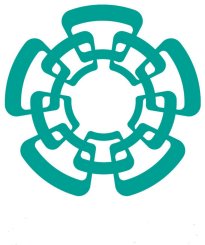


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Cinvestav Tamaulipas

Construcción de *covering arrays* en el dominio de grafos

Tesis que presenta:

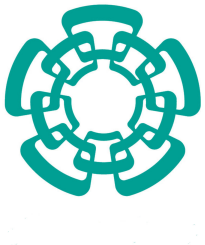
José Carlos Pérez Torres

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



Center for Research and Advanced Studies
of the National Polytechnic Institute

Cinvestav Tamaulipas

Covering arrays construction in the graph domain

Thesis by:

José Carlos Pérez Torres

as the fulfillment of the
requirement for the degree of:

**Master of Science
in Engineering and Computing
Technologies**

Thesis Director:
Dr. José Torres Jiménez

Cd. Victoria, Tamaulipas, México. October, 2017

The thesis of José Carlos Pérez Torres is approved by:

_____

Dr. Hector Hugo Avilés Arriaga

Dr. Ricardo Landa Becerra

Dr. José Torres Jiménez, Committe Chair

Cd. Victoria, Tamaulipas, México., October 6 2017

A mis padres, por todo el apoyo brindado.

Acknowledgements

- Special thanks to Samantha and my friends Gildardo, Daniel, Rebeca, Edi, Carlos, Raul, Oscar, Jordy, Vago, Hugo, and Melesio for the cheering and support.
- I also thank the administrative personnel at CINVESTAV Tamaulipas for their help during my stay.
- I thank CONACyT for the provided economic support which allowed me to concentrate in my studies and CINVESTAV-Tamaulipas for the opportunity to pursue graduate studies.
- I thank to my advisor Dr. José Torres-Jimenez, his patience and help in the development of this work.
- I also acknowledge "Xiuhcoatl"-CGSTIC of CINVESTAV and ABACUS-CINVESTAV, CONACYT grant EDOMEX-2011-COI-165873, for providing access to high-performance computing.
- This research was partially funded by the project CONACyT238469 - Métodos Exactos para Construir Covering Arrays.

Contents

List of Figures

List of Tables

List of Algorithms

Publications

- Perez-Torres Jose Carlos, Torres-Jimenez Jose. A graph based post-optimization approach for covering arrays. At the journal Quality and Reliability Engineering International 2017;0:1-9. Available at: https://doi.org/10.1002/qre.2176. Impact Factor: 1.366

Resumen

Construcción de *covering arrays* en el dominio de grafos

por

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

Los covering arrays (CAs) son objetos combinatorios con amplias aplicaciones prácticas, como la calibración de parámetros en experimentos, las pruebas de software y hardware y en la implementación de códigos surjectivos. Un covering array $CA(N; t, k, v)$ es denotado por la tupla de cuatro enteros N, t, k, y v con $t \leq k$ y representado por una matriz de $N \times k$ elementos en la cual cada subarreglo de tamaño $N \times t$ contiene todas las t-tuplas formadas con v símbolos al menos una vez, donde N representa la cantidad de filas del CA, k la cantidad de columnas del CA y t representa la fuerza de cobertura del CA.

Este documento presenta un nuevo paradigma para la construcción de CAs a través de la transformación de ellos al dominio de grafos. Se propone la representación *cobertura en nodos* que embebe la cobertura provista por un CA en los nodos de un grafo. Esta representación ofrece la posibilidad de construir CAs con el mínimo número de filas a través de la solución del problema NP-duro de encontrar la cobertura con cliques de todos los nodos de un grafo con el mínimo número de cliques. Se muestran tres enfoques distintos para resolver este problema: a) un algoritmo exacto, b) un algoritmo avaro, y c) un algoritmo metaheurístico. Además se presenta un algoritmo de post-optimización de CAs basado en grafos (GBPO) para reducir el numero de filas de un covering array o el número de interacciones que faltan en un covering array parcial.

Los resultados muestran que el algoritmo exacto construye CAs óptimos que igualan las mejores cotas conocidas para 7 de 7 instancias probadas, el algoritmo avaro iguala 18 instancias y mejora 77 instancias de las 95 probadas respecto a una version del algoritmo avaro *In-parameter-order* (IPOG-F), el algoritmo metaheurístico construye 17 instancias que igualan a las mejores cotas conocidas y 4 instancias que son mejoradas del total de 33 instancias, y finalmente el algoritmo GBPO mejora 560 de 560 CAs del repositorio de instancias generado por el algoritmo IPOG-F.

Abstract

Covering arrays construction in the graph domain

by

José Carlos Pérez Torres
Cinvestav Tamaulipas
Center for Research and Advanced Studies of the National Polytechnic Institute, 2017
Dr. José Torres Jiménez, Advisor

Covering arrays (CAs) are combinatorial objects that have wide practical applications such as the hardware and software testing, the parameter tuning for experimental design, and the implementation of surjective codes. A covering array is denoted as $CA(N; t, k, v)$, with a 4-tuple of integers N, t, k, and v and represented by an $N \times k$ array such that every $N \times t$ sub-array contains all t-tuples from v symbols at least once, where N denotes the number of rows of the CA, k denotes the number of columns of the CA and t represents the strength of the CA.

This document presents a novel paradigm to construct CAs through a mapping to the graph domain. We propose the *coverage in nodes* representation that embeds the coverage provided by a CA in the nodes of a graph. This representation allows to construct CAs with the minimal number of rows through the solution of the minimum clique covering problem (MCCP) classified as an NP-hard problem. We present three approaches for solving of MCCP: a) an exact algorithm, b) a greedy algorithm, and c) a metaheuristic algorithm. Furthermore, a graph-based post-optimization (GBPO) for CAs is presented. GBPO reduces the number of rows of a CA or the number of missing interactions of a partial CA.

Results show that for the experimentation: of 7 CAs for the exact algorithm, 7 CAs matched the best-known upper bounds for CAs; of 95 CAs for the greedy algorithm: i) 18 CAs matched versus a state of the art greedy algorithm, and ii) 77 instances improved the upper bounds of the In-parameter-order (IPOG-F) algorithm; of 77 CAs for the metaheuristic algorithm: i) 17 CAs matched the best-known upper bounds and, ii) 4 CAs of the best-known upper bounds; and for the experimentation of 560 CAs for the GBPO algorithm: 560 CAs were improved of the IPOG-F instances repository.

Nomenclature

CA	Covering array
CAs	Covering arrays
CAN	Covering array number
CAK	Covering array column
CACP	Covering array construction problem
CN	Coverage in nodes
MCA	Mixed covering array
MCAN	Mixed covering array number
MCP	Maximum clique problem
MCCP	Minimum clique covering problem

Definitions

Definition 1 (Covering array (CA))

Given the positive integers N, t, k and v a covering array CA$(N; t, k, v)$ is a matrix A of size $N \cdot k$, where $A = (a_{i,j})$ for $0 \leq i \leq N-1$ and $0 \leq j \leq k-1$ with values over the set $\mathbb{Z}_v = \{0, \ldots, v-1\}$ stands the property that for each t distinct columns $0 \leq c_0 < \ldots < c_{t-1} \leq k-1$ and each t-tuple $(x_0, \ldots, x_{t-1}) \in \mathbb{Z}_v^t$ at least one row exist such that $x_i = a_{r,c_i}$ for $0 \leq i \leq t-1$.

Definition 2 (Covering array number (CAN))

Given t, k and v parameters the covering array number CAN(t, k, v) is the CA$(N; t, k, v)$ such that N is the minimum possible.

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

Definition 3 (Covering array column (CAK))

Given t, N and v parameters the covering array column CAK(t, N, v) is the CA$(N; t, k, v)$ such that k is the maximum possible.

$$\text{CAK}(t, N, v) = max(k)\{k \mid \exists\ \text{CA}(N; t, k, v)\}$$

Definition 4 (Mixed covering array (MCA))

Given t, k and $(v_0, \ldots, v_{k-1})$ parameters, the MCA$(N; t, k, (v_0, \ldots, v_{k-1})$ is a matrix of size $N \cdot k$, where the values for column i are over the set $\mathbb{Z}_{v_i} = \{0, \ldots, v_i - 1\}$ and for each t distinct columns $i_0, \ldots, i_{t-1}$ every t-tuple in $\prod_{j=0}^{t-1} \{0, \ldots, v_{i_j} - 1\}$ appear at least in one row.

Definition 5 (Mixed covering array number (MCAN))

Given t, k and $(v_0, \ldots, v_{k-1})$ parameters the mixed covering array number MCAN$(t, k, (v_0, \ldots, v_{k-1}))$ is the MCA$(N; t, k, (v_0, \ldots, v_{k-1}))$ such that N is the minimum possible.

$$\text{MCAN}(t, k, (v_0, \ldots, v_{k-1})) = min(N)\{N \mid \exists\ \text{MCA}(N; t, k, (v_0, \ldots, v_{k-1}))\}$$

Definition 6 (Isomorphic CAs)

A CA is isomorphic of A if its possible to turn it into A by a combination of a row permutation, a column permutation and a symbol permutation of a subset of columns (defined in [58]).

Definition 7 (Clique)

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a clique is a set $\mathcal{C} \subseteq \mathcal{V}$ such that, for every distinct nodes $u, v \in \mathcal{C}$ the edge (u, v) exists in $\mathcal{E}$. A clique is a complete subgraph of $\mathcal{G}$.

Definition 8 (Maximal clique)

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a maximal clique is a clique that cannot be extended with another vertex of $\mathcal{V}$.

Definition 9 (Maximum clique)

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a maximum clique is the clique with the largest amount of vertices. Therefore a maximum clique is maximal but, not all maximal cliques are maximum.

Definition 10 (Maximum clique problem (MCP))

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ find the largest complete subgraph $\mathcal{C}$ of $\mathcal{G}$. MCP is a problem NP-complete, probed by mapping to satisfiability problem (demonstration in [54]).

1 Introduction

This chapter presents the thesis problem, main objective and specific objectives of the thesis. Section 1.1 presents an introduction to covering arrays. Section 1.2 describes the practical applications of covering arrays. Section 1.3 presents the advantages of solving the covering array construction problem (CACP) in the graph domain. Section 1.4 shows a study that illustrate the impact of inadequate software testing in US and how the strength of a CA is related with the detection rate of software faults. In Section 1.5 the thesis problem is stated, followed by the research hypothesis, the main objective, and the particular objectives of the thesis.

1.1 An introduction to covering arrays

A covering array CA$(N; t, k, v)$, denoted with a 4-tuple of integers, is used to construct a set of test cases to verify components of software or hardware. Each one of the N rows of the CA represents a test case and is composed by k values that represent every factor that integrate the component. These factors have values over the set $\mathbb{Z}_v = \{0, \ldots, v-1\}$. The strength parameter represented

as t indicates that all combinations of t factors occurs at least once in the CA. The covering array ensures that each of the v^t interactions of each one of the $\binom{k}{t}$ possible t-subsets is covered in, at least, one row.

Formally, a covering array is denoted as $CA(N; t, k, v)$, where N represents the number of rows of the CA, k represents the number of columns, v is the number of symbols of the alphabet or vocabulary for the CA and it delimits the possible values for the cells of the CA to the set $\mathbb{Z}_v = \{0, \ldots, v-1\}$, and t represents the strength of the CA.

