole que el nivel de recursión actual (0) (Línea 14). Dentro de este nivel se define un subconjunto $\mathcal{V}' \subseteq \mathcal{V}$ compuesto por el conjunto de nodos que son compatibles con los nodos que se encuentran en el clique actual $\bigcup_{x \in \mathcal{V} \wedge \forall y | y \in AC \wedge \Lambda(y,x)} x$ (Línea 22), además se definirá el conjunto de candidatos I como los nodos que cumplen con la restricción $x \in \mathcal{V}' \wedge \beta^x \subseteq L_{nivel}$ (Línea 24). Cada uno de estos candidatos $x \in I$ es agregado al clique actual para determinar su impacto en la expansión del clique, por lo

que primero se deberá asegurar que $\nexists y | y \in AC \wedge \zeta(y) = \zeta(x)$ (Línea 25), en caso contrario se procede a remover dicho nodo (Línea 26) para asegurar que todos los nodos en AC se encuentren conectados entre sí. Una vez que ya se ha agregado el nuevo candidato x al clique actual, se deberá identificar aquellos nodos $y \in \mathcal{V} \wedge \exists b | b \in B_{nivel} \wedge \beta^y \subseteq b$ que ademas son compatibles con todos los nodos presentes en el clique actual $\forall x | x \in AC \wedge \Lambda(y, x) \wedge \zeta(x) \neq \zeta(y)$ (Línea 27). El grado de expansión del clique actual agregando un candidato x se denotará como $\Xi(x)$ (Línea 29). Una vez que todos los candidatos han sido evaluados, se procederá a realizar un ordenamiento del conjunto de candidatos de manera descendente tomando en cuenta el grado de expansión $\Xi(x)$ de tal manera que los candidatos con un grado de expansión mayor son explorados primero (Línea 31). Cuando se ha determinado el orden del conjunto de candidatos, cada candidato $x \in I$ será añadido al clique actual asegurando que $\nexists y | y \in AC \wedge \zeta(y) = \zeta(x)$ (Línea 33) y se realizará nuevamente una llamada recursiva $expandir(nivel + 1, AC)$ (Línea 37) siempre y cuando se cumpla que el tamaño del clique actual más una variable ε es mayor que el tamaño del máximo clique (Línea 36). Definimos ε como el mínimo valor entre el numero de t maneras de tomar k elementos que no han sido explorados aún y el numero de colores que siguen presentes en los nodos que poseen un enlace con cada uno de los nodos del clique actual.

$$\varepsilon = min(\sum_{i=level}^{|L|} |B_i|, |\bigcup_{x \in \mathcal{V} \wedge \{\forall y | y \in AC \wedge \zeta(y) \neq \zeta(x)\}} \zeta(x)|) \tag{4.2}$$

Al finalizar el conjunto que define el clique actual es restaurado para iniciar una nueva expansión con otro candidato. Cuando el nivel de recursión i haya alcanzado el límite $i > |L|$ (Línea 17) se deberá comparar la cardinalidad del clique actual es mayor que la cardinalidad del maximo clique, de ser así se define el clique actual como el máximo clique encontrado hasta el momento.

Algoritmo 2: Pseudocódigo de búsqueda de máximo clique para un grafo modelado mediante el problema de construcción por vectores inicializadores.

```
1  función busquedaClique (t, k, v, VIG);
   Entrada: tres enteros no negativos t, k, y v y un grafo G = (V, E).
   Salida  : maximo clique MC.
2  Δ ;                    // conjunto que contiene todas las t maneras de tomar k elementos
3  L ;     // conjunto que contiene en cada posición t nuevos elementos a explorar por cada
   nivel de recursión
4  B ;          // conjunto que contiene en cada posición i todas las t maneras de tomar los
   elementos contenidos en ⋃_{j=0}^{i} L_j para 0 ≤ i < ⌈k/t⌉
5  i ← 0;
6  mientras i < ⌈k/t⌉ hacer
7      si i(t + 1) ≤ k entonces
8          L_i ← {i · t, ..., i · t + (t − 1)};
9      de lo contrario
10         L_i ← {k − t, ..., k − 1};
11     B_i ← ⋃_{∀b|Δ_b⊆⋃_{j=0}^{i} L_j} Δ_b;
12     Δ ← Δ\B_i;
13 MC ← AC ← ∅ ;                    // (MC) Máximo Clique (AC) Clique Actual
14 expandir(0, AC);
15 devolver MC;
```

16 función expandir $(nivel, AC)$;
Entrada: un entero no negativo $nivel$ y el clique actual AC

17 **si** $nivel > |L|$ **entonces**
18 **si** $|AC| > |MC|$ **entonces**
19 $MC \leftarrow AC$;
20 **de lo contrario**
21 $AAC \leftarrow AC$; // Almacenar AC en variable auxiliar AAC
22 $\mathcal{V}' \leftarrow \bigcup_{x \in \mathcal{V} \wedge \forall y | \Lambda(y,x)} x$;
23 **para** $x \in \mathcal{V}' \wedge \beta^x \subseteq L_{nivel}$ **hacer**
24 $I_i \leftarrow x$; // Nuevo candidato
25 **si** $\exists y | y \in AC \wedge \zeta(y) = \zeta(x)$ **entonces**
26 $AC \leftarrow AC \backslash y$;
27 **para** $y \in \mathcal{V}' \wedge \exists b | b \in B_{nivel} \wedge \beta^y \subseteq b \wedge \{\forall x | x \in AC \wedge \Lambda(y,x) \wedge \zeta(x) \neq \zeta(y)\}$ **hacer**
28 $AC \leftarrow AC \cup y$;
29 $\Xi(x) \leftarrow |AC|$; // Evaluar grado de expansión del candidato x
30 $AC \leftarrow AAC$;
31 $ordenar(I, \Xi)$; // Los mejores candidatos se evaluarán primero
32 **para** $x \in I$ **hacer**
33 **si** $\exists y | y \in AC \wedge \zeta(y) = \zeta(x)$ **entonces**
34 $AC \leftarrow AC \backslash y$;
35 $AC \leftarrow AC \cup x$;
36 **si** $|AC| + \varepsilon > |MC|$ **entonces**
37 $expandir(nivel + 1, AC)$;
38 $AC \leftarrow AAC$;

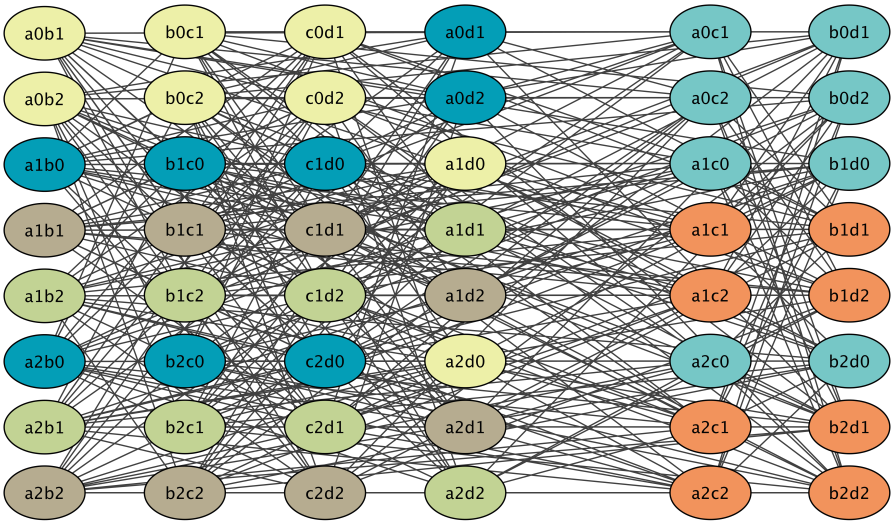


Figura 4.5: Modelado del problema de búsqueda de VIs en el dominio de grafos.

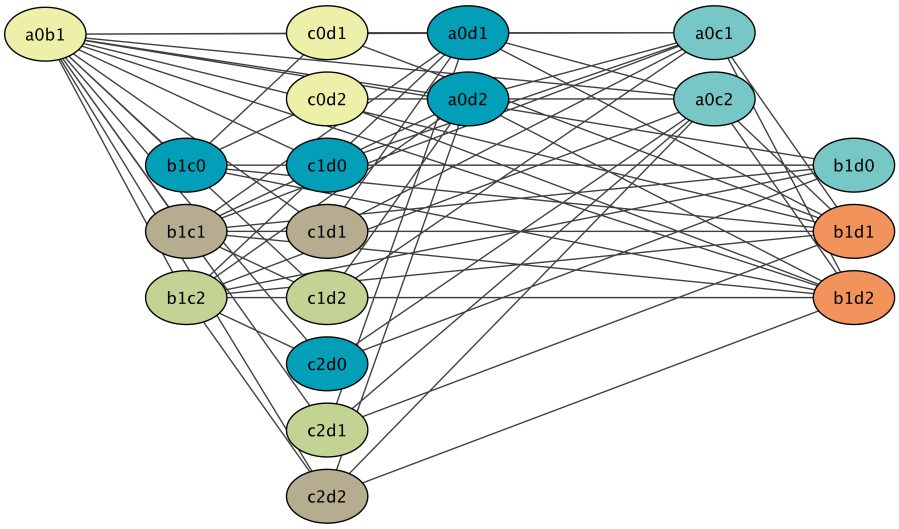


Figura 4.6: Grafo resultante para el nivel de recursión 1 después de seleccionar el nodo a0b1 en el nivel de recursión previo.

Retomando el ejemplo que se pretende analizar el máximo nivel de expansión para el nivel de recursión 0 es de $|L_0| = 1$ por lo que resulta indistinto el orden en el cual se comience a realizar la expansión en el siguiente nivel, por cuestión de orden, se incluye el nodo a0b1 dentro del clique actual y se procede al siguiente nivel de recursión 1. Para este nivel se redefine el conjunto $\mathcal{V}$

eliminando aquellos nodos que no son compatibles con los nodos presentes en el clique actual $\bigcup_{x\in\mathcal{V}\wedge\forall y|y\in AC\wedge\Lambda(y,x)} x$. El grafo resultante se presenta a continuación.

Las siguientes imágenes representan los nodos que conforman el clique actual al evaluar los distintos candidatos que se tienen en el nivel de recursión 1.

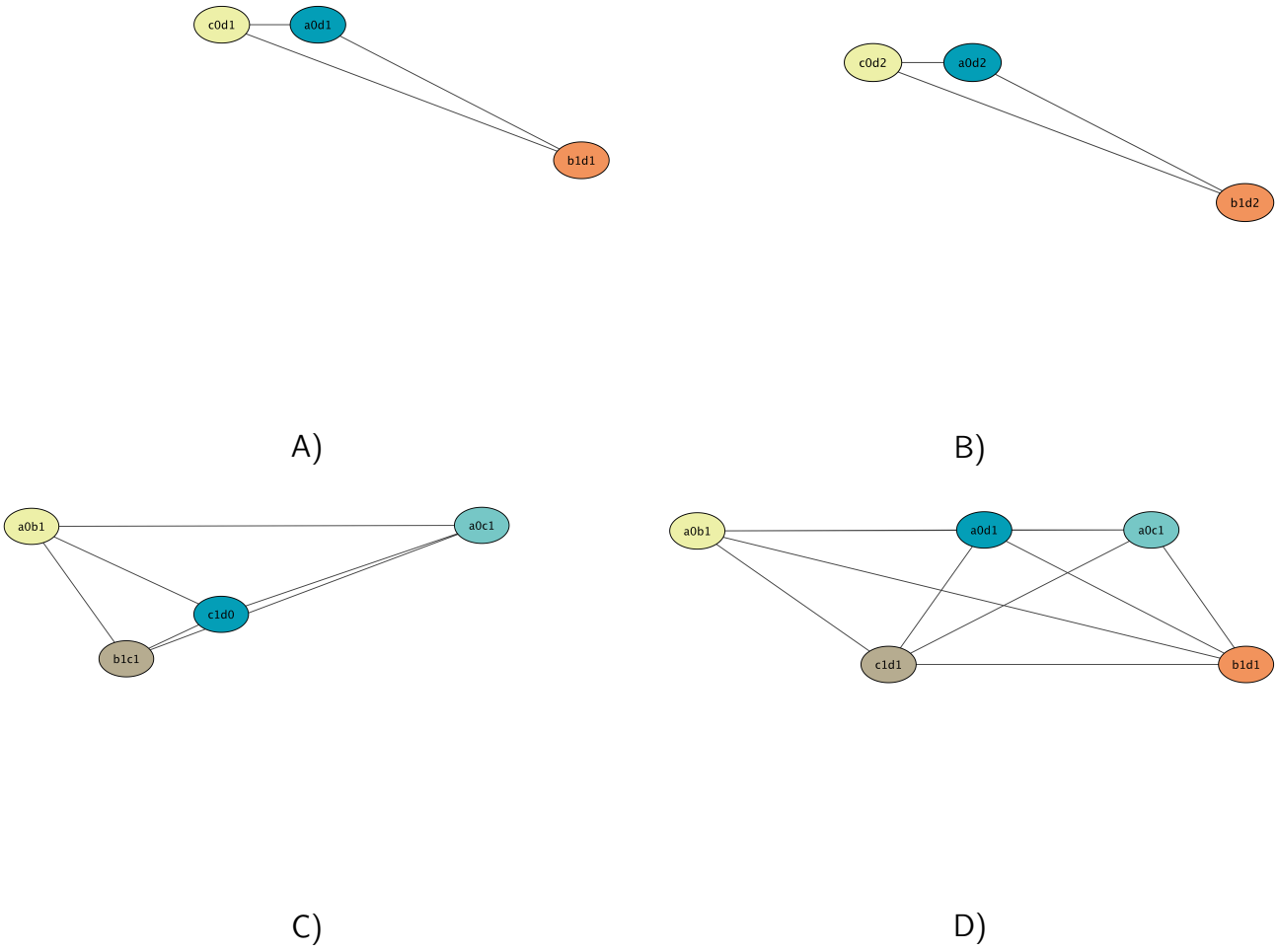


Figura 4.7: Cliques resultantes despues de seleccionar los distintos candidatos del nivel de recursión 1. A) Ξ(c0d1) = 3. B) Ξ(c0d2) = 3. C) Ξ(c1d0) = 4. D) Ξ(c1d1) = 5.

Al seleccionar el nodo c1d2, el grado de expansión resultante es $\Xi(c1d2) = |B_1|$, lo cual es el máximo nivel que puede aportar este candidato, además de que la cardinalidad del clique actual es igual al número de colores presentes en la matriz de cobertura completa por lo que no es necesario

seguir explorando los demás candidatos.

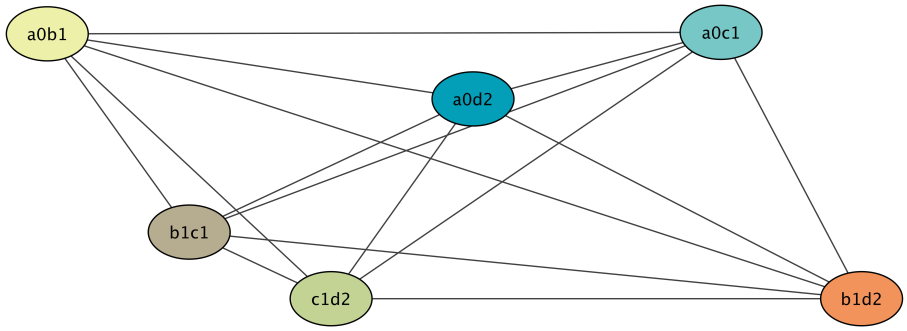


Figura 4.8: Máximo clique de 6 elementos para la construcción de un CA(9;2,4,3).

4.3 Análisis matemático

En esta sección se presenta el análisis matemático, con respecto al número de nodos y aristas así como el máximo clique obtenidos en el dominio de grafos para los casos que se encuentran reportados actualmente en el estado del arte y que son listados a continuación:

- k rotaciones y v traslaciones.
- k rotaciones y v traslaciones + v renglones constantes.
- k rotaciones y $v - f$ traslaciones.
- k rotaciones y $v - f$ traslaciones + v renglones constantes.

4.3.1 Número de grupos cíclicos ψ

Para determinar el máximo clique que es posible obtener a través del mapeo del problema de búsqueda de vectores inicializadores a grafos es necesario tomar en cuenta el número de grupos cíclicos

que es posible obtener para un vector inicializador de k elementos y fuerza t. En [19] se establece que para obtener el número de grupos cíclicos (denotado como ψ) se deben contabilizar todos los vectores de distancia eliminando aquellos que sean equivalentes y define una fórmula (Ecuación 4.3) que determina el número de grupos cíclicos para los casos en el que t es primo, $t = 4$ o $t = 6$.

Dado que los vectores de distancia pueden ser expresados como ecuaciones diofánticas de t variables tal que $\sum_{i=1}^{t} X_i = k$, estos pueden ser contabilizados como combinaciones con repetición $\binom{k+t-1}{t}$ y tomando en cuenta la restricción de que cada elemento debe ser mayor a 0, se pueden sustraer t elementos $\binom{k-1}{t}$. Además, dado que el último elemento del vector de distancia puede ser obtenido mediante la resta $k-\sum_{i=1}^{t} X_i$, este elemento puede ser removido de la ecuación obteniendo $\binom{k-1}{t-1}$. Tentativamente cada vector de distancia tiene t vectores equivalentes (incluyéndose a sí mismo), por lo que dividir $\frac{\binom{k-1}{t-1}}{t}$ podría ser una buena aproximación para el valor de ψ, pero esto no siempre ocurre para todos los casos y es necesario añadir elementos en caso de que existan vectores de distancias que posean menos de t vectores equivalentes.

$$\psi = \frac{1}{t}\left[\binom{k-1}{t-1} + \begin{cases} (t-1)(\lceil \frac{1}{t}(t-k \mod t)\rceil) & \textbf{Si } t \textbf{ es primo} \\ (t-1)(\lceil \frac{1}{t}(t-k \mod t)\rceil)+ & \\ (t-2)\frac{2}{t}(\binom{\frac{k}{2}-1}{\frac{t}{2}-1} - (\lceil \frac{2}{t}(\frac{t}{2}-k \mod \frac{t}{2})\rceil)) & \\ ((k+1) \mod 2) & \textbf{Si } t\textbf{=4} \\ (t-1)(\lceil \frac{1}{t}(t-k \mod t)\rceil)+ & \\ (t-2)\frac{3}{t}(\binom{\frac{k}{3}-1}{\frac{t}{3}-1} - (\lceil \frac{3}{t}(\frac{t}{3}-k \mod \frac{t}{3})\rceil)) & \\ (\frac{1}{3}(3-k \mod 3)) & \\ (t-3)\frac{2}{t}(\binom{\frac{k}{2}-1}{\frac{t}{2}-1} - (\lceil \frac{2}{t}(\frac{t}{2}-k \mod \frac{t}{2})\rceil)) & \\ ((k+1) \mod 2) & \textbf{Si } t\textbf{=6} \end{cases}\right] \tag{4.3}$$

A continuación se presenta una generalización de la Ecuación 4.3 para determinar el número de vectores de distancia que poseen $\frac{t}{d}$ vectores equivalentes $\forall d|t \equiv 0(\mod d) \wedge k \equiv 0(\mod d)$ y para cualquier valor de t.