For example, the Figure 1.1 shows the covering array CA$(11; 2, 5, 3)$. This covering array has $N = 11$ rows and $k = 5$ columns, its alphabet is $v = 3$, and its strength is $t = 2$. Each one of the $\binom{5}{2} = 10$ subarrays of two distinct columns covers every member of the $\mathbb{Z}_3^2$ at least once. Set $\mathbb{Z}_3^2$ is equal to

$$\begin{aligned}\mathbb{Z}_3^2 &= \{0,1,2\}^2 = \{0,1,2\} \times \{0,1,2\} \\ &= \{(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)\}.\end{aligned}$$

↓			↓	
1	1	0	2	0
2	2	1	1	0
0	1	2	1	2
2	0	0	0	2
0	0	0	1	1
2	1	1	2	1
0	2	0	2	2
1	0	1	1	2
0	1	1	0	0
2	0	2	2	0
1	2	2	0	1

Figure 1.1: The covering array CA$(11; 2, 5, 3)$. In the columns marked down with an arrow the first occurrence of each tuple of the set $\{0,1,2\}^2$ is colored in red.

Consider the following real application: A Webmaster wants to validate the successful access to his new web service, taking into account the next information about the requesting user: operating system, web browser, broadband speed, time at which a request was sent, and the location from where the user sent it. Considering that each of those parameters has three possible values (shown in Table 1.1). The total number of test cases of an exhaustive test that should be executed in order to assess the behavior of the web service would be 243, which means testing every single combination of the possible parameters (3^5).

Operating System	Web browser	Broadband speed	Time of request	Location
Windows	Firefox	512 Kbps	2 AM	China
OS X	Opera	1 Mbps	10 AM	Russia
Linux	Chrome	10 Mbps	6 PM	USA

Table 1.1: Possible values for the parameters to be taken into account for a real application testing.

In a small example such as this, the number of total test cases might not be impactful, but if the number of parameters taken into account or the range of values for each parameter increase, the number of test cases to assess a correct behavior would grow exponentially. For small test cases, such as testing the success or failure of a request, that can be evaluated in short time the cost of executing every single case might not be meaningful, but when testing cases are expensive, reducing the amount of tests needed to assure a correct behavior or detect failures greatly reduces the costs associated to the test, both monetary and time wise.

For example, for the previously mentioned application, using combinatorial testing we can construct a CA$(11; 2, 5, 3)$ with 11 test cases that validates every interaction between two parameters or a CA$(33; 3, 5, 3)$ with 33 test cases that validates every interaction between three parameters. The CA with $t = 2$ represents $< 5\%$ of the test cases needed by the exhaustive test. Meanwhile, the CA with $t = 3$ represents $< 15\%$ of the test cases needed by the exhaustive test. Studies shown that

validating interactions between two parameters detect $> 70\%$ of the software failures and validating interactions between three parameters detect $> 90\%$ of those failures. In the Table 1.2 are shown the test cases needed to validate every interaction between two parameters.

No.	Operating System	Web browser	Broadband speed	Time of request	Location
1	OS X	Opera	512 Kbps	6 PM	China
2	Linux	Chrome	1 Mbps	10 AM	China
3	Windows	Opera	10 Mbps	10 AM	USA
4	Linux	Firefox	512 Kbps	2 AM	USA
5	Windows	Firefox	512 Kbps	10 AM	Russia
6	Linux	Opera	1 Mbps	6 PM	Russia
7	Windows	Chrome	512 Kbps	6 PM	USA
8	OS X	Firefox	1 Mbps	10 AM	USA
9	Windows	Opera	1 Mbps	2 AM	China
10	Linux	Firefox	10 Mbps	6 PM	China
11	OS X	Chrome	10 Mbps	2 AM	Russia

Table 1.2: A set of test cases built from the covering array CA$(11; 2, 5, 3)$.

1.2 Background

CAs are combinatorial objects effective in the processes of: software testing [18], hardware testing [4, 56], experimental designs [20], detection of trojans in hardware [27], and in the implementation of surjective codes [47].

Nowadays, several approaches have been proposed to solve the CACP, such as: exact, greedy, algebraic, and metaheuristic algorithms, see [57].

1.3 Justification

Despite the diversity in the proposed approaches to solve CACP, there is not, to our best knowledge, any method that solves CACP in the graph domain. The proposed approaches have used as representation a matrix of integers [15] or a vector of integers [11] but not a graph representation. Despite a graph could be implemented as a matrix through an adjacency matrix, the proposed approaches that use a matrix of integers do not have the same purpose.

The solution of CACP in the graph domain allows us to take advantage of the state-of-the-art algorithms that are related to graph problems such as the maximum clique.

Using a graph representation to solve CACP also allows us to save time spent in process of validation of the coverage of a CA, due to the fact that the solution to the problem mapped to the graph domain keeps tracking of the interactions that are already covered.

1.4 Impact

In 2003, the National Institute of Standards and Technology (NIST) published that inadequate software testing costs to the US economy $59.5 billion per year, even though that 50 to 80% of the cost of the total software development is for software testing process [30]. Software testing based on combinatorial objects implies that most failures in the software are caused by interaction between few parameters.

In 2009 Kuhn *et al.* [49] showed the relation among the detection of software faults and the strength of the CA used to do the tests, which is the number of parameters involved in an interaction. Others studies in [31, 32, 33] pointed out that when the strength of the CA is from 4 to 6, the fault detection rate reaches 100%.

1.5 Thesis problem

1.5.1 Problem statement

Given the parameters t, k, and v; the CACP consists in the construction of a CA$(N; t, k, v)$ where the number of rows N is minimal. The proposed representation of the CA characteristics has its own problem statement and is discussed in the Chapter 3.

1.5.2 Hypothesis

It is possible to construct CAs of strength $t = 2$ in the graph domain that match or reduce the number of rows of the best reported CAs.

1.5.3 Main objective

Constructing CAs of strength ($t = 2$) in the graph domain one row at a time with equal or lesser number of rows than those reported by the state-of-the-art algorithms.

1.5.4 Specific objectives

Three specific objectives are stated in order to fulfill the main objective.

1 Development of an algorithm to map a CA to the graph domain and to transform the obtained solution in the graph domain to a CA.

2 Development of algorithms to solve the CACP in the graph domain.

3 Development of a post-optimization algorithm to improve the quality of previously constructed solutions.

1.6 Thesis contents

The thesis document is organized in six chapters; the following is a brief summary of each chapter content:

- *Chapter 1: Introduction*. The first chapter describes practical applications of the covering arrays, the benefits of using combinatorial testing and the problem statement of this thesis.

- *Chapter 2: State of the art of the construction of covering arrays*. This chapter presents several methods already developed to solve CACP. These methods are classified in: exact, greedy, metaheuristic, and algebraic. It also presents algorithms to post-optimize covering arrays.

- *Chapter 3: A graph representation for covering arrays*. This chapter describes the mapping of a covering array instance to a graph, so the equivalent problem can be solved in the graph domain.

- *Chapter 4: Methodology to solve the CACP in the graph domain*. This chapter presents the methodology followed to solve the graph-based problem to construct a covering array and the algorithms developed to perform the methodology.

- *Chapter 5: Experimentation and results*. This chapter describes the computational experimentation done and the relevant results obtained from the developed approaches of solution.

- *Chapter 6: Conclusions and future works*. The last chapter presents a summary of the contributions of this thesis and some topics for future works.

1.7 Summary

This chapter introduced the thesis generals and the definition of a covering array, described the practical applications of covering arrays in the hardware and software testing and the benefits of using a graph representation for the construction of covering arrays. Also is presented an study that analyzes the impact of testing inside the complete software development process. The next chapter will present the state-of-the-art of methods that construct covering arrays.

2

State of the art of the construction of covering arrays

In this chapter we present distinct state-of-the-art approaches to solve the CACP. These approaches are grouped by the construction process used: exact methods (Section 2.1), greedy algorithms (Section 2.2), metaheuristic algorithms (Section 2.3), algebraic methods (Section 2.4), and post-optimization methods (Section 2.5). The last subsection reviews problems related to CAs that are confronted using some graph representation.

In the Figure 2.1, we provide a brief summary of the construction methods, the following sections will explain in detail examples of construction of the methods that are presented in these figure.

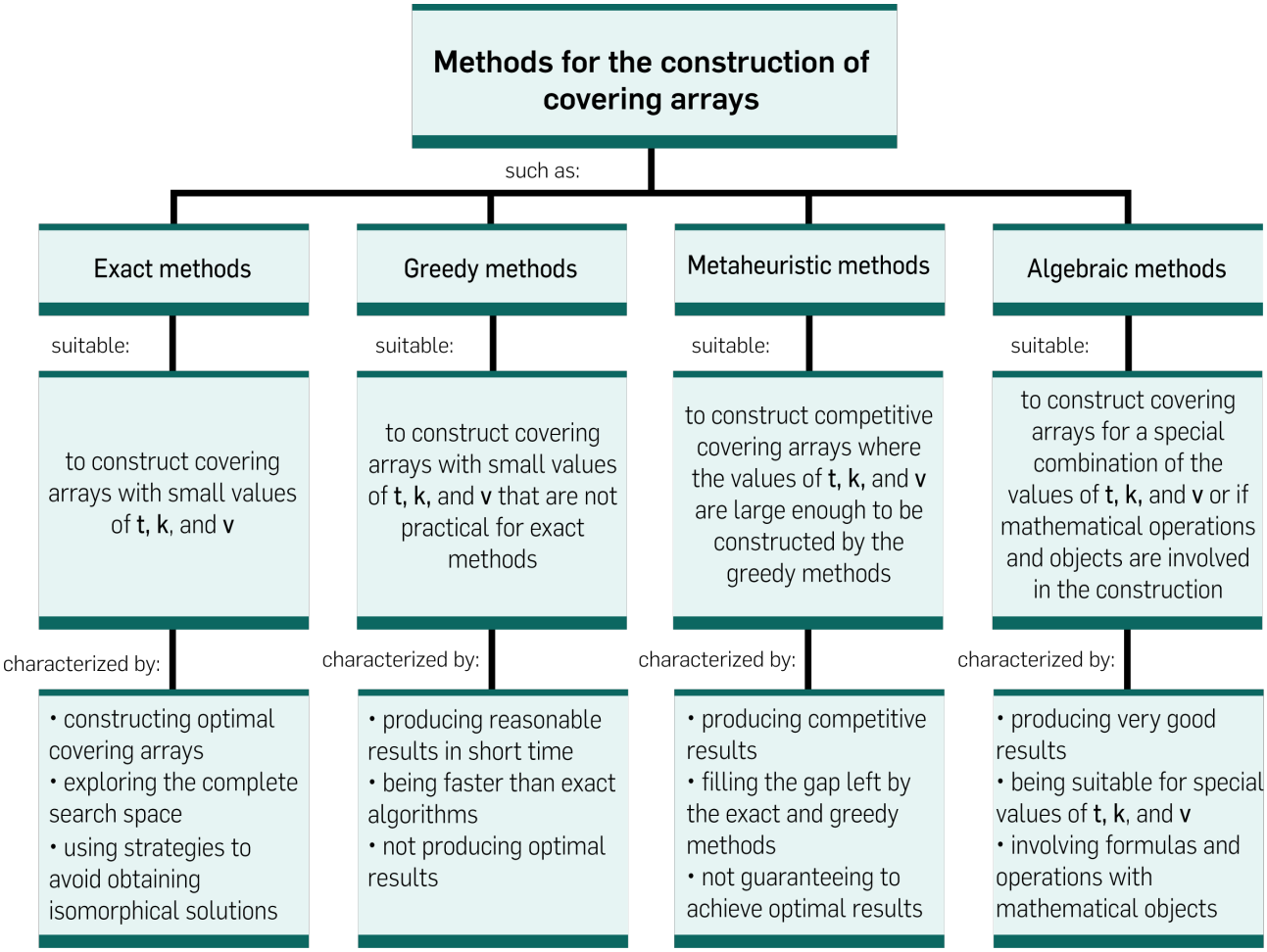


Figure 2.1: Summary of construction methods for covering arrays.

2.1 Exact methods to construct CAs

Exact methods have the characteristic that given the values t, k, and v they construct a CA$(N; t, k, v)$ where the number of rows N is minimal. Despite the use of strategies to speed up the construction process, they result practical only for constructing small covering arrays, the cases where $v \leq 6$, $k \leq 10$, and $t = 2$ are the only ones that result practical [29].

2.1.1 Integer programming

In 2002, Williams and Probert [65] presented a formulation of CACP using {0,1} integer programming. A vector x represents the set of all possible rows or configurations, which are $|x| = v^k$; a vector y represents all the interactions required, which are $|y| = \binom{k}{t}v^t$; and a matrix $A = a_{ij}$ of size $|y| \times |x|$ to identify which rows cover each one of all interactions, $a_{ij} = 1$ when the interaction i is covered by the row j, otherwise $a_{i,j} = 0$. The coverage is mapped into a set of $|y|$ constraints of the form $\sum_{j=1}^{|x|} a_{i,j}x_j \geq 1$ for $1 \leq i \leq |y|$. $x_j = 1$ when the row j is taken for the solution, otherwise $x_j = 0$. So, the complete formulation consists in: minimize $\sum_{j=1}^{|x|} x_j$ subject to the constraints $\sum_{j=1}^{|x|} a_{i,j}x_j \geq 1$ for $1 \leq i \leq |y|$. This formulation allows a linear programming solver to construct optimal covering arrays but only works for small cases of t, k, and v, where the number of variables is less than 100.

2.1.2 The automatic generator EXACT

In 2006, Yan and Zhang [67] introduced the exhaustive search of combinatorial test suites generator (EXACT). This generator assigns values to elements of a CA whenever it is possible and backtracks when is detected that it is not possible to fulfill the CA avoiding the exploration of isomorphic solutions by considering only the arrays lexicographically sorted in rows and columns.

2.1.3 Constraint programming

Later on, Hnich *et al.* [3] proposed 4 matrix models of constraint programming to solve CACP: naive matrix model, alternative matrix model, integrated matrix model, and weakened matrix model. The naive matrix model consists of a matrix X of dimensions $N \times k$ with entries $x_{r,i}, 1 \leq r \leq N, 1 \leq i \leq k$ such that $x_{r,i} = m$ if the value of parameter i in test vector r is m. The alternative matrix model of size $N \times \binom{k}{t}$ of compounds variables $y_{r,j}$ where each variable represents a tuple $(x_{r,l_1}, x_{r,l_2}, \ldots, x_{r,l_t})$ of the naive matrix model. The integrated matrix model links the previous models by channeling constraints that associate compounds variables of the alternative matrix model with their t corresponding variables in the naive matrix model. The weakened matrix model is a satisfiability (SAT) encoding with several constraints omitted.

2.1.4 SAT encodings

For $t = 2$, Lopez-Escogido *et al.* [38] proposed a codification for SAT problem to map CACP to this domain and take advantage of existing SAT local search methods. For each element $m_{i,j}$ of matrix M associated with an instance, v variables are introduced $m_{i,j,x}$, $0 \leq x < v$. The model of transformation uses two sets of clauses in conjunction (z_1 and z_2) to ensure that each element in M takes exactly one value of the set $\{0, 1, \ldots, v-1\}$ and uses another set of clauses (z_3) non-conjunctives to ensure that interactions of strength 2 are fulfilled.

$$z_1 = \bigwedge_{i=0}^{N-1} \Big(\bigvee_{\forall j,x | 0 \leq j < k, 0 \leq x < v} m_{ijx} \Big) \tag{2.1}$$

$$z_2 = \bigwedge_{i=0}^{N-1} \Big(\bigvee_{\forall j,x,y | 0 \leq j < k, 0 \leq x < y < v} (\overline{m_{i,j,x}} \vee \overline{m_{i,j,y}}) \Big) \tag{2.2}$$

$$z_2 = \bigwedge_{i=0}^{N-1} \Big(\bigwedge_{\forall j,l | 0 \leq j < l < k} \Big(\bigvee_{\forall i | 0 \leq i < N} (m_{i,j,x} \vee m_{i,l,y}) \Big) \Big) \tag{2.3}$$

2.1.5 Backtracking algorithm for binary CAs

Later on, Bracho-Rios *et al.* [5] introduced a branch and bound algorithm to construct binary covering arrays of variable strength. This algorithm constructs one column at a time using all possible columns with $\lfloor \frac{N}{2} \rfloor$ zeros. If the new column forms a partial CA and is bigger than the previous column then it is inserted, otherwise, it tries the next element. Whenever it is no longer possible to insert a new column the algorithm backtracks to a previous step. They also provide two techniques for improving the search efficiency: symmetry breaking and partial verification.

2.1.6 Non-isomorphic CAs

In 2016, Torres-Jimenez *et al.* [58] developed the construction of all the non-isomorphic CAs for values N, t, k, and v. A CA B is considered isomorphic to a CA A if it is constructed by a combination of row permutation, a column permutation, and a symbol permutation of a subset of columns of A. This construction extends a subarray from 0 to k columns, one column at a time, validating that the new column is a CA with minimum lexicographical order with the previous columns. Whenever the process reaches k columns it reports a new non-isomorphic CA that is optimal.

2.2 Greedy methods to construct CAs

Where combinations of values t, k, v are impractical for exact methods, CACP could be solved by a greedy method producing a reasonable solution in short time.

2.2.1 The AETG system

In 1994, Cohen *et al.* [14] presented a test case generator with variables defined by the user known as *automatic efficient test generator* (AETG). This method starts with an empty test case and generates one test at a time until all interactions of size t are covered. A test case is generated

through probabilistic method, building a row that has a higher probability to cover the largest amount of missing tuples taking into account the current covered tuples. The authors say that this method can construct CA and MCA but only for strength $t = 2, 3$.

2.2.2 Deterministic density algorithm

In 2007, Bryce and Colbourn [6] introduced *deterministic density algorithm* (DDA), a test case generator for strength ($t = 2$) that builds one row at a time. Values of a row are dynamically fixed one at a time in an order based on density. New rows are continually added until all interactions have been covered. DDA fixes one value for each factor and updates the densities for those factors to construct a test case.

The factor to be fixed is the one with the largest density among the non-fixed factors. The density $\delta_{i,j}$ for factors i and j is computed as follows: a) if both factors have more than one level $\delta_{i,j} = (r_{i,j}/l_{max}^2)^2$, where l_{max} is the largest cardinality of the two factors, and $r_{i,j}$ is the number of missing interactions among factors i and j; b) if only one factor has one level left $\delta_{i,j} = (r_{i,j}/l_{max})^2$; c) if both factors have exactly one level and a new pair is covered $\delta_{i,j} = 1.0$; d) If both factors have exactly one level and no new pair is covered $\delta_{i,j} = 0$. The density of the factor i is the summation of the local densities over each factor $j \neq i$. After the selection of a factor, DDA assigns a level to it; this level is the one with the higher level density. These level densities are computed for a specific level v_i to an individual factor k_j as follows: a) if k_j has more than one level involved in uncovered pairs with v_i, $\delta = (r_{i,j}/l_{max})$; b) if k_j has only one level involved in uncovered pairs and a new pair is covered, $\delta = 1.0$; c) if k_j has only one level involved in uncovered pairs an no new pairs are covered, $\delta = 0$. If there exists a tie in fixing factors or levels the most simple tie-breaking rule is to choose the smaller factor or level in lexicographic order.

2.2.3 In-Parameter-Order algorithm

From 2006 to 2008, Lei and Tai [35] developed In-Parameter-Order (IPO) algorithm to construct all the interactions of size t of the k available factors. The main feature of this construction is that the CA performs two types of expansion: expansion in rows; and expansion in columns, until the CA reaches the number of desired columns. Two variants of IPO algorithm were developed: in 2007 Lei *et al.* [34] proposed IPOG, a version with a generalization for higher strengths; and Forbes *et al.* [15] proposed an improvement for IPO. This three algorithms share the strategy of using a CA of $k-1$ factors to build a CA with k factors. This recursion starts with the listing of t-tuples of the set $\{0,\ldots,v-1\}$ for $k=t$ and keep adding one column at a time. To add factors, two types of growth exist:

- Horizontal growth. An additional column is added corresponding to the new factor. The cells of the new column are filled in a way that new tuples of the expanded CA are covered.
- Vertical growth. New rows are added to the matrix to cover the missing tuples after the horizontal growth.

In the horizontal growth the values for the new column are filled bottom-up and the value chosen for every cell is the one that covers the largest amount of missing tuples. The vertical growth performs the insertion of missing interactions in existent rows, if it is not possible to add a missing interaction in an existent row, a new row is inserted with the current missing interaction.

2.2.4 Intersection residual pair set strategy

In 2008, Younis *et al.* [68] presented the Intersection Residual Pair Set Strategy (IRPS), a greedy algorithm to generate MCAs and CAs of strength $t=2$. For a test set with k factors, a PI list contains $k-1$ linked lists, where the i-th linked list contains nodes equal to the number of values defined for the i factor and for each of them an array of linked lists that represents the interactions

among the i factor with the $i < j < k$ remaining factors combined with the number of values defined for each one of the j factors. The last factor does not have linked lists.To generate a test case the authors define the $weight$ as the interactions covered by a test case and $wmax$ as the maximum interactions covered by a test case ($wmax = k(k-1)/2$ at the beginning of the test generation).

The algorithm searches for a test case that cover $wmax$ interactions, if this case does not exist $wmax$ is decremented. To perform the searching of such that case, IRPS begin the search in the nodes of the first factor that still has nodes and computes the interactions covered by a recursive intersection of the available elements of it linked list down to the $k-1$ factor that only have nodes.

This process is repeated until the PI list is empty.

2.2.5 Coverage inheritance

In 2011, Calvagna and Gargantini [8] presented a greedy algorithm to generate MCAs of any strength. This algorithm adds one new column at a time until all needed columns are covered, and it is based on *coverage inheritance*. When $t < k$, the algorithm starts with an MCA of strength t for t parameters by listing all the combinations of the first t parameters (parameters must be in descending order). If v_i is the alphabet for parameter i, then $v_i \geq v_j$ for all $i \leq j$. To append the missing $k-t$ parameters a column of the CA is copied, and all the elements that are not in range for the alphabet of the new columns are settled to null to indicate that they can be reassigned with another value. Since the new column denoted as j is a copy of a previous one i, all the interactions of size t are covered except the ones where i and j are involved, this coverage guarantee is known as *coverage inheritance*. The null elements are used to cover the missing interactions. New rows are added to fulfill the coverage when the fixing of values for the null elements are not enough to cover the missing interactions.