Teorema 1

Cuando $k \not\equiv 0(\mod t)$, un vector de distancia posee $\frac{t}{d}$ vectores de distancia equivalentes siempre y cuando dicho vector puede ser dividido en d subvectores tal que $\{\forall i|X_i = X_{i+t/d}$ para $0 \leq i \leq \frac{t}{d}\}$ y t/d sea un número primo.

Demostración. Considerando el caso en el que $k = 6$ y $t = 4$, el vector de distancia $X = \{1, 2, 1, 2\}$ puede ser dividido en $d = 2$ subvectores $X_{0\ldots t/n-1} = \{1, 2\}$ y $X_{t/n\ldots t-1} = \{1, 2\}$ que cumplen la condición $\{\forall i|X_i = X_{i+t/d}$ para $0 \leq i \leq \frac{t}{d}\}$. Cada rotación que es posible aplicar al vector de distancia X para obtener un vector equivalente corresponde al intercambio de elementos con el mismo valor entre subvectores, por lo que el número de vectores equivalentes que es posible producir se reduce al número de elementos presentes en un subvector $\frac{t}{d}$.

$$X^1 = \{1, 2, 1, 2\}$$

$$X^2 = \{2, 1, 2, 1\}$$

□

Teorema 2

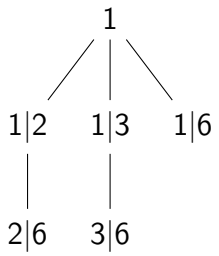
Sea D un conjunto de enteros positivos tal que $\{\forall d|t \equiv 0(\mod d) \wedge k \equiv 0(\mod d)$ para $d \in D\}$, el número de vectores de distancia que poseen $\frac{t}{d}$ vectores es calculado a partir de la siguiente función recursiva $f(d)$:

$$f(d) = \binom{\frac{k}{d} - 1}{\frac{t}{d} - 1} - \sum_{\forall j|(d|D_j) \wedge D_j \neq d} f(D_j)$$

Demostración. Determinar el número de vectores de distancia que poseen t vectores equivalentes $f(1)$ para el caso en el que $k = 12$ y $t = 6$.

$$D = \{1, 2, 3, 6\}$$

Contabilizar los vectores de distancia que cuentan con t vectores equivalentes incluye vectores de distancia que poseen $\frac{t}{2}$, $\frac{t}{3}$ y $\frac{t}{6}$ vectores equivalentes los cuales deben ser descartados. A su vez, los vectores de distancia que poseen $\frac{t}{2}$ y $\frac{t}{3}$ vectores equivalentes contienen un elemento que posee $\frac{t}{6}$ vectores equivalentes. Internamente esto se resuelve de manera recursiva creando el árbol de dependencia que se presenta a continuación:



$$f(6) = \binom{1}{0} - 0 = 1$$

$$\{2, 2, 2, 2, 2, 2\}$$

$$f(2) = \binom{5}{2} - (f(6)) = 9$$

$$\{1, 2, 3, 1, 2, 3\}, \{1, 3, 2, 1, 3, 2\}, \{4, 1, 1, 4, 1, 1\}$$

$$\{2, 3, 1, 2, 3, 1\}, \{3, 2, 1, 3, 2, 1\}, \{1, 1, 4, 1, 1, 4\}$$

$$\{3, 1, 2, 3, 1, 2\}, \{2, 1, 3, 2, 1, 3\}, \{1, 4, 1, 1, 4, 1\}$$

$$f(3) = \binom{3}{1} - (f(6)) = 2$$

$$\{1, 3, 1, 3, 1, 3\}$$

$$\{3, 1, 3, 1, 3, 1\}$$

$$f(1) = \binom{11}{5} - (f(2) + f(3) + f(6)) = 450$$

□

El número de grupos cíclicos, formado por los vectores de distancia divididos entre los $\frac{t}{d}$ vectores de distancia equivalentes, se encuentra dado por:

$$\psi = \sum_{\forall d | d \in D} \frac{d\ f(d)}{t}$$

$$\psi = \frac{450}{6} + \frac{2}{2} + \frac{9}{3} + \frac{1}{1} = 80$$

Corolario 1. *Cuando t es primo y $k \not\equiv 0(\mod t)$, existe para cada vector de distancia t vectores de distancia equivalentes.*

Corolario 2. *Cuando t es primo y $k \equiv 0(\mod t)$, existe un único vector de distancia con un vector de distancia equivalente.*

Corolario 3. *Cuando $t = 4$, $k \not\equiv 0(\mod t)$ y $k/2 \equiv 0(\mod t/2)$, existen*

$$\frac{2}{t}\binom{k/2-1}{t/2-1}$$

vectores de distancia con dos vectores de distancia equivalentes.

Corolario 4. *Cuando* $t = 4$ *y* $k \equiv 0(\mod t)$, *existen*

$$\frac{2}{t}\left(\binom{k/2-1}{t/2-1}-1\right)$$

vectores de distancia con dos vectores de distancia equivalentes y un único vector de distancia con un vector de distancia equivalente.

Corolario 5. *Cuando* $t = 6$ *y* $k \not\equiv 0(\mod t)$, $k/2 \not\equiv 0(\mod t/2)$ *y* $k/3 \equiv 0(\mod t/3)$, *existen*

$$\frac{3}{t}\binom{k/3-1}{t/3-1}$$

vectores de distancia con dos vectores de distancia equivalentes.

Corolario 6. *Cuando* $t = 6$ *y* $k \equiv 0(\mod t)$, $k/2 \not\equiv 0(\mod t/2)$ *y* $k/3 \equiv 0(\mod t/3)$, *existen*

$$\frac{3}{t}\left(\binom{k/2-1}{t/2-1}-1\right)$$

vectores de distancia con dos vectores de distancia equivalentes y un único vector de distancia con un vector de distancia equivalente.

Corolario 7. *Cuando* $t = 6$ *y* $k \equiv 0(\mod t)$, $k/2 \equiv 0(\mod t/2)$ *y* $k/3 \equiv 0(\mod t/3)$ *existen*

$$\frac{3}{t}\left(\binom{k/3-1}{t/3-1}-1\right)$$

vectores con dos vectores de distancia equivalentes, además de

$$\frac{2}{t}\left(\binom{k/2-1}{t/2-1}-1\right)$$

vectores con tres vectores de distancia equivalentes y un único vector de distancia con un vector de distancia equivalente.

4.3.2 k rotaciones y v traslaciones

- Nodos: Para este caso el número de nodos es equivalente a las maneras de tomar t elementos de k columnas $\binom{k}{t}$ por las v^t combinaciones que requieren estar presentes en cada uno de estos conjuntos.

$$\binom{k}{t}v^t \tag{4.4}$$

- Aristas: La cantidad de aristas presentes en el grafo puede ser aproximada para la función $\Lambda(x,y)\ \forall y|x,y\in\mathcal{V}$. Dado que se contemplan $\binom{k}{t}v^t$ nodos, se tiene que cada uno de estos puede compartir i columnas para $0<i\leq t$ con cualquier otro conjunto de t elementos, así tenemos que existen $\binom{t}{i}$ maneras de seleccionar las columnas compartidas y cada una de estas están presentes en $\binom{k-t}{t-i}$ conjuntos que son las distintas formas en que se pueden asignar valores en aquellas columnas que no son compartidas. Tomando en cuenta que solo se toman i columnas compartidas para un conjunto de t columnas con v^t combinaciones por conjunto, se tiene que cada combinación formada por i elementos se encuentra repetida v^{t-i} veces dentro de v^t combinaciones totales. Al tratarse de un grafo no dirigido se pueden eliminar los enlaces duplicados dividiendo entre dos.

$$\frac{1}{2}\binom{k}{t}v^t\sum_{i=0}^{t-1}\binom{t}{i}\binom{k-t}{t-i}v^{t-i} \tag{4.5}$$

- Máximo clique: Para determinar el máximo clique que un vector inicializador puede generar en el dominio de grafos es necesario calcular el número de grupos cíclicos ψ contenidos en el ya que las combinaciones se distribuyen de manera interna entre los t conjuntos de su mismo grupo. Una vez determinado el valor de ψ se calcula el número de combinaciones requeridas por grupo para producir las v^t combinaciones. El número de combinaciones requeridas por grupo está definida por el total de combinaciones a cubrir entre el número de traslaciones, por lo que para este caso es de v^t/v.

$$\psi\frac{v^t}{v} \tag{4.6}$$

4.3.3 k rotaciones y v traslaciones + v renglones constantes

- Nodos: Dado que se cuenta con v renglones constantes, los cuales aseguran la presencia de v combinaciones en $\binom{k}{t}$ conjuntos, por lo tanto las combinaciones faltantes por cada conjunto son $(v^t - v)$.