2.3 Metaheuristic methods to construct CAs

Metaheuristic methods cannot guarantee to find the minimum number of rows for a CA construction, but the obtained results through these methods have been very competitive. As the parameters t, k, v grow, the greedy methods begin to fall off in their solution's quality and the exact algorithms requires an endlessly amount of time to construct an optimal solution. That gives the opportunity to the metaheuristic algorithms to fill this gap.

2.3.1 Genetic algorithms

One of the first techniques used were the genetic algorithms (GA) [19], proposed by Holland in 1975, GA simulates the biological evolution of a population through mutations, recombination and individual selection.

In 2001, Stardom [55] presented a GA for CACP. He uses a matrix representation where the recombination is performed selecting a set ε of coordinates of the father and the remaining of the second father. The coordinates in the set ε are constituted by the first i rows, the first j columns or a block of the union of i and j. Mutation is applied after recombination and consists in the change of an element of the matrix randomly.

In 2004 Shiba *et al.* [53] proposed a GA that is a modification of AETG [14]. In this algorithm, every chromosome represents a test case and is evaluated according to the new tuples that provide missing interactions. An initial population of m individuals is randomly generated and use a tournament selection. The mutation replaces one element of the chromosome for a random value of the alphabet.

2.3.2 Tabu search

Tabu search (TS) is a technique proposed by Glover [16]. TS uses a list of last changes to avoid returning to neighborhoods recently explored, which prevents the search from being trapped in local optima.

In 2004, Nurmela [45] used TS to construct CAs. TS starts with a random matrix and using missing tuples it forms a valid CA as the objective function. TS selects missing tuples and puts them in one row that needs just one modification to generate the tuple. The selected row to perform the action is the row that produces the less cost; otherwise, it is randomly selected.

2.3.3 Simulated annealing

Simulated Annealing (SA) algorithms simulate the slow cooling process of metals [26]. The implementation of SA requires at least three parameters: initial temperature T_0, final temperature T_f and the cooling rate α $(0 < \alpha < 1)$.

In 2008, Covarrubias-Flores [13] proposed in her Ms thesis an SA to construct CAs of several strengths and $v = 2$. An initial solution M is generated randomly balancing a number of 0s and 1s. The evaluation function is the number of missing tuples of matrix M.

Later on, in 2010 Martinez-Pena *et al.* [42], presented an SA for CAs construction with $v = 3$ using trinomial coefficients (TC). The initial solution A is built selecting $t + k$ trinomial coefficients.

In 2010, Torres-Jimenez *et al.* [63] presented a SA to construct binary covering arrays of variable strength. The algorithm incorporates a heuristic to generate good quality initial solutions and a compound neighborhood that combines two neighborhood functions. This algorithm improves 22 upper bounds.

Afterwards, Avila-George *et al.* [1, 2] presented several parallel implementations of SA denominated *independent search*, *semi-independent search* and *cooperative search*. Independent search performs multiple executions of an SA using the same initial solution and selecting the best

solution of all the executions. Semi-independent search divides the Markov chains in a balanced way, the processors interchange intermediate solutions and perform the search separately. Cooperative search keeps a best so far solution that is updated whenever a processor ends its own Markov chain, every processor continue without waiting updates of the best so far, so when the process continues after an ending Markov chain, the processors could start with different initial solutions and avoid local optima.

2.3.4 Particle swarm optimization

Another technique used is the particle swarm optimization (PSO) proposed by Kennedy and Eberhart [24] which simulates the movement of a bird flock (swarm) with two vectors that represent velocity and position of each particle of the swarm.

In 2015, Wu *et al.* [66] presented several extended (PSO) algorithms to construct CAs, a set-based PSO and a discrete PSO (DPSO). Two auxiliary strategies to enhance the performance of PSO were presented: additional evaluation of the global best and a particle reinitialization. A comparative of PSO variants is also presented showing the strengths and weaknesses of the proposed variants of PSO.

Also in 2015, Mahmoud *et al.* [40] presented a fuzzy self-adaptive PSO that integrates a Mamdani-type fuzzy inference system (FIS) to adapt the controllers of the PSO performance. Three different FISs are built to monitor PSO performance and adjust the parameters for improving the efficiency of the algorithm.

2.4 Algebraic methods to construct CAs

The algebraic methods have the characteristic that during the construction several formulas, mathematical operations, and mathematical objects (such as vectors, finite fields, or CAs) are used to construct CAs.

2.4.1 Orthogonal arrays

In 1952, Bush [7] introduced a construction method for orthogonal arrays. An orthogonal array is denoted by $OA(v^t; t, v+1, v)$, and exists when v is a prime power number where α denotes the power. This construction uses the elements of the Galois fields $GF(v^\alpha)$. If we denote each element in $GF(v^\alpha)$ as e_i for $0 \leq i \leq v-1$, the Bush construction performs in this way:

1. Generate a matrix M with $(v^\alpha)^t$ rows and $(v^\alpha)+1$ columns.

2. Label the first v columns of M with the elements $e_i \in GF(v^\alpha)$.

3. Label each row of M with a polynomial $y_j(x)$.

4. Each value of a cell $m_{ji} \in M$ where $0 \leq j \leq (v^\alpha)^t - 1$ and $0 \leq i \leq (v^\alpha) - 1$ is the polynomial evaluation of $y_j(e_i)$ using the Galois field $GF(v^\alpha)$.

5. Each value of a cell $m_{ji} \in M$ where $0 \leq j \leq (v^\alpha)^t - 1$ and $i = (v^\alpha)$ is the coefficient of higher degree of the polynomial $y_j(x)$, in other words the term a_{t-1} of $y_j(x)$.

For the evaluation of polynomials in Galois fields, Torres-Jimenez *et al.* [61] proposed a method to multiply polynomials using additions and subtractions of the logarithm and antilogarithm discrete tables of $GF(v)$. The multiplication is reduced to an access to a table and a simple addition.

For example, for $v = 3^1$ and $t = 2$ the Bush construction allow us to build the OA$(9; 2, 4, 3)$, the field $GF(3^1)$ is formed by the elements $0, 1$, and 2, which are the 3^1 different polynomials of $t = 2$ and degree $t - 1$, and will be denoted as e_0, e_1, and e_2 respectively. The polynomials y_i labels the rows and since $t = 2$, they are of the form $y_i = a_1 x + a_0$, where $a_1, a_0 \in GF(3^1)$. The labeling for the rows, the labeling for the columns and the OA itself are shown in Figure 2.2.

	e_0	e_1	e_2	∞
$y_0(x) = e_0x + e_0$	0	0	0	0
$y_1(x) = e_0x + e_1$	1	1	1	0
$y_2(x) = e_0x + e_2$	2	2	2	0
$y_3(x) = e_1x + e_0$	0	1	2	1
$y_4(x) = e_1x + e_1$	1	2	0	1
$y_5(x) = e_1x + e_2$	2	0	1	1
$y_6(x) = e_2x + e_0$	0	2	1	2
$y_7(x) = e_2x + e_1$	1	0	2	2
$y_8(x) = e_2x + e_2$	2	1	0	2

Figure 2.2: OA produced by the Bush construction for $v = 3$ and $t = 2$.

2.4.2 Case $t = 2$ and $v = 2$

For CAs with parameters $t = v = 2$, it exists an algorithm that given a number of rows N constructs a CA$(N; 2, k, 2)$ for a maximum possible value of k in polynomial time. This algorithm was proposed by Renyi for a pair number N, and generalized by Katona [23] and Spencer and Kleitman [28] for any N. Optimal number of columns is

$$k = \binom{N-1}{\lceil N/2 \rceil}$$

To construct the CA a matrix $N \times k$ is needed where the first row is filled with 0s and the remaining $N - 1$ rows with $\binom{N-1}{\lceil N/2 \rceil}$ combinations of $\lceil N/2 \rceil$ 1s and $N - 1 - \lceil N/2 \rceil$ 0s. Figure 2.3 shows a CA developed by this construction where the optimal k for $N = 6$ is 10.

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
0	0	0	0	0	0	0	0	0	0

Figure 2.3: The CA$(6; 2, 10, 2)$ produced by the Katona, Spencer and Kleitman construction for $k = 10$.

2.4.3 Constant weight vectors

In 1983, Tang and Woo [56] presented a method of construction based in constant weight vectors. The weight of a vector $s = (s_1, s_2, \ldots, s_k)$ is defined as the summation of its inputs $w = \sum_{i=1}^{k} s_i$. This method constructs a CA, concatenating a set of vectors with the same weight. For a binary case the construction is defined as: given k and t with $k > t$, a set of T binary vectors covers all the interactions of size t if T contains all the binary vectors of size w such that $w \equiv c \pmod{k-t+1}$ for a constant $c \in \{0, 1, \ldots, k-t\}$.

2.4.4 Roux-type constructions

In the PhD thesis of Roux [50] is introduced the Roux type construction derived of the expression

$$CAN(3, 2k, 2) \leq CAN(3, k, 2) + CAN(2, k, 2)$$

that indicates that a CA could be extended doubling the number of columns using two CAs A and B. Given $A = \mathsf{CA}(N_3; 3, k, 2)$ and $B = \mathsf{CA}(N_2; 3, 2k, 2)$. They are used to form a CA $C = \mathsf{CA}(N_3 + N_2; 3, 2k, 2)$, using the complement of B as follows:

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

Later on, Chateauneuf *et al.* [9], proposed three generalizations of Roux's type construction. The first two constructions require a combinatorial object called *ordered design*, denoted as $\mathsf{OD}(t, k, v)$ that represents a matrix of dimensions $\binom{v}{t} t! \times k$ with v symbols where every t distinct columns has each t-tuple exactly once. Three constructions could be summarized in these expressions:

- For $v \geq 3$, $CAN(2, 3k, v) \leq CAN(2, k, v) + v(v-1)$.

- If an OD$(2, m, u)$ exists, then for each $v \leq u$ and $2 \leq k \leq m$, stands that $CAN(2, m(m-1)k, v) \leq CAN(2, k, v) + 2u(u-1)$.

- $CAN(3, 2k, v) \leq CAN(2, k, v) + (v-1)CAN(2, k, v)$.

2.4.5 Power of a covering array

In 2004, Hartman [18] proposed the power of a CA, this construction consists in squaring the number of columns k of a CA $A = \text{CA}(N; t, k, v)$. To complete the powering, an OA $B = \text{OA}(k^2; 2, T(v,t)+1, k)$ is required. The alphabet of the OA is k because its elements will be used as an index for the columns of A. $T(v,t)$ denotes the Turán number that is the maximum number of edges in a v-partite graph with t nodes. The powering consists in creating the matrix of blocks C of k^2 columns and $T(v,t)+1$ rows. Each element of C contains one column of A. Let be $B[i,j]$ the entry (i,j) of B, and let be A_i the i-th column of A. Cell (i,j) of C is the column $B[i,j]$ of A. The result is $C = \text{CA}(N(T(v,t)+1); t, k^2, v)$.

2.4.6 Product of covering arrays of stregth two

In 2006, Colbourn *et al.* [12] presented the product of CAs of strength two, let A be a matrix representing a $CA(N; 2, k, v)$ and let B be a second matrix representing a $CA(M; 2, l, v)$. The product of A and B is the matrix $C = (c_{i,j})$ of dimensions $(N+M) \times kl$ such tat:

- $c_{i,(f-1)k+g} = a_{i,g}$ for $1 \leq i \leq N, 1 \leq f \leq l, 1 \leq g \leq k$.

- $c_{N+i,(f-1)k+g} = a_{i,g}$ for $1 \leq i \leq M, 1 \leq f \leq l, 1 \leq g \leq k$.

The construction is shown in the following matrix:

	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}
N rows	$\vdots$				$\vdots$				$\cdots$	$\vdots$			
	a_{N1}	a_{N2}	$\cdots$	a_{Nk}	a_{N1}	a_{N2}	$\cdots$	a_{Nk}	$\cdots$	a_{N1}	a_{N2}	$\cdots$	a_{Nk}
	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}
M rows	$\vdots$				$\vdots$				$\cdots$	$\vdots$			
	b_{M1}	b_{M1}	$\cdots$	b_{M1}	b_{M2}	b_{M2}	$\cdots$	b_{22}	$\cdots$	b_{Ml}	b_{Ml}	$\cdots$	b_{Ml}

2.4.7 Cyclotomy

In 2009, Colbourn [11] presented the *cyclotomic construction* derived from the analysis of cyclotomic matrices. Let ω be a primitive element of $GF(q)$ with $q \equiv 1 \pmod{v}$, the prime power selected for a finite field must have remainder 1 when divided by v. For any q and ω is possible to form a cyclotomic vector $x_{q,d,w} = (x_i : i \in GF(q)) \in GF(q)^q$. Let $x_0 = 0$ and $x_i = j(mod\ v)$ where $i = \omega^j$ for $i \in GF(q)^q$. The value of index i is given by the primitive element raised to the jth power using $GF(q)$. For example using the discrete logarithm table of $GF(9)$ $\{8, 4, 1, 2, 7, 5, 3, 6\}$, and using the remainder for $v = 2$ we get the vector $x_{9,2} = (000101110)$, and rotating the vector we obtain a matrix of 9×9 which is a CA$(9; 2, 9, 2)$ shown in Figure 2.4.

0	0	0	1	0	1	1	1	0
0	0	1	0	1	1	1	0	0
0	1	0	1	1	1	0	0	0
1	0	1	1	1	0	0	0	0
0	1	1	1	0	0	0	0	1
1	1	1	0	0	0	0	1	0
1	1	0	0	0	0	1	0	1
1	0	0	0	0	1	0	1	1
0	0	0	0	1	0	1	1	1

Figure 2.4: The CA$(9; 2, 9, 2)$ produced by the cyclotomic construction.

2.4.8 Construction using groups

The construction using groups were introduced by Chateauneuf *et al.* [9] for strength three And adapted by Meagher and Stevens [43] for strength. Meagher *et al.* produce a $CA(k(v-1)+1; 2, k, v)$ by means of a group $G < Sym_v$ and a starter vector $s \in \mathbb{Z}_v^k$. Given the group G find a starter vector $s \in \mathbb{Z}_v^k$ where $d_1, d_2, \ldots, d_{k-1}$ defined as $d_i = \{(s_j, s_{j+1}) | j = 0, 1, \ldots, k-1\}$ with indices j modulo k, have at least one element from every orbit of the group action of G on pairs of $\mathbb{Z}_v = \{0, 1, \ldots, v-1\}$. The starter vector s is used to form a matrix M of size $k \times k$ by juxtaposing vertically the rotations of s. After that, the group action G over M produce $v-1$ matrices of $k \times k$; these matrices and a constant vector of 0s form the CA. This two methods use a group of symbols of order $v-1$ where v is the alphabet and use one fixed symbol.

Later on, Lobb *et al.* [37] included fixing more than one symbol, the amount of fixed symbols is denoted by $f \in \{0, 1, 2\}$ and allows that the group of symbols be of order $v - f$. Let C be a finite group of size k and $\Gamma = (C, \odot)$ also a finite group, for each $c \in C$ let $inv_\Gamma(c)$ be the inverse of c. Let Γ have identity element e_Γ. Let $C_0, \ldots, C_{f-1}$ satisfy $C_i \bigcap C_j = \emptyset$ for $0 \leq i < j < f$. Let $\overline{C} = C \setminus \{C_0 \bigcup \ldots \bigcup C_{f-1}\}$. Then $(C_0, \ldots, C_{f-1})$ is a (k, f)-pattern for Γ if $\{inv_\Gamma(a) \odot b : a \in C_i, v \in \overline{C}\} = C \setminus \{e_r\}$ for each $0 \leq i < f$.

Suppose that $(C_0, \ldots, C_{f-1})$ is a (k, f)-pattern for Γ with $\overline{C} = C \setminus \{C_0 \bigcup \ldots \bigcup C_{f-1}\}$. Let $\Lambda = (V, \otimes)$ be a finite group of order $v - f$. A (v-f)-lift of a (k, f)-pattern is a relationship ϕ $\overline{C} \mapsto V$, so for each $\gamma \in C - e_\Gamma, inv_\Lambda(\phi(a)) \otimes \phi(b) : a, b \in \overline{C}, inv\Gamma(a) \odot b = \gamma = V$.

When a set of fixed columns$(C_0, \ldots, C_{f-1})$ is a (k, f)-pattern for $\Gamma = (C, \odot)$ having a $(v-f)$-lift mapped by ϕ $\overline{C} \mapsto V$ over $\Lambda = (V, \otimes)$,then $((C_0, \ldots, C_{f-1}), \phi)$ is a (k, v, f)-starter vector of (Γ, Λ).

Lobb *et al.* derived a lemma that say if a (k, v, f)-starter vector exists, then a matrix of $k(v-f) \times k$ on v symbols exist. $V \bigcup \{\infty_0, \ldots, \infty_{f-1}\}$ so that for every two distinct columns c and c' and every two elements $a, b \in V \bigcup \{\infty_0, \ldots, \infty_{f-1}\}$ there exists a row r in which the entry in cell (r, c) is a

and in the cell (r, c') is b, unless a and $b \in \{\infty_0, \ldots, \infty_{f-1}\}$.

Another case of interest is the (k, v, w, f)-relative cover starter, let $\mu = 0$ when $w = 0$ and $\mu = \frac{v-f}{w}$ when $w > 0$, if this vector exists:

1. $CAN(2, k, v) \leq k(v - f) + \mu CAN(2, k, w) + CAN(2, k, f)$

   For this case concatenating a CA of alphabet f plus the cyclic construction using the starter vector creates a valid CA when $w = 0$, but when $w > 0$ is necessary to add μ copies of a CA with alphabet w.

2. If the relative cover starter is distinct,

   (a) MCAN$(2, (v)^k(v - f + \alpha)) \leq k(v - f) + \mu$ MCAN$(2, w^k a^1)$ for $0 \leq a \leq f$.

   (b) If $w \leq f = 1$, CAN$(2, k + 1, v) \leq (k + 1)(c - 1) + 1$ particular case of previous incise when $a = 1$.

2.4.9 Permutation vectors

In 2006, Sherwood *et al.* [52] introduced permutation vectors, a compact representation of a CA based in one vector. When v is prime power, a covering perfect hash family is denoted by CPHF$(n; k, v^{t-1}, t)$ and is a matrix of $n \times k$ of v^{t-1} symbols in which every $n \times t$ matrix contains at least one row that is covering. The v^{t-1} symbols in a CPHF could be seen as a $(t - 1)$-tuple of v symbols, this tuple represents a permutation vector of length v^t over the finite fields $GF(v)$. Given the $(t - 1)$-tuple $(h_1, h_2, \ldots, h_{t-1})$ with $h_j \in \{0, 1, \ldots, v - 1\}$ for $1 \leq j \leq t - 1$, a permutation vector $(\overrightarrow{h_1, h_2, \ldots, h_{t-1}})$ of length v^t have the symbol $(h_{t-1} \cdot \beta_{t-1}^{(i)}) + \ldots + (h_2 \cdot \beta_2^{(i)}) + (h_1 \cdot \beta_1^{(i)}) + \beta_1^{(i)}$ on position i where i is represented in base v as $i = \sum_k^{(i)} v^k \cdot \beta_k^{(i)}$.

To determine if a set of permutation vectors are covering, consider the next t permutation vectors:

$$((h_1^{(1)}, h_2^{(1)}, \ldots, h_{t-1}^{(1)}), (h_1^{(2)}, h_2^{(2)}, \ldots, h_{t-1}^{(2)}), \ldots, (h_1^{(t)}, h_2^{(t)}, \ldots, h_{t-1}^{(t)}))$$

This set is covering if the expansion to a matrix of size $v^t \times t$ is an OA(t, t, v). To verify this condition is necessary that the matrix contains at least one of the t-tuples is repeated. Then a set of permutation vectors is not covering iff exist $i, j \in \{0, 1, \ldots, v^t - 1\}$ $i \neq j$ such that:

$$\begin{aligned}
\beta_0^{(i)} + (h_1^{(1)} \cdot \beta_1^{(i)}) + \ldots + (h_{t-1}^{(1)} \cdot \beta_{t-1}^{(i)}) &= \beta_0^{(j)} + (h_1^{(1)} \cdot \beta_1^{(j)}) + \ldots + (h_{t-1}^{(1)} \cdot \beta_{t-1}^{(j)}) \\
\beta_0^{(i)} + (h_1^{(2)} \cdot \beta_1^{(i)}) + \ldots + (h_{t-1}^{(2)} \cdot \beta_{t-1}^{(i)}) &= \beta_0^{(j)} + (h_1^{(2)} \cdot \beta_1^{(j)}) + \ldots + (h_{t-1}^{(2)} \cdot \beta_{t-1}^{(j)}) \\
&\vdots \\
\beta_0^{(i)} + (h_1^{(t)} \cdot \beta_1^{(i)}) + \ldots + (h_{t-1}^{(t)} \cdot \beta_{t-1}^{(i)}) &= \beta_0^{(j)} + (h_1^{(t)} \cdot \beta_1^{(j)}) + \ldots + (h_{t-1}^{(t)} \cdot \beta_{t-1}^{(j)})
\end{aligned}$$

If we define $\alpha_r = \beta_r^{(i)} - \beta_r^{(j)}$ for $0 \leq r \leq t - 1$ for certain values of i and j, equations could be rearranged to:

$$\begin{aligned}
\alpha_0 + (h_1^{(1)} \cdot \alpha_1) + (h_2^{(1)} \cdot \alpha_2) + \ldots + (h_{t-1}^{(1)} \cdot \alpha_{t-1}) &= 0 \\
\alpha_0 + (h_1^{(2)} \cdot \alpha_1) + (h_2^{(2)} \cdot \alpha_2) + \ldots + (h_{t-1}^{(2)} \cdot \alpha_{t-1}) &= 0 \\
&\vdots \\
\alpha_0 + (h_1^{(t)} \cdot \alpha_1) + (h_2^{(t)} \cdot \alpha_2) + \ldots + (h_{t-1}^{(t)} \cdot \alpha_{t-1}) &= 0
\end{aligned}$$

The approach used for constructing permutation vectors by Sherwood was exhaustive for $t = 3, 4$. Later on, Walker and Colbourn [64] used a tabu search algorithm, achieving solution for strengths $3, 4, 5, 6, 7$.