$$\binom{k}{t}(v^t - v) \tag{4.7}$$

- Aristas: Para este caso podemos tomar como base el número de aristas que se obtienen para el caso de k rotaciones y v traslaciones a diferencia de que unicamente se contemplan $\binom{k}{t}(v^t - v)$ nodos y se sustrae el caso en que no se comparte ninguna columna, ya que en estos conjuntos todas las combinaciones contenidas en ellos son compatibles, de esta manera se obtiene la expresión $\binom{k}{t}(v^t - v)(\binom{k-t}{t}(v^t - v) + \sum_{i=1}^{t-1}\binom{t}{i}\binom{k-t}{t-i}v^{t-i})$ y para finalizar se realiza la resta de los aportes de los $\binom{k}{t}v$ nodos con combinaciones constantes tomando en cuenta que ahora existen solo $v^{t-i} - 1$ combinaciones de i elementos repetidas en el conjunto de v^t dado que una combinación de estas es aportada por una combinación de t elementos constante.

$$\frac{1}{2}\left[\binom{k}{t}(v^t - v)\left(\binom{k-t}{t}(v^t - v) + \sum_{i=1}^{t-1}\binom{t}{i}\binom{k-t}{t-i}v^{t-i}\right) - \binom{k}{t}v\sum_{i=1}^{t-1}\binom{t}{i}\binom{k-t}{t-i}(v^{t-i} - 1)\right] \tag{4.8}$$

- Máximo clique: Tomando como base el caso anterior en el que el número de combinaciones necesarias que deben estar presentes en cada grupo cíclico es de $\frac{v^t}{v}$, al total de combinaciones se sustraen v combinaciones que son proporcionadas por los renglones constantes, por lo que ahora el número de combinaciones requeridas por grupo es de $\frac{v^t-v}{v}$.

$$\psi\frac{v^t-v}{v} \tag{4.9}$$

4.3.4 k rotaciones y $v-f$ traslaciones

- Nodos: Dado que se cuenta con f símbolos fijos, al finalizar la construcción es necesario agregar un covering array de $CA(t,k,f)$ con lo que se asegura la presencia de f^t combinaciones en $\binom{k}{t}$ conjuntos, por lo tanto las combinaciones faltantes por cada conjunto son (v^t-f^t).

$$\binom{k}{t}(v^t-f^t) \tag{4.10}$$

- Aristas: En este caso se sigue la misma lógica que se emplea para determinar el número de aristas para k rotaciones y v traslaciones + v renglones constantes a diferencia de que en este caso se sustraen f^t elementos en lugar de v, y se realiza la resta de los aportes de los $\binom{k}{t}f^t$ nodos que conforman las combinaciones presentes en el $CA(t,k,f)$ tomando en cuenta que ahora existen solo $v^{t-i}-f^{t-i}$ combinaciones en el conjunto de v^t.

$$\begin{aligned}\frac{1}{2}\Bigg[\binom{k}{t}(v^t-f^t)\Bigg(\binom{k-t}{t}(v^t-f^t)+\sum_{i=1}^{t-1}\binom{t}{i}\binom{k-t}{t-i}v^{t-i}\Bigg)\\ -\binom{k}{t}f^t\sum_{i=1}^{t-1}\binom{t}{i}\binom{k-t}{t-i}(v^{t-i}-f^{t-i})\Bigg]\end{aligned} \tag{4.11}$$

- Máximo clique: De igual manera que en el caso anterior el número de combinaciones necesarias que deben estar presentes en cada grupo cíclico es de $\frac{v^t-f^t}{v-f}$, ya que para este caso se encuentran

cubiertas f^t combinaciones y únicamente se realizan $v - f$ traslaciones.

$$\psi \frac{v^t - f^t}{v - f} \tag{4.12}$$

4.3.5 k rotaciones y $v - f$ traslaciones + v renglones constantes

- Nodos: De los $v^t - f^t$ nodos necesarios por cada t conjuntos para el caso de k rotaciones y $v - f$ traslaciones, son sustraídos $v - f$ nodos que representan los renglones constantes menos los f renglones constantes presentes en el $CA(t, k, f)$ que es añadido al final de la construcción.

$$\binom{k}{t}\Big(v^t - f^t - (v - f)\Big) \tag{4.13}$$

- Aristas: Para determinar el número de aristas que se obtienen para el caso k rotaciones y $v - f$ traslaciones + v renglones constantes se requiere sustraer $f^t - (v - f)$ de las v^t combinaciones originales, los cuales representan las combinaciones que resultan de fijar f símbolos y v renglones constantes. Posteriormente se eliminan aquellos enlaces que tienen los nodos restantes con nodos que representan a las combinaciones ya cubiertas, por lo que además de sustraen los aportes de los símbolos fijos como en el caso anterior y, de igual manera que en el caso con renglones constantes, eliminar el aporte de los $v - f$ renglones constantes que no son contemplados en el $CA(t, k, f)$.

$$\frac{1}{2}\Bigg[\binom{k}{t}\Big(v^t - f^t - (v - f)\Big)\Bigg(\binom{k-t}{t}\Big(v^t - f^t - (v - f)\Big) + \sum_{i=1}^{t-1}\binom{t}{i}\binom{k-t}{t-i}v^{t-i}\Bigg) - \binom{k}{t}f^t\sum_{i=1}^{t-1}\binom{t}{i}\binom{k-t}{t-i}(v^{t-i} - f^{t-i}) - \binom{k}{t}(v - f)\sum_{i=1}^{t-1}\binom{t}{i}\binom{k-t}{t-i}(v^{t-i} - 1)\Bigg] \tag{4.14}$$

- Máximo clique: Tomando como base el caso anterior en el que el número de combinaciones necesarias que deben estar presentes en cada grupo cíclico es de $\frac{v^t-f^t}{v-f}$, al total de combinaciones se sustraen $v-f$ combinaciones que son proporcionadas por los renglones constantes menos los renglones constantes presentes en el $CA(t,k,f)$, por lo que ahora el número de combinaciones requeridas por grupo es de $\frac{v^t-f^t-(v-f)}{v-f}$.

$$\psi\Big(\frac{v^t-f^t-(v-f)}{v-f}\Big) \tag{4.15}$$

4.4 Resumen

En este capítulo se presentó un enfoque original que permite trasladar el problema de la búsqueda de vectores inicializadores al dominio de grafos, el cual funciona de manera general para cualquier valor de t, k y v y VIG dado mediante la búsqueda del máximo clique en un grafo. Además se presentó un análisis para determinar el número de enlaces y nodos del grafo resultante así como el máximo clique necesario lograr una cobertura total empleando la construcción por vectores inicializadores. En el siguiente capítulo se presenta los detalles de la experimentación realizada para este trabajo de tesis, así como los resultados obtenidos.

5

Experimentación y resultados

Dentro de este capítulo se encuentran detallados los resultados encontrados mediante este trabajo de investigación. Éstos se encuentran divididos por cada generalización presentada dentro del capítulo 3 y que a su vez se encuentran clasificadas como construcciones obtenidas de forma directa empleando vectores inicializadores generalizados y resultados obtenidos mediante un proceso de postoptimización a partir de un resultado directo. Además, se presenta una variable de la construcción por símbolos fijos implementada al realizarse las rotaciones de un vector inicializador, permitiendo la construcción de arreglos ortogonales de fuerza dos.

5.1 Construcción de arreglos ortogonales de fuerza $t = 2$ por retroalimentación

En esta sección se presenta un análisis acerca de la construcción de arreglos ortogonales de índice unitario de fuerza dos bajo la siguiente hipótesis:

Hipótesis. *La construcción de arreglos ortogonales de índice unitario para fuerza $t = 2$ es posible mediante $k(v-1)$ renglones obtenidos a partir de un vector inicializador de tamaño k donde $k = v + 1$ y añadiendo un vector en el que todos sus elementos sean iguales a cero.*

Dado que un arreglo ortogonal contiene estrictamente las v^t combinaciones de manera única, el número de renglones N siempre es igual a $N = v^t$. Como $t = 2$ podemos expresar el número de filas como:

$$N = v^2$$

el número de filas que se espera obtener, denotado como N', a partir de un vector inicializador es de:

$$N' = k(v-1) + 1$$

dado que $k = v + 1$ es posible sustituirlo de la ecuación, obteniendo:

$$N' = (v+1)(v-1) + 1$$

aplicando las operaciones correspondientes se tiene que:

$$N' = v^2 - v + v - 1 + 1$$

$$N' = v^2$$

Como $N' = N$ la construcción propuesta es capaz de crear el número de filas necesarias para la construcción de un arreglo ortogonal para fuerza $t = 2$.

Hasta este punto las únicas opciones que se tienen para realizar esta construcción es fijar un elemento en el vector τ (es decir realizar $\delta = v - 1$ traslaciones) o bien fijar un elemento en el vector ρ (es decir realizar $\theta = v - 1$ rotaciones). A continuación analizaremos cada una de estas opciones para determinar si existe una equivalencia entre ambas construcciones o bien, determinar las ventajas

de una construcción sobre la otra. Para ello se analizará el caso más extraordinario que es posible construir, donde $k \equiv 0(\mod t)$, en el que existe un grupo cíclico representado por un vector de distancia con 1 vector de distancia equivalente.