2.4.10 Towers of covering arrays

In 2013, Torres-Jimenez *et al.* [60] presented a method for construction of CAs denominated *Towers of Covering Arrays* (TCA). Defining the height h of a CA as a succession of $h+1$ CAs $c_0, c_1, \ldots, c_h$ where c_0 is the base CA and $1 < i < h$, c_i represents a CA with a strength superior in one unit of c_{i-1}. This methodology starts wit a covering array $CA(N; t, k, v)$ denoted as A to generate a $CA(Nv, t+1, k+1, v)$ by juxtaposing v vertical copies of A and translating the j-th column of the

i-th copy adding a value c and applying modulo v to every value of the column. The column $k+1$ is constructed by adding N 0s then N 1s, and so on until adding N $(v-1)$s. The obtained CA could be used as a base for a TCA of higher height.

2.4.11 Binomial coefficients

In 2015, Torres Jimenez *et al.* [59] presented the construction of covering arrays with $v = 2$ using binomial coefficients. A binomial coefficient with parameters k and r represents the set of all the k-tuples from $\{0, 1\}$ that have r ones and $k - r$ zeroes. A branch and bound algorithm is presented to find the set of binomial coefficients that, when juxtaposed, provide a covering array CA$(N; t, k, 2)$ with the minimum possible number of rows. This construction finds optimal covering arrays for $k = t + 1$ and $k = t + 2$.

2.4.12 Trinomial coefficients

In 2010, Martínez-Pena *et al.* [41] [42] proposed the construction of CAs with $v = 3$ based in a set of rows represented as trinomial coefficients. To calculate a trinomial coefficient let a, b, c, k integers such that $0 \leq a, b, c \leq k$ and $k = a + b + c$.

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

. That coefficient represents the set of different rows of k elements that can be created with a number of 0s, b number of 1s and c number of 2s. For any strength $t \leq k$ a CA of k columns could be constructed by vertically juxtaposing the subset of rows represented by the trinomial coefficients. For example, with this construction for $t = 2$ and $k = 4$ the CA with minimal rows is created with the coefficients: $\binom{4}{3,0,1}, \binom{4}{0,1,3}, \binom{4}{1,3,0}$ which form a CA with 12 rows CA$(12; 2, 4, 3)$.

2.5 Post-optimization methods to reduce the number of rows of CAs

Post-optimization methods improve the previously constructed CAs. Post-optimization methods receive a CA and try to reduce the number rows by exploiting that some interactions are covered more than once.

In 2013, Nayeri *et al.* [44] presented a randomized post-optimization algorithm. This algorithm uses flexible positions in the CA, a flexible position or wildcard is the one that has the possibility to be changed arbitrarily or omitted in the determination of the coverage, but the CA still fulfill all the interactions. When there are two or more flexible positions, the use of a flexible position may make other positions no longer flexible so, the flexibility of positions is extended to a flexible set. This strategy chooses a flexible set F and places different symbols in each position of F; forming a new CA with a possibly different collection of flexible positions. In some cases, a flexible set contains an entire row which can be deleted and, therefore, a CA is improved by the deletion of a row.

Later on, in 2014 Li *et al.* [36] proposed a modification of Nayeri's algorithm, using a deterministic selection of the row to delete instead of doing the search among all the rows, avoiding the permutation of rows because the nominated row to be deleted is independent of the order. They also provide an estimator to predict the extent of when a CA can be improved.

2.6 CA related problems mapped to a graph representation

2.6.1 Minimization of constant rows in covering arrays

Quiz-Ramos in [48] used a graph representation to maximize the amount of constants rows in a CA, in this case each node represents a row of the CA and an edge exists between two nodes if the rows of the CA are compatible. Two rows R, R' are compatible if $R_i \neq R'_i, 0 \leq i \leq k-1$. Maximization of constant rows is transformed into the problem of finding the maximum clique [39] of the graph generated. When a maximum clique is obtained the rows involved of the CA are replaced by constant rows of the alphabet values involved.

2.6.2 Covering arrays completion by the vertex coloring problem of a graph

Recently, in Sarkar's PhD dissertation [51] is shown a mapping of the missing interactions of a PCA to the graph domain that represent the incompatibility between interactions. The nodes of that graph represent each missing interaction and an edge exists between two nodes if their interactions cannot be covered in the same row. The problem to be solved in the graph domain is the vertex coloring problem, that is NP-complete [21] and is solved using a greedy approach. The better approximation of the chromatic number lowers the number of rows added to fulfill the coverage.

2.7 Summary

In this chapter we presented several relevant methods to construct covering arrays using the following representations: a matrix of integers, integer programming, constraint programming, a set of linked

lists, a set of binomial coefficients, a set of trinomial coefficients, a cyclotomic vector, a vector of integers, and permutation vectors; but none of them use a graph representation to directly solve CACP.

The presented approaches were classified in five categories: exact, greedy, metaheuristic, algebraic, and post-optimization. The exact methods produce optimal covering arrays, but they are practical only to construct covering arrays of small size given that the search space grows exponentially, and to ensure optimal results all the search space is explored. The greedy methods generate covering arrays of greater size than the exact methods in less time, these results are not optimal but offer competitive results in short time. Metaheuristic methods also do not guarantee to find optimal covering arrays, but provide better solutions than the greedy ones. Algebraic methods produce very good results, but in the process of constructing a covering array they involve formulas and operations with mathematical objects that in some cases are only available for a special combination of values for the parameters $t, k,$ and v. The post-optimization methods attempt to reduce the number of rows of a covering array by exploiting the positions that can be changed arbitrarily, until all the positions of a row are disposable which means that this row can be deleted.

In the next chapter a graph representation to mapping the characteristics of a covering array will be presented. It also will be shown how the CACP is mapped to the graph domain.

3

A graph representation for covering arrays

This chapter presents a novel representation that embeds the characteristics of a covering array into a graph. The mapping of a CA may vary according on how the graph embeds the characteristic of the CAs.

The following sections describe the representation denominated Covering in Nodes (CN), explain in detail the way the characteristics of a CA are mapped to a graph, and the equivalent problem in the graph domain of the CACP.

The proposed representation allows us to construct a CA with the minimum number of rows for some values of factors (k) and alphabet (v). It is remarkable that the proposed representation enables to construct a CA one row at a time.

3.1 Coverage in nodes

The proposed approach for CAN determination is denominated as Coverage in Nodes (CN) representation, this representation embeds in the nodes of a simple graph each interaction that must be present in a CA. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ the set of edges representing relationships between nodes. To embed the characteristics of a CA, each node represents a selection of t of the k factors with a value for each factor selected over the $\mathbb{Z}_v = \{0, 1, \ldots, v-1\}$, in other words, an interaction.

The number of nodes of the corresponding graph is given in Theorem 1 and the set of nodes is described in Equation (3.1). The edge set $\mathcal{E}$ establishes the compatibility between nodes. Two nodes are compatible if there are no factors in common or if the values assigned for the factors in common are the same, this is shown in Definition 11. In other words, if there is a subset of columns represented in both nodes the values assigned for each column must be the same, and this operator of compatibility is denoted by the symbol Λ in Equation (3.3). Theorem 2 shows the degree of each node and is the basis for the Theorem 3 that shows the number of elements in the set of edges of the graph $\mathcal{G}$ and the Equation (3.2) describes this edge set.

$$\begin{aligned}\mathcal{V} =&\{(\alpha_0\beta_0 \ldots \alpha_i\beta_i \ldots \alpha_{t-1}\beta_{t-1}) \mid (\forall i, j \in \{0, \ldots, k-1\} \wedge i < j)(\alpha_i < \alpha_j) \\ &\wedge \alpha_i \in \{0, \ldots, k-1\} \wedge \beta_i \in \{0, \ldots, v-1\}\}\end{aligned} \tag{3.1}$$

$$\begin{aligned}\mathcal{E} =&\{(\alpha_0\beta_0 \ldots \alpha_i\beta_i \ldots \alpha_{t-1}\beta_{t-1}), (\gamma_0\delta_0 \ldots \gamma_i\delta_i \ldots \gamma_{t-1}\delta_{t-1}) \mid \alpha_i, \gamma_i \in \{0, \ldots, k-1\} \\ &\wedge \beta_i, \delta_i \in \{0, \ldots, v-1\} \wedge \Lambda((\alpha_0\beta_0 \ldots \alpha_i\beta_i \ldots \alpha_{t-1}\beta_{t-1}), (\gamma_0\delta_0 \ldots \gamma_i\delta_i \ldots \gamma_{t-1}\delta_{t-1})) = true\}\end{aligned} \tag{3.2}$$

Theorem 1

Given $t, k,$ and v, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated by the representation CN of this parameters, stands that $|\mathcal{V}| = v^t\binom{k}{t}$.

Proof. Since a node is the encoding of t factors with its respective value over the set $\mathbb{Z}_v = \{0, \ldots, v-1\}$, there are $\binom{k}{t}$ ways to choose factors, and for every possible choice of factors there exists v^t ways to fix their values, then $|\mathcal{V}| = v^t\binom{k}{t}$. □

Definition 11 (Compatibility)

Given $t, k,$ and v, in the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated by the representation CN of this parameters, two nodes $(\alpha_0\beta_0 \ldots \alpha_i\beta_i \ldots \alpha_{t-1}\beta_{t-1})$ and $(\gamma_0\delta_0 \ldots \gamma_i\delta_i \ldots \gamma_{t-1}\delta_{t-1})$ are compatible iff

$\forall p, q \mid (\alpha_p = \gamma_q) \implies (\beta_p = \delta_q) \vee \forall p, q \mid (\alpha_p \neq \gamma_q)$ for $0 < p, q < t-1$, this operation is denoted by letter Λ in Equation (3.3).

Theorem 2

Given $t, k,$ and v, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated by the representation CN with this parameters, stands that $deg(\psi \in \mathcal{V}) = \sum_{i=1}^{\min(t,k-t)} \binom{t}{i}\binom{k-t}{i} v^i$.

Proof. Using Definition 11 it is possible to determine the number of edges that involve a particular node. A node has subsets S where $|S| \in \{1, \ldots, t\}$ representing the $|S|$ factors selected that can be replaced with another of the $k-t$ remaining factors. A set S has $\binom{t}{|S|}$ different ways to be formed by using the t factors of the node. Each set could be replaced in $\binom{k-t}{|S|}$ distinct ways and each replacement has $v^{|S|}$ possible configurations of values taken from the alphabet. In some cases where $k < 2t$ the amount of columns available to replace the $|S|$ selected are not enough, so it is only possible to replace $\min(t, k-t)$ columns. Then a node has $deg(\psi \in \mathcal{V}) = \sum_{i=1}^{\min(t,k-t)} \binom{t}{i}\binom{k-t}{i} v^i$. □

Theorem 3

Given t, k, and v, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated by representation CN with this parameters, stands that $|\mathcal{E}| = \frac{1}{2} v^t \binom{k}{t} \sum_{i=1}^{\min(t,k-t)} \binom{t}{i} \binom{k-t}{i} v^i$.