Caso de construcción de un CA(9;2,4,3)

$X^1 = \{1,3\}$
0 1
1 2
2 3
$X^2 = \{3,1\}$
0 3

$X^3 = \{2,2\}$
0 2
1 3

Figura 5.1: Para el caso $t = 2$, $k = 4$, existen $\binom{k}{t} = 6$ distintas maneras de tomar t elementos de k columnas. Dichos elementos se encuentran agrupados en 2 grupos cíclicos representados por los vectores de distancias $X^1 = \{1,3\}, X^3 = \{2,2\}$ y sus respectivos vectores equivalentes.

Construcción mediante la generalización en traslación aplicando $\delta = v - 1$ traslaciones

Para determinar si es factible la construcción de arreglos ortogonales para fuerza dos realizando $\delta = v - 1$ traslaciones, se realiza el coloreo de la matriz de cobertura completa teniendo en cuenta los siguientes valores para el VIG:

$$\rho = \{0,1,2\};\ \tau = \{0,2,1\}$$

$$\pi = \{0,1,2\};\ \phi = \{1,1\};\ \sigma = \{1,1\}$$

Una vez efectuado el algoritmo de coloreo de la matriz de cobertura completa (Figura 5.2) obtenemos que para el grupo cíclico representado por el vector de distancia $X^1 = \{1,3\}$ se requieren 4 colores y para el grupo cíclico representado por el vector de distancia $X^3 = \{2,2\}$ se necesitan 3 colores.

ab	bc	cd	ad	ac	bd
01	01	01	01	01	01
02	02	02	02	02	02
10	10	10	10	10	10
11	11	11	11	11	11
12	12	12	12	12	12
20	20	20	20	20	20
21	21	21	21	21	21
22	22	22	22	22	22

Figura 5.2: Coloreo de la matriz de cobertura completa empleando un símbolo fijo.

Dado que debe existir un representante de cada color contenido en el vector inicializador y únicamente se puede incluir un elemento por órbita del grupo, el número de representantes que pueden ser contenidos se limita a dicho número de órbitas. Por esta razón, no es posible construir un covering array mediante un solo vector inicializador si el número de colores presentes en un grupo cíclico es mayor que el número de órbitas que componen dicho de grupo.

Realizando $\delta = v - 1$ traslaciones, no es posible la construcción de arreglos ortogonales de fuerza dos, debido a que el grupo cíclico representado por el vector de distancia para el que solo existe un vector de distancia equivalente ($X^3 = \{2, 2\}$) no puede ser completado en dos órbitas ya que la que la combinación $\{1, 1\}$ no genera nuevos elementos al efectuarse las rotaciones (Figura 5.3 (B)), únicamente cuando el vector es trasladado, esto porque se encuentra compuesto de elementos del mismo símbolo. Caso contrario se produce para combinación $\{1, 2\}$ en el que se producen nuevos elementos al efectuarse la rotación, aunque vuelve a caer en redundancia al ser trasladado (Figura 5.3 (C)).

Construcción mediante la generalización en retroalimentación aplicando $\theta = k(v - 1)$ rotaciones

Ahora se analizará la construcción de arreglos ortogonales para fuerza dos realizando $\theta = k(v-1)$ rotaciones, por lo que a continuación se presenta el coloreo de la matriz de cobertura completa teniendo en cuenta los siguientes valores para el VIG:

$$\rho = \{1, 2, 0\};\ \tau = \{0, 1, \}$$

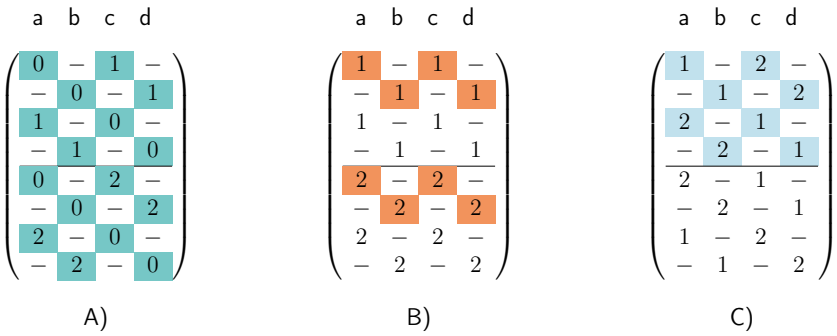


Figura 5.3: Proceso de coloreo de elementos del grupo cíclico representado por el vector de distancia ($X^3 = \{2, 2\}$) para la construcción de símbolos fijos en traslaciones.

$$\pi = \{0, 1, 2\};\ \phi = \{1, 1\};\ \sigma = \{1, 1\}$$

ab	bc	cd	ad	ac	bd
01	01	01	01	01	01
02	02	02	02	02	02
10	10	10	10	10	10
11	11	11	11	11	11
12	12	12	12	12	12
20	20	20	20	20	20
21	21	21	21	21	21
22	22	22	22	22	22

Figura 5.4: Coloreo de matriz de cobertura completa empleando retroalimentación.

A diferencia de la construcción por traslaciones, la construcción por retroalimentación en las rotaciones genera nuevas combinaciones al efectuarse cada rotación, completando así el grupo cíclico representado por el vector de distancia ($X^3 = \{2, 2\}$) tal y como se muestra en la Figura 5.5.

Como conclusión se puede afirmar que existe una relativa equivalencia entre las construcciones por retroalimentación y la construcción por traslaciones debido a que, aunque no generen el mismo vecindario para un mismo color dado un nodo en específico, ambos generan una cantidad de colores equivalente cuando el vector de distancia que representa el grupo cíclico tiene t vectores equivalentes. Además se puede concluir que la construcción por retroalimentación favorece las construcciones que están compuestas por grupos cíclicos representados por vectores de distancia que tienen menos de t

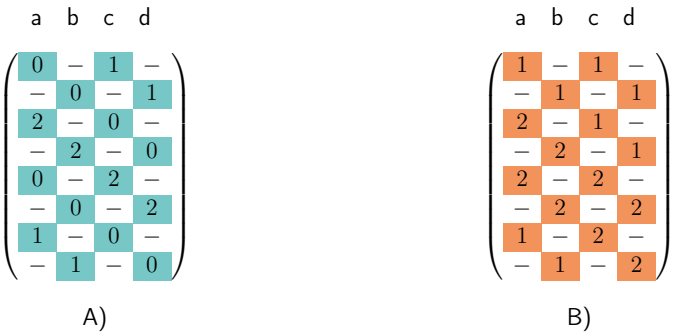


Figura 5.5: Proceso de coloreo de elementos del grupo cíclico representado por el vector de distancia ($X^3 = \{2, 2\}$) para la construcción de símbolos fijos en la retroalimentación en rotaciones.

vectores de distancias equivalentes.

5.2 Resultados obtenidos empleando la generalización en retroalimentación

En la Tabla 5.1 se presentan los resultados obtenidos como conclusión de la sección previa presentada en este capítulo en el que se aborda la construcción de arreglos ortogonales de índice unitario para fuerza dos empleando la generalización en retroalimentación definiendo los valores $f = 1 \wedge \rho_0 = 0 \wedge \rho_{i+f} = (i \mod v) + f\{\forall i \mid 0 \leq i < v - f\}$.

5.3 Resultados obtenidos empleando la generalización en traslación

Resultados obtenidos de forma directa

A continuación se presentan en la Tabla 5.2 los resultados obtenidos de forma directa mediante vectores inicializadores generalizados. Algunos resultados fueron obtenidos agregando v renglones

N	t	k	v	VI
				$f = 1 \wedge \rho_0 = 0 \wedge \rho_{i+f} = (i \mod v) + f\{\forall i \mid 0 \leq i < v - f\}$
4	2	3	2	1 1 0
9	2	4	3	2 2 0 2
16	2	5	4	3 3 0 2 1
25	2	6	5	4 4 0 1 4 1
49	2	8	7	6 6 0 2 1 5 1 4
64	2	9	8	7 7 0 6 5 2 3 6 4
81	2	10	9	8 8 0 3 2 4 2 6 1 2
121	2	12	11	10 10 8 2 9 5 8 10 5 6 5 0
169	2	14	13	0 1 1 7 3 7 4 2 3 5 12 3 8 7
256	2	17	16	0 1 1 4 1 9 7 11 1 8 2 12 13 4 6 2 1
289	2	18	17	0 1 1 3 15 6 12 16 1 6 16 14 9 2 10 13 10 9
361	2	20	19	0 1 1 3 6 1 9 2 18 12 4 11 16 17 5 14 18 14 11 10
528	2	24	23	0 1 1 4 16 8 5 16 11 16 7 13 14 12 5 13 7 11 21 1 8 17 13 12
625	2	26	25	0 1 4 18 20 3 3 23 3 24 2 12 22 10 20 8 23 2 20 14 17 24 12 2 5 4

Tabla 5.1: Arreglos ortogonales de fuerza $t = 2$, construidos mediante vectores inicializadores aplicando retroalimentación en el vector ρ tal que $\rho_0 = 0 \wedge \rho_{v-1} = 1 \wedge (\rho_i = i+1\{\forall i \mid 0 < i < v-1\})$.

constantes (indicados con la abreviación *vconst*), o una columna adicional empleando el método *zero-sum* [10].

Resultados usando postoptimización

Los resultados que se presentan dentro de la Tabla 5.3 fueron obtenidos mediante un proceso de postoptimización aplicado a un covering array construido a partir de vectores inicializadores generalizados. La metodología de postoptimización empleada consiste en 3 componentes: un detector de redundancia; un reductor de filas; y un algoritmo de recocido simulado encargado de reducir el número de combinaciones restantes. Este algoritmo se encuentra reportado en [57].

N	t	k	v	VI
$f = 1 \wedge \tau_0 = 0 \wedge \tau_{i+f} = (i \mod v) + f\{\forall i \mid 0 \leq i < v - f\}$				
19	2	6	4	0 1 1 2 1 2
29	2	7	5	0 2 3 1 1 4 1
33	2	8	5	0 1 1 2 2 4 1 4
46	2	9	6	0 1 1 2 1 1 3 5 3
61	2	10	7	0 1 1 1 3 4 1 3 2 6
78	2	11	8	0 1 1 2 2 4 2 5 6 3 2
105	2	13	9	0 1 1 1 3 2 1 6 2 5 5 3 4
127	2	14	10	0 1 1 2 1 4 7 2 7 5 3 9 2 7
136	2	15	10	0 1 1 2 1 4 6 4 1 1 4 8 9 5 8
145	2	16	10	0 1 1 2 1 4 6 2 9 6 1 5 4 9 7 4
161	2	16	11	0 1 1 2 1 4 6 3 1 7 8 2 9 10 5 8
171	2	17	11	0 1 1 2 1 4 9 7 4 1 3 9 10 7 4 8 10
188	2	17	12	0 1 1 2 1 4 1 1 8 5 3 9 1 3 7 6 11
199	2	18	12	0 1 1 2 1 4 6 10 4 3 1 10 6 8 5 11 2 6
210	2	19	12	0 1 1 2 1 4 6 2 10 5 7 6 3 1 11 5 9 3 8
221	2	20	12	0 1 1 2 1 4 6 2 10 5 9 1 3 7 9 1 5 3 6 11
$\tau = \{1, 0, 3, 2, 4\}$ + zero-sum				
40	2	21	5	4 4 0 4 3 4 1 3 1 3 2 2 2 4 1 0 2 1 1 2
$\tau = \{1, 2, 0, 4, 5, 3\}$ + vconst + zero-sum				
57	2	18	6	5 4 2 3 0 1 0 3 2 4 0 5 3 1 5 1 5
$\tau_i = i + 1 \mod v\{\forall i \mid 0 \leq i < v\}$				
8	3	4	2	1 1 0 1
10	3	5	2	1 0 0 0 0
12	3	6	2	0 0 0 1 0 0
66	3	22	3	0 0 0 0 0 0 1 0 1 2 2 1 0 1 1 2 0 1 0 2 1 1
69	3	23	3	0 0 0 0 0 0 1 2 2 0 1 0 1 1 0 1 1 0 1 0 2 2 1
42	4	21	2	0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 1
50	4	25	2	0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1 0 1 0 1 0 0 1 1 1
$\tau_i = i\{\forall i \mid 0 \leq i < v\} + 1$ renglón de ceros				
17	3	16	2	0 0 0 0 1 0 1 1 0 1 1 1 0 1 1 1
$\tau_i = i + 1 \mod v\{\forall i \mid 0 \leq i < v\}$ + vconst				
45	3	14	3	0 0 1 0 1 2 2 0 0 2 0 2 1 1
51	3	16	3	0 1 0 0 0 1 2 1 1 0 1 1 1 0 2 0
$\tau_i = i + 1 \mod v\{\forall i \mid 0 \leq i < v\}$ + vconst + zero-sum				
24	4	12	2	0 0 1 0 0 0 1 1 1 0 1

Tabla 5.2: Resultados que igualan las cotas reportadas actualmente en el estado del arte [12] construidos empleando la construcción de vectores inicializadores sin postprocesamiento.

5.4 Resultados obtenidos empleando la generalización en traslación, retroalimentación y orden

En la Tabla 5.4 se muestran algunos resultados obtenidos con la interacción de las primeras dos generalizaciones (generalización en retroalimentación y traslación) con la tercera generalización de

N	t	k	v	VI
				$f = 2 \wedge \tau_i = i\{\forall i \mid 0 \leq i < f\} \wedge \tau_{i+f} = (i \mod v) + f\{\forall i \mid 0 \leq i < v - f\}$
22	2	9	4	0 2 1 2 3 3 1 2 0
71	2	13	7	0 2 1 2 3 0 1 6 3 4 4 3 6
76	2	14	7	0 2 1 2 2 0 1 3 5 4 6 2 3 6
79	2	15	7	0 2 1 2 1 3 5 4 4 5 3 3 1 5 0
84	2	13	8	7 7 0 2 4 7 2 3 3 1 3 7 6
150	2	18	10	9 9 2 8 7 4 0 2 4 1 6 9 9 3 9 5 2 2
186	2	20	11	10 10 8 4 0 4 10 9 3 5 9 1 4 2 3 4 8 10 9 2
				$f = 3 \wedge \tau_i = i\{\forall i \mid 0 \leq i < f\} \wedge \tau_{i+f} = (i \mod v) + f\{\forall i \mid 0 \leq i < v - f\}$
104	2	18	8	7 7 2 5 6 4 6 0 6 1 5 4 1 5 2 4 0 6
107	2	19	8	7 7 2 5 6 4 6 0 6 1 7 2 6 0 1 6 4 5 4
108	2	20	8	7 7 2 5 6 4 6 0 6 1 7 6 1 4 2 7 0 4 3 3
132	2	20	9	8 8 6 8 1 7 4 3 4 2 7 0 4 8 7 0 1 6 2 7
139	2	21	9	8 8 6 8 1 7 4 3 4 2 7 0 4 8 1 7 5 0 2 7 7
155	2	20	10	9 9 3 1 6 9 4 9 2 4 9 8 5 0 0 7 2 6 4 1
162	2	21	10	9 9 3 1 6 9 4 9 2 4 3 0 6 3 7 1 7 0 4 2 3
166	2	22	10	9 9 3 1 6 9 4 9 2 4 0 8 5 9 8 8 0 6 2 4 4 1
171	2	23	10	9 9 3 1 6 9 4 9 2 4 0 8 5 9 8 7 8 6 0 6 1 5 2
192	2	22	11	10 10 8 4 2 8 3 5 4 1 3 7 8 0 4 5 0 5 2 3 1 7
200	2	23	11	10 10 8 4 2 8 3 5 4 1 3 7 8 0 8 5 3 2 10 0 1 5 9
203	2	24	11	10 10 8 4 2 8 3 5 4 1 3 7 8 0 8 5 3 9 2 4 4 0 1 3
				$f = 4 \wedge \tau_i = i\{\forall i \mid 0 \leq i < f\} \wedge \tau_{i+f} = (i \mod v) + f\{\forall i \mid 0 \leq i < v - f\}$
178	2	25	10	9 9 7 9 2 4 7 6 0 9 1 4 3 7 5 6 4 1 0 9 2 9 9 3 4
				$\tau = \{1, 2, 3, 0, 5, 4\}$
55	2	13	6	5 5 1 0 5 2 1 3 0 0 4 0 4
				$\tau = \{1, 2, 0, 4, 5, 3\}$
59	2	19	6	5 4 2 3 0 1 1 3 1 4 0 4 2 4 5 0 5 4
61	2	20	6	5 4 2 3 0 1 1 4 0 5 2 3 5 3 0 2 5 2 0
63	2	22	6	5 5 0 3 4 0 1 0 4 3 3 0 0 5 1 0 0 3 1 2 3 5
65	2	23	6	5 5 0 3 4 0 1 0 4 3 3 0 3 2 0 4 1 1 3 5 2 2 3
66	2	24	6	5 5 0 3 4 0 1 0 4 3 3 0 3 2 0 4 1 1 3 5 2 2 3
67	2	26	6	5 5 0 3 4 0 1 0 4 3 3 0 2 2 4 0 4 1 5 4 1 4 3 0 4
68	2	28	6	5 5 0 3 4 0 1 0 4 3 3 0 2 2 4 0 1 3 0 4 0 4 5 0 0 5 3
				$\tau = \{1, 2, 3, 0, 5, 6, 4\}$
87	2	22	7	6 6 1 5 4 5 3 2 6 3 0 1 6 2 1 2 0 4 0 5 5 4
				$\tau = \{1, 2, 0, 4, 5, 3, 6\}$
88	2	29	7	6 6 1 5 4 5 3 2 6 3 0 1 6 2 4 0 5 1 0 4 2 4 5 2 0 2 1 3 3
				$\tau = \{1, 0, 3, 2, 5, 4, 6\}$
93	3	46	3	2 2 2 1 0 2 1 2 0 1 1 0 1 0 1 2 2 0 0 1 2 1 1 0 2 0 1 1 0 1 2 0 2 1 2 2 1 1 1 1 1 0 0 0 2 1
95	3	47	3	2 2 2 1 0 2 1 2 0 1 1 0 1 0 1 2 2 0 0 1 2 1 1 0 2 2 0 1 2 0 2 0 2 0 0 2 1 2 2 2 2 2 1 1 1 0 0
97	3	48	3	2 2 2 1 0 2 1 2 0 1 1 0 1 0 1 2 2 0 0 1 2 1 1 0 2 2 0 2 1 0 2 1 1 0 1 0 2 0 0 0 2 1 1 2 2 2 2 2
99	3	49	3	2 2 2 1 0 2 1 2 0 1 1 0 1 0 1 2 2 0 0 1 2 1 1 0 2 2 0 2 1 0 1 0 2 2 0 1 1 2 1 1 0 0 0 2 1 1 0 2 0

Tabla 5.3: Resultados que igualan las cotas reportadas actualmente en el estado del arte [12] construidos empleando covering arrays parciales construidos mediante vectores inicializadores y postprocesamiento.

orden. Los resultados obtenidos no igualan a las mejores cotas reportadas en el estado del arte aun así, las cotas obtenidas resultan ser soluciones competitivas.