Proof. Each node in the graph $\mathcal{G}$ has the same degree, using Theorem 2 and the identity $2|\mathcal{E}| = \sum_{\forall v \in \mathcal{V}} deg(v)$. Then $|\mathcal{E}| = \frac{1}{2} v^t \binom{k}{t} \sum_{i=1}^{\min(t,k-t)} \binom{t}{i} \binom{k-t}{i} v^i$. □

$$\Lambda((\alpha_0\beta_0 \dots \alpha_i\beta_i \dots \alpha_{t-1}\beta_{t-1}), (\gamma_0\delta_0 \dots \gamma_i\delta_i \dots \gamma_{t-1}\delta_{t-1})) = \begin{cases} true & \text{if } \forall p, q | (\alpha_p = \gamma_q) \Longrightarrow (\beta_p = \delta_q) \\ & \vee \quad \forall p, q | (\alpha_p \neq \gamma_q) \\ & \text{for } 0 < p, q < t-1 \\ false & \text{otherwise} \end{cases} \tag{3.3}$$

There is an example of how a CA is mapped to the graph domain in Figure 3.1, corresponding graph for a CA with $k = 3, t = 2$ and $v = 2$, it shows that the graph has 12 nodes and 24 edges.

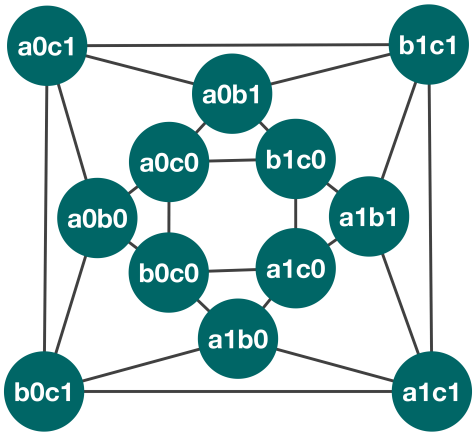
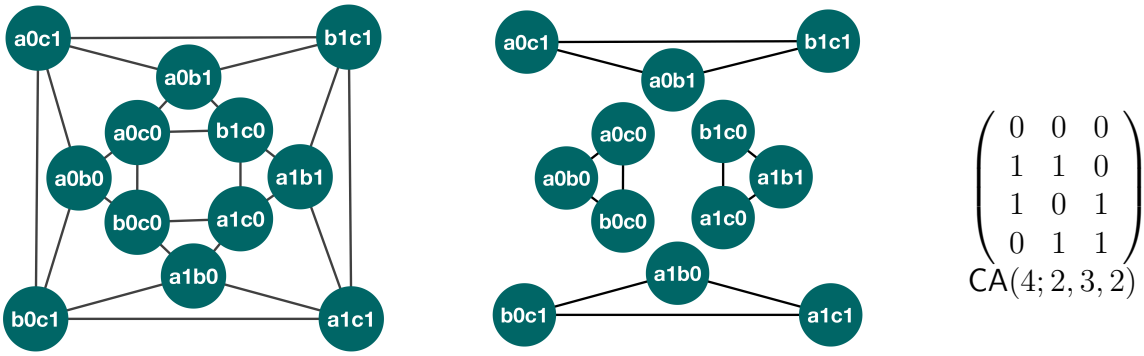


Figure 3.1: The graph generated for parameters $t = 2, k = 3$, and $v = 2$ by the coverage in nodes representation.

Since the nodes are equivalent to the interactions provided by a covering array and the set of edges represent the coverage compatibilities, every complete subgraph of $\mathcal{G}$ represents a set of nodes that are pairwise compatible, this means that a set of factors with their respective fixed value of the alphabet have no collision to build a row of the covering array. If we manage to obtain the maximum subgraph of graph $\mathcal{G}$, the row generated would provide a maximum coverage of missing interactions. The problem of determining the largest complete subgraph is known as the *maximum clique problem*, and determining the set of cliques that covers all the nodes of a graph is the *minimum clique covering problem*.

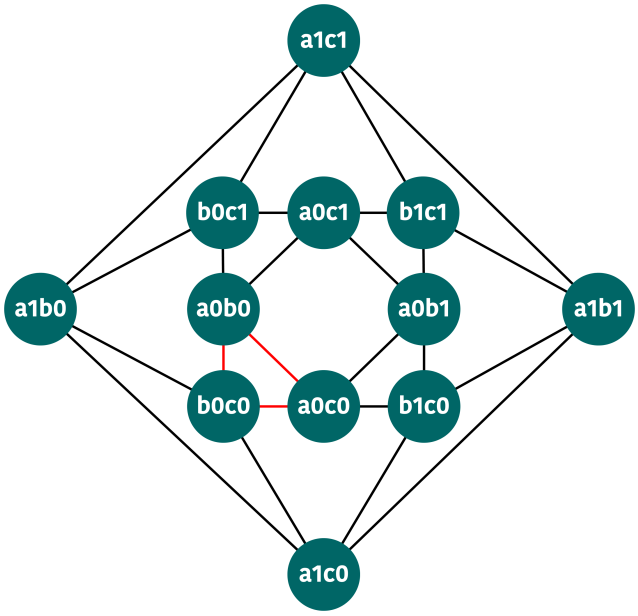
The solution process of the corresponding graph for a CA with $k = 3, t = 2$ and $v = 2$ is shown in the Figure 3.2. It can be seen that the constructed maximum cliques are: $\{a0b0, a0c0, b0c0\}$; $\{a1b1, a1c0, b1c0\}$; $\{a1b0, a1c1, b0c1\}$; $\{a0b1, a0c1, b1c1\}$, corresponding to the rows: 000; 110; 101; 011, respectively.



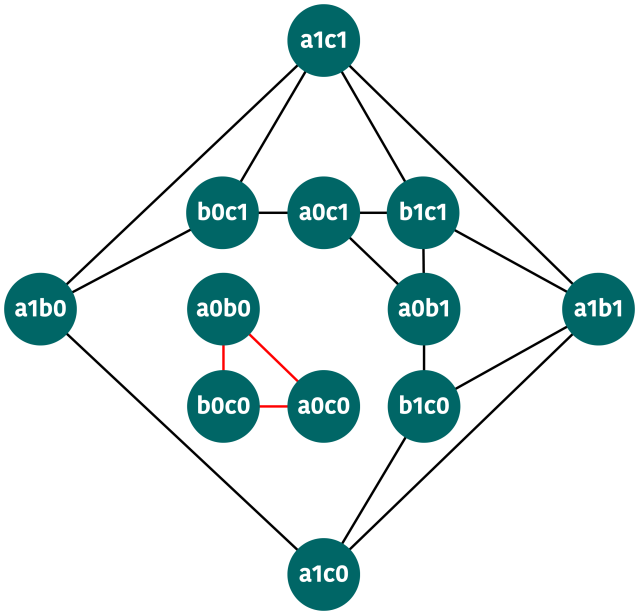
$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$
$\mathsf{CA}(4; 2, 3, 2)$

Figure 3.2: Original graph, solved graph, and the CA constructed for the instance with parameters $t = 2, k = 3$ and $v = 2$.

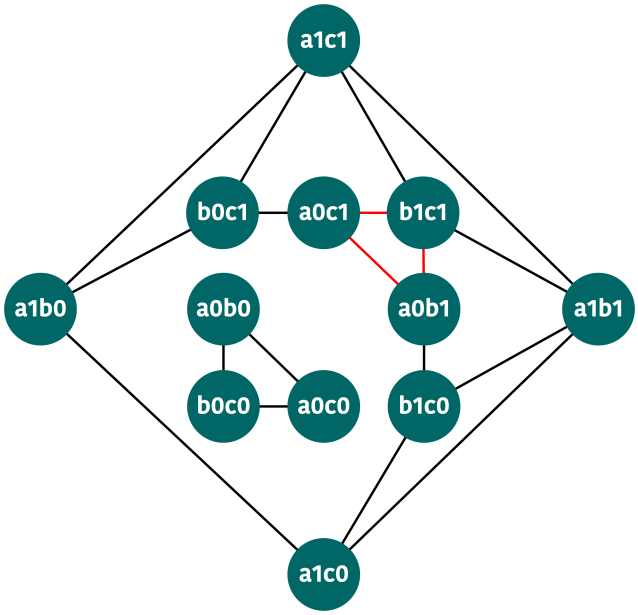
A detailed example of the step by step solution of a covering array instance is depicted in Figure 3.3, the clique selected to be added to the clique cover is colored in red, the images in the left columns show the clique that is selected, and the images in the right columns show the edges that are deleted of the graph after fixing the clique selected.



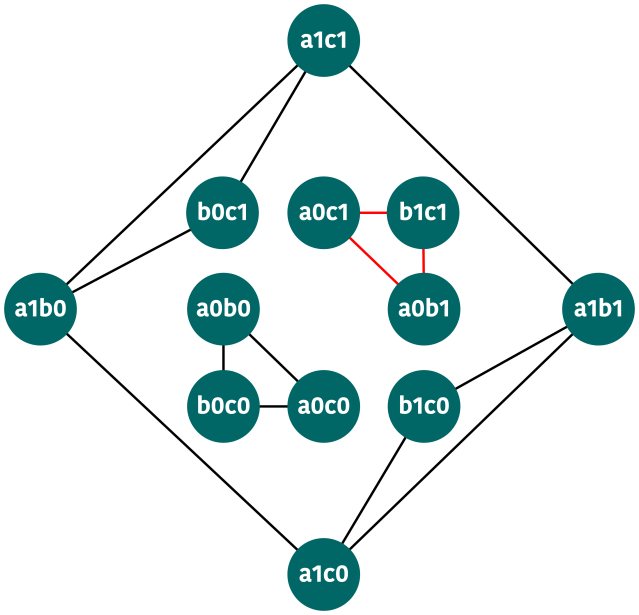
(a) The clique $\{a0b0, a0c0, b0c0\}$ is chosen, this clique forms the row $0, 0, 0$.



(b) All incident edges to the clique $\{a0b0, a0c0, b0c0\}$ are deleted.

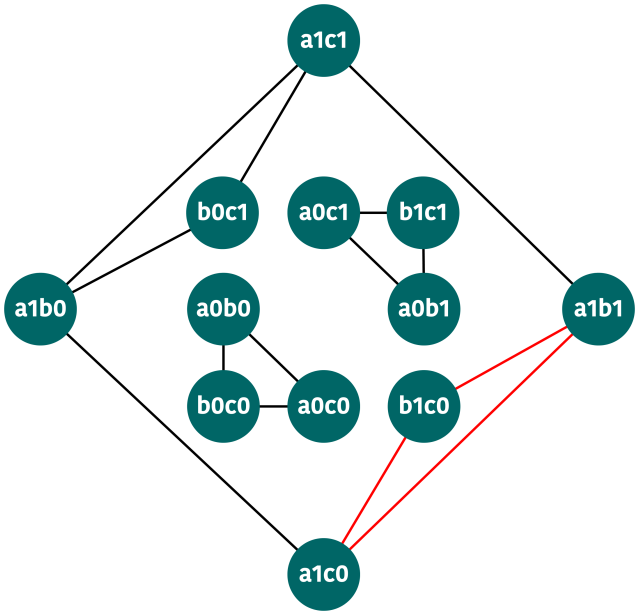


(c) The clique $\{a0b1, a0c1, b1c1\}$ is chosen, this clique forms the row $0, 1, 1$.