N	Mejor reportado	t	k	v	VI
					$\tau = \{0,1\} \wedge (\pi_i = k-1-i\{\forall i \mid 0 < i < k\}) \wedge \sigma = \{0,1\}$
15	10	3	7	2	1 1 1 0 0 1 0
					$\rho = \{0,2,1\} \wedge (\pi_i = k-1-i\{\forall i \mid 0 < i < k\}) \wedge \sigma = \{0,1\}$
53	45	3	13	3	2 2 2 1 2 0 1 0 1 1 0 0 2
					$\tau_i = i+1 \mod v\{\forall i \mid 0 \leq i < v\} \wedge \phi_0 = 0 \wedge \phi_i = 1\{\forall i \mid 0 < i < k\}$
196	194	3	49	4	3 3 3 2 1 3 2 3 0 2 0 1 1 3 0 3 2 2 3 3 3 1 3 0 0 3 2 3 0 2 0 1 1 0 0 3 1 3 2 3 1 1 1 3 2 1 0 2 3
244	240	3	61	4	3 3 3 2 1 3 2 3 0 2 0 1 1 3 0 3 2 2 3 3 3 1 3 0 0 3 2 3 0 2 1 3 2 2 0 1 2 2 2 1 2 2 0 1 3 2 3 1 2 2 1 0 3 0 2 0 2 0 3 1 1
405	404	3	81	5	4 4 4 2 1 0 2 0 1 2 2 4 0 3 1 2 1 0 3 0 0 4 0 2 4 0 0 3 4 4 1 3 2 0 4 4 0 0 0 1 0 0 3 3 0 2 1 2 4 2 4 2 3 2 1 4 4 2 3 4 0 4 2 0 1 1 1 3 2 4 0 3 4 4 4 2 1 1 1 3 0

Tabla 5.4: Resultados construidos empleando la generalización en traslación, retroalimentación y orden comparadas con las cotas reportadas actualmente en el estado del arte [12].

5.5 Resumen

En este capítulo se presentaron los resultados obtenidos en comparación con las mejores cotas obtenidas actualmente en el estado del arte. Los resultados que se presentaron fueron obtenidos ya sea de manera directa (es decir, empleando únicamente el concepto de vectores inicializadores generalizados) o bien mediante un proceso de postoptimizacion de un resultado obtenido de manera directa. Además, se demostró la factibilidad de construir arreglos ortogonales de fuerza dos mediante vectores inicializadores generalizados empleando retroalimentación de elementos al efectuarse una rotación en combinación con la construcción por símbolos fijos que anteriormente solo se realizaba al efectuarse una traslación del vector inicializador. En el siguiente capítulo se presentan las conclusiones y principales contribuciones de este trabajo de tesis.

6

Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones que han sido obtenidas en este trabajo de investigación. A continuación se presentan algunas de las ventajas que se obtienen al realizar el cambio de representación al dominio de grafos y los resultados obtenidos. Para finalizar se incluyen aspectos a considerar como trabajo futuro que se derivan de esta generalización propuesta.

6.1 Principales contribuciones

En este documento de tesis se exploró el problema de la construcción de covering arrays mediante el uso de vectores inicializadores debido a que era una área que ya había agotado la posibilidad de construir nuevas cotas con lo que se encontraba reportado actualmente, además de que no existía hasta el momento un método para la búsqueda del vector inicializador, salvo las construcciones algebraicas que son limitadas para ciertos parámetros de t, k y v. Inicialmente se presentó de manera formal la generalización de las operaciones que actualmente están reportadas en el estado del arte además de que abre la posibilidad de desarrollar nuevos operadores que permiten modular el número

de filas N del covering array resultante.

Se planteó cambiar el dominio del problema de búsqueda de vectores inicializadores al dominio de grafos, ya que de esta manera se podrían modelar de forma adecuada la compatibilidad entre combinaciones evitando redundancias entre las mismas. Mediante este trabajo de tesis se logró igualar 77 cotas reportadas en el estado del arte. Aunque las cotas obtenidas sólo igualaron cotas reportadas actualmente en el estado del arte, esta generalización podría obtener resultados para casos de t, k y v mayores, los cuales no fueron explorados por cuestiones de tiempo.

Como principal contribución se presenta un cambio de representación del problema que permite generalizar la búsqueda de vectores inicializadores de una manera eficiente. Una vez que el problema ha sido trasladado al dominio de grafos, es posible discriminar ciertos valores, para el vector inicializador, que no vale la pena explorar dado que solo se consideran como criterio de compatibilidad aquellos nodos que minimicen la redundancia. Es importante aclarar que bajo la representación del dominio de grafos, el procedimiento para la búsqueda de vectores inicializadores, el cual se traduce como la búsqueda del máximo clique, es el mismo para todos los casos ya que este procedimiento no se encuentra ligado a las características de los valores de t, k y v, ni el tipo de operadores que se definan para la construcción del covering array.

Trabajar en el dominio de grafos permite identificar únicamente aquellas combinaciones que son necesarias para proporcionar cobertura completa, por lo que puede existir el caso de encontrar un vector inicializador que contenga elementos comodines que pueden ser remplazados por cualquier elemento del conjunto $\mathbb{Z}_v$ sin impactar la cobertura del covering array. Además, permite la construcción de un covering array mediante un conjunto de n vectores, aplicando de manera iterativa la búsqueda del máximo clique e ingresando nuevamente el grafo de entrada excluyendo las combinaciones presentes en el máximo clique encontrado en la iteración anterior.

6.2 Trabajos futuros

A continuación se presentan algunos aspectos que pueden ser retomados como trabajos futuros a partir de este trabajo de investigación.

- Dado que ha quedado demostrado que es posible trasladar el problema de la construcción de vectores inicializadores al dominio de grafos, sería conveniente comenzar ha explorar implementaciones reportadas actualmente en el estado del arte que abordan el problema del máximo clique para determinar la mejor heurística que explote las características propias de los vectores inicializadores y de esta manera agilizar su proceso de búsqueda.

- Ya que los valores definidos en los vectores para un vector inicializador generalizado determinan la estructura del grafo, es posible desarrollar un algoritmo que mediante un proceso de sintonización de los valores para dichos vectores, determine la mejor configuración tal que el máximo clique presente en el grafo represente el vector inicializador que aporte la mayor cobertura posible para los valores t, k y v del covering array.

- Explorar la construcción de covering arrays mediante vectores inicializadores generalizados para regiones de t, k y v que no fueron exploradas dentro de este trabajo de tesis.

Bibliografía

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

[2] Bracho-Rios, J., Torres-Jimenez, J., and Rodriguez-Tello, E. (2009). A new backtracking algorithm for constructing binary covering arrays of variable strength. In *MICAI 2009: Advances in Artificial Intelligence, 8th Mexican International Conference on Artificial Intelligence, Guanajuato, México, November 9-13, 2009. Proceedings*, pages 397–407. Springer Science + Business Media.

[3] Bryce, R. C. and Colbourn, C. J. (2007). The density algorithm for pairwise interaction testing. *Software Testing, Verification and Reliability*, 17(3):159–182.

[4] Bush, K. A. (1952). Orthogonal arrays of index unity. *Ann. Math. Statist.*, 23(3):426–434.

[5] Calvagna, A. and Gargantini, A. (2011). T-wise combinatorial interaction test suites construction based on coverage inheritance. *Software Testing, Verification and Reliability*, 22(7):507–526.

[6] Chateauneuf, M. A., Colbourn, C. J., and Kreher, D. L. (1999). *Designs, Codes and Cryptography*, 16(3):235–242.

[7] Chateauneuf, M. A. and Kreher, D. L. (2002). On the state of strength-three covering arrays. *Journal of Combinatorial Designs*, 10(4):217–238.

[8] Cohen, D. M., Dalal, S. R., Fredman, M. L., and Patton, G. C. (1997a). The AETG system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444.

[9] Cohen, D. M., Dalal, S. R., Fredman, M. L., and Patton, G. C. (1997b). The aetg system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444.

[10] Colbourn, C. (2010). *CRC Handbook of Combinatorial Designs*. Discrete Mathematics and Its Applications. CRC Press. ISBN 9781420049954.

[11] Colbourn, C. J. (2009). Covering arrays from cyclotomy. *Designs, Codes and Cryptography*, 55(2-3):201–219.

[12] Colbourn, C. J. (2016). Covering array tables for t=2,3,4,5,6. http://www.public.asu.edu/ ccolbou/src/tabby/catable.html. Accessed: 2017-09-20.

[13] Colbourn, C. J., Martirosyan, S. S., Mullen, G. L., Shasha, D., Sherwood, G. B., and Yucas, J. L. (2006). Products of mixed covering arrays of strength two. *Journal of Combinatorial Designs*, 14(2):124–138.

[14] Covarrubias-Flores, E. (2008). Cálculo de covering arrays binarios de fuerza variable, usando un algoritmo de recocido simulado. Master's thesis, CINVESTAV-TAMAULIPAS.

[15] Forbes, M., Lawrence, J., Lei, Y., Kacker, R., and Kuhn, D. (2008). Refining the in-parameter-order strategy for constructing covering arrays.

[16] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549. Applications of Integer Programming.

[17] Glover, F. (1989). Tabu search—part i. *ORSA Journal on Computing*, 1(3):190–206.

[18] Glover, F. (1990). Tabu search—part II. *ORSA Journal on Computing*, 2(1):4–32.

[19] Gonzalez-Gomez, A. (2013). Construcción de covering arrays utilizando vectores inicializadores. Master's thesis, CINVESTAV-TAMAULIPAS.

[20] Guan, D. (1998). Generalized gray codes with applications. 16:841–848.

[21] H. Avila-George, J. T.-J. and Hernández, V. (2012). Parallel simulated annealing for the covering arrays construction problem. page 1.

[22] Hartman, A. (2005). Software and hardware testing using combinatorial covering suites. In *Graph Theory, Combinatorics and Algorithms*, pages 237–266. Springer Science + Business Media.

[23] Hartman, A. and Raskin, L. (2004). Problems and algorithms for covering arrays. *Discrete Mathematics*, 284(1-3):149 – 156. Special Issue in Honour of Curt Lindner on His 65th Birthday.

[24] Hnich, B., Prestwich, S. D., Selensky, E., and Smith, B. M. (2006). Constraint models for the covering test problem. *Constraints*, 11(2-3):199–219.

[25] Holland, J. H. (1992). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control and artificial intelligence*. MIT Press, Cambridge, MA, USA. ISBN 0262082136.

[26] Katona, G. O. H. (1973). Two applications (for search theory and truth functions) of sperner type theorems. *Periodica Mathematica Hungarica*, 3(1-2):19–26.

[27] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

[28] Kitsos, P., Simos, D. E., Torres-Jimenez, J., and Voyiatzis, A. G. (2015). Exciting FPGA cryptographic trojans using combinatorial testing. In *26th IEEE International Symposium on Software Reliability Engineering, ISSRE 2015, Gaithersbury, MD, USA, November 2-5, 2015*. Institute of Electrical & Electronics Engineers (IEEE).

[29] Kleitman, D. J. and Spencer, J. (1973). Families of k-independent sets. *Discrete Mathematics*, 6(3):255–262.

[30] Kuhn, D., Wallace, D., and Gallo, A. (2004). Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering*, 30(6):418–421.

[31] Kuhn, D. R., Kacker, R., and Lei, Y. (2010). Practical combinatorial testing. Technical report.

[32] Lei, Y., Kacker, R., Kuhn, D. R., Okun, V., and Lawrence, J. (2007). IPOG: A general strategy for t-way software testing. In *14th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS 2007), 26-29 March 2007, Tucson, Arizona, USA*. Institute of Electrical & Electronics Engineers (IEEE).

[33] Lei, Y. and Tai, K. (2005). In-parameter-order: a test generation strategy for pairwise testing.

[34] LI, C. (2017). Maxclq for maxclique. http://home.mis.u-picardie.fr/ cli/EnglishPage.html. Accessed: 2017-04-17.

[35] Lobb, J. R., Colbourn, C. J., Danziger, P., Stevens, B., and Torres-Jimenez, J. (2012). Cover starters for covering arrays of strength two. *Discrete Mathematics*, 312(5):943–956.

[36] Lopez-Escogido, D., Torres-Jimenez, J., Rodriguez-Tello, E., and Rangel-Valdez, N. (2008). Strength two covering arrays construction using a SAT representation. In *MICAI 2008: Advances in Artificial Intelligence, 7th Mexican International Conference on Artificial Intelligence, Atizapán de Zaragoza, Mexico, October 27-31, 2008, Proceedings*, pages 44–53. Springer Science + Business Media.

[37] Martinez-Pena, J. and Torres-Jimenez, J. (2010). A branch and bound algorithm for ternary covering arrays construction using trinomial coefficients. *Res. Comput. Sci*, 49:61–71.

[38] Martinez-Pena, J., Torres-Jimenez, J., Rangel-Valdez, N., and Avila-George, H. (2010a). A heuristic approach for constructing ternary covering arrays using trinomial coefficients. In *Advances in Artificial Intelligence – IBERAMIA 2010*, pages 572–581. Springer Science + Business Media.

[39] Martinez-Pena, J., Torres-Jimenez, J., Rangel-Valdez, N., and Avila-George, H. (2010b). *A heuristic approach for constructing ternary covering arrays using trinomial coefficients*, pages 572–581. Springer Berlin Heidelberg, Berlin, Heidelberg.

[40] Meagher, K. (2002). Non-isomorphic generation of covering arrays. Technical report, University of Regina.

[41] Meagher, K. and Stevens, B. (2004). Group construction of covering arrays. *Journal of Combinatorial Designs*, 13(1):70–77.

[42] Nurmela, K. J. (2004). Upper bounds for covering arrays by tabu search. *Discrete Applied Mathematics*, 138(1-2):143–152.

[43] Ostergard, P. (2017). Cliquer - routines for clique searching. https://users.aalto.fi/ pat/cliquer.html. Accessed: 2017-04-17.

[44] Perez-Torres, J. C. and Torres-Jimenez, J. (2017). A graph-based postoptimization approach for covering arrays. *Quality and Reliability Engineering International*.

[45] Raaphorst, S., Moura, L., and Stevens, B. (2013). A construction for strength-3 covering arrays from linear feedback shift register sequences. *Designs, Codes and Cryptography*, 73(3):949–968.

[46] Renyi, A. (2007). *Foundations of Probability*. Dover books on mathematics. Dover Publications. ISBN 9780486462615.

[47] Sherwood, G. B., Martirosyan, S. S., and Colbourn, C. J. (2006). Covering arrays of higher strength from permutation vectors. *Journal of Combinatorial Designs*, 14(3):202–213.

[48] Shiba, T., Tsuchiya, T., and Kikuno, T. (2004). Using artificial life techniques to generate test cases for combinatorial testing. In *28th International Computer Software and Applications Conference (COMPSAC 2004), Design and Assessment of Trustworthy Software-Based Systems, 27-30 September 2004, Hong Kong, China, Proceedings*, pages 72–77 vol.1.

[49] Stardom, J. (2001). *Metaheuristics and the search for covering and packing arrays*. PhD thesis, Simon Fraser University.

[50] Tang and Woo (1983). Exhaustive test pattern generation with constant weight vectors. *IEEE Transactions on Computers*, C-32(12):1145–1150.

[51] Timana-Pena, J., Cobos-Lozada, C., and Torres-Jimenez, J. (2016). Metaheuristic algorithms for building covering arrays: A review. *Revista Facultad de Ingeniería*, 25(43):31–45.

[52] Torres-Jimenez, J. and Izquierdo-Marquez, I. (2013). Survey of covering arrays. In *15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2013, Timisoara, Romania, September 23-26, 2013*, pages 20–27.

[53] Torres-Jimenez, J. and Izquierdo-Marquez, I. (2016). Construction of non-isomorphic covering arrays. *Discrete Math. Algorithm. Appl.*, 08(02):1650033.

[54] Torres-Jimenez, J., Izquierdo-Marquez, I., Gonzalez-Gomez, A., and Avila-George, H. (2015a). *A branch & bound algorithm to derive a direct construction for binary covering arrays*, pages 158–177. Springer International Publishing, Cham.

[55] Torres-Jimenez, J., Izquierdo-Marquez, I., Kacker, R., and Kuhn, D. R. (2015b). Tower of covering arrays. *Discrete Applied Mathematics*, 190-191:141–146.

[56] Torres-Jimenez, J., Rangel-Valdez, N., Gonzalez-Hernandez, A. L., and Avila-George, H. (2011). Construction of logarithm tables for galois fields. *International Journal of Mathematical Education in Science and Technology*, 42(1):91–102.

[57] Torres-Jimenez, J. and Rodriguez-Cristerna, A. (2017). Metaheuristic post-optimization of the nist repository of covering arrays. *CAAI Transactions on Intelligence Technology*, 2(1):31 – 38.

[58] Torres-Jimenez, J. and Rodriguez-Tello, E. (2010). Simulated annealing for constructing binary covering arrays of variable strength. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*. IEEE.

[59] Torres-Jimenez, J. and Rodriguez-Tello, E. (2012). New bounds for binary covering arrays using simulated annealing. *Information Sciences*, 185(1):137–152.

[60] Tzanakis, G., Moura, L., Panario, D., and Stevens, B. (2016). Constructing new covering arrays from LFSR sequences over finite fields. *Discrete Mathematics*, 339(3):1158–1171.

[61] Walker, R. A. and Colbourn, C. J. (2009). Tabu search for covering arrays using permutation vectors. *Journal of Statistical Planning and Inference*, 139(1):69–80.

[62] Wu, Q. and Hao, J. (2015). A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709.

[63] Yan, J. and Zhang, J. (2006). Backtracking algorithms and search heuristics to generate test suites for combinatorial testing. In *30th Annual International Computer Software and Applications Conference, COMPSAC 2006, Chicago, Illinois, USA, September 17-21, 2006. Volume 1*. Institute of Electrical & Electronics Engineers (IEEE).