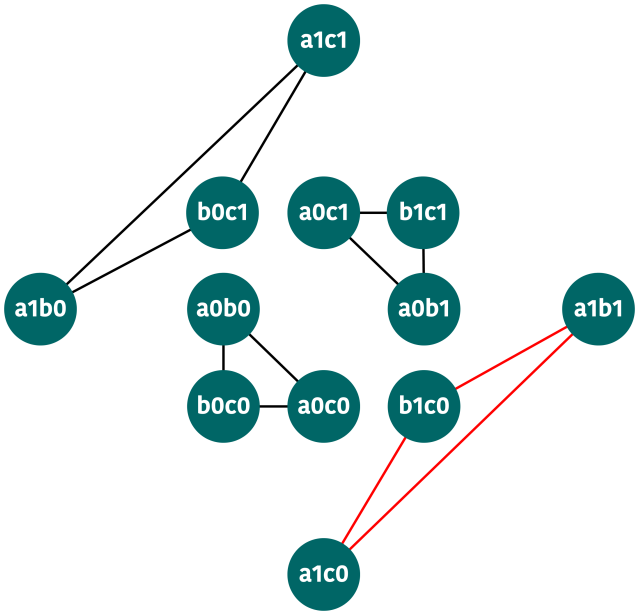


(d) All incident edges to the clique $\{a0b1, a0c1, b1c1\}$ are deleted.

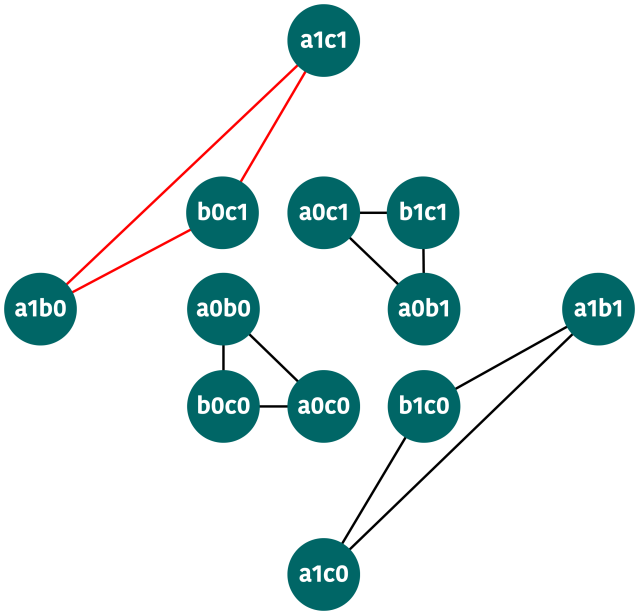
Figure 3.3: A step by step solution of the clique covering problem for the covering array instance CA$(2, 3, 2)$. Part 1 of 2.



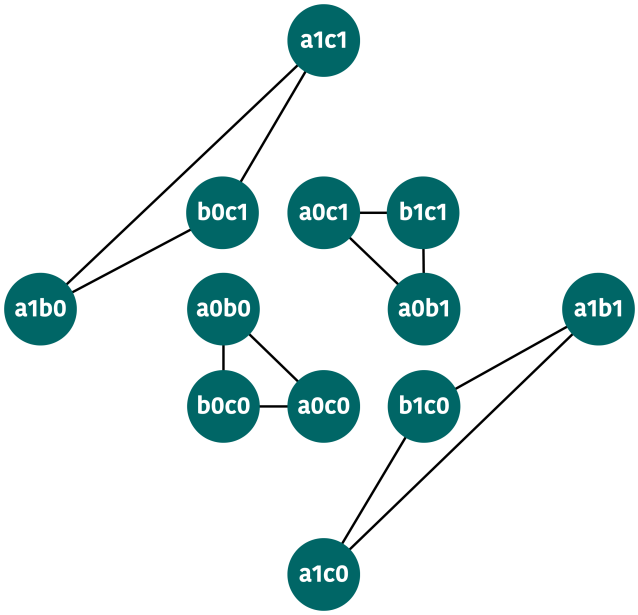
(e) The clique $\{a1b1, a1c0, b1c0\}$ is chosen, this clique forms the row $1, 1, 0$.



(f) All incident edges to the clique $\{a1b1, a1c0, b1c0\}$ are deleted.



(g) The clique $\{a1b0, a1c1, b0c1\}$ is chosen, this clique forms the row $1, 0, 1$.



(h) The obtained clique cover with 4 cliques.

Figure 3.4: A step by step solution of the clique covering problem for the covering array instance CA$(2, 3, 2)$. Part 2 of 2.

3.1.1 Representing a mixed covering array with the coverage in nodes representation

It is also possible to map the characteristics of an MCA to a graph with the representation CN, but now each factor has its own alphabet, the set of factors $C = \{0, \ldots, k-1\}$ has the set of values $V = \{v_0, \ldots, v_{k-1}\}$, derived from Theorem 1 the actual number of nodes for a MCA is shown in Theorem 4 and is described in Equation (3.4). To obtain the number of edges of a node it is necessary to know the factors that are not involved in the node and to replace subsets of size $\{1, \ldots, t\}$ in every possible way, the number of edges is shown in Theorem 5 and the edge set is described in Equation (3.5).

$$\begin{aligned}\mathcal{V} =&\{(\alpha_0\beta_0 \ldots \alpha_i\beta_i \ldots \alpha_{t-1}\beta_{t-1}) \mid (\forall i, j \in \{0, \ldots, k-1\} \wedge i < j)(\alpha_i < \alpha_j) \\ &\wedge \alpha_i \in \{0, \ldots, k-1\} \wedge \beta_i \in \{0, \ldots, v_{\alpha_i} - 1\}\}\end{aligned} \tag{3.4}$$

$$\begin{aligned}\mathcal{E} =&\{(\alpha_0\beta_0 \ldots \alpha_i\beta_i \ldots \alpha_{t-1}\beta_{t-1}), (\gamma_0\delta_0 \ldots \gamma_i\delta_i \ldots \gamma_{t-1}\delta_{t-1}) \mid \alpha_i, \gamma_i \in C \wedge \beta_i \in \{0, \ldots, v_{\alpha_i} - 1\} \\ &\wedge \delta_i \in \{0, \ldots, v_{\gamma_i} - 1\} \wedge \Lambda((\alpha_0\beta_0 \ldots \alpha_i\beta_i \ldots \alpha_{t-1}\beta_{t-1}), (\gamma_0\delta_0 \ldots \gamma_i\delta_i \ldots \gamma_{t-1}\delta_{t-1})) = true\}\end{aligned} \tag{3.5}$$

Theorem 4

Given t, k and a set $(v_0, \ldots, v_{k-1})$ of available alphabet for each distinct factor a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated by the representation CN with this parameters, stands that

$$|\mathcal{V}| = \sum_{\forall (a_0,\ldots,a_{t-1})_{\neq} \in C} \prod_{i=0}^{t-1} v_{a_i}$$

.

Proof. Since the set of nodes is a concatenation formed by a column of the set $C = \{0, \ldots, k-1\}$ and its respective alphabet of the set $V = \{v_0, \ldots, v_{k-1}\}$ the number of nodes depends on which factor is selected in the node and its own possible values. The possible values for $(a_0, \ldots, a_{t-1})_{\neq} \in C$ is $\prod_{i=0}^{t-1} v_{a_i}$ then $|\mathcal{V}| = \sum_{\forall (a_0,\ldots,a_{t-1})_{\neq} \in C} \prod_{i=0}^{t-1} v_{a_i}$ □

Theorem 5

Given t, k and a set $(v_0, \ldots, v_{k-1})$ of values for each distinct factor a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated by the representation CN with this parameters, stands that

$$|\mathcal{E}| = \frac{1}{2} \sum_{\forall (a_0,\ldots,a_{t-1})_{\neq} \in C} \left(\prod_{j=0}^{t-1} v_{a_j} \sum_{i=1}^{\min(t,k-t)} \binom{t}{i} \sum_{\forall (b_0,\ldots,b_{i-1})_{\neq} \in C \setminus \{a_0,\ldots,a_{t-1}\}} \prod_{j=0}^{i-1} v_{b_j} \right)$$

.

Proof. The set of edges represent compatible nodes, using Definition 11 and overriding the number of values available for each member of the subset used as substitute of the actual factors in the node, because now each factor has its own alphabet. Then a node with columns $(a_0, \ldots, a_{t-1})_{\neq} \in C$ has

$$\sum_{i=1}^{\min(t,k-t)} \binom{t}{i} \sum_{\forall (b_0,\ldots,b_{i-1})_{\neq} \in C \setminus \{a_0,\ldots,a_{t-1}\}} \prod_{j=0}^{i-1} v_{b_j}$$

compatible nodes, then

$$|\mathcal{E}| = \frac{1}{2} \sum_{\forall (a_0,\ldots,a_{t-1})_{\neq} \in C} \left(\prod_{j=0}^{t-1} v_{a_j} \sum_{i=1}^{\min(t,k-t)} \binom{t}{i} \sum_{\forall (b_0,\ldots,b_{i-1})_{\neq} \in C \setminus \{a_0,\ldots,a_{t-1}\}} \prod_{j=0}^{i-1} v_{b_j} \right)$$

. □

3.1.2 Handling constraints on the coverage in nodes representation

For this mapping, constraints of size ≤ 2 can be fully handled as it is shown in Theorem 6 but, for greater sizes the constraint handling is not granted for $t > 2$ as it is demonstrated in Theorem 7, then the validation of constraints has to be performed when the graph problem is solved. However, reducing the amount of nodes and edges even if the constraint is not completely handled is worth because it represents a reduction in the running time of a solution algorithm. Predicting how the parameters would be reduced is a difficult problem, starting on a graph without constraints and adding the first constraint of size ζ the edges reduced are given by:

$$|\mathcal{E}_r| = \binom{|\mathcal{V}| - a}{t - a} \binom{|\mathcal{V}| - \zeta}{t - b} \tag{3.6}$$

where a, b represent all possible solutions for the diophantine equation $a + b = \zeta$ for $a, b > 0$ and $a \leq b$.

Predicting the next constraints becomes an inclusion-exclusion problem (see [22] for examples), because edges that are already deleted would be taking into account in another constraint. So, for an accurate prediction of the number of edges deleted it is necessary to evaluate the intersection of the edges deleted by the current constraint in process with the edges that are already deleted by another constraint.

Theorem 6

Given t, k, and v a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated by the representation CN with this parameters, it is possible to handle constraints of size ≤ 2.

Proof. Since a node is a concatenation of t columns and values, a constraint of size 1 $(a), (x)$ is handled by deleting all the nodes where the column a is concatenated with the value x and all edges involving between these nodes.

A constraint $(a, b), (x, y)$ where $a, b \in \{0, \ldots, k-1\}$ and $x, y \in \{0, \ldots, v-1\}$ is handled by deleting the edges where the union of both nodes includes the columns a, b with the respective values x, y. □

Theorem 7

Given $t > 2, k$, and v a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated by the representation CN with this parameters, it is not possible to handle constraints of size > 2.

Proof. A constraint $(a, b, c, \ldots)(x, y, z, \ldots)$ where $a, b, c, \ldots \in \{0, \ldots, k-1\}$ and $x, y, z, \ldots \in \{0, \ldots, v-1\}$ could be rearranged as an ordered pair of (column, value) as $(a, x), (b, y), \ldots$, since the solution is given in terms of nodes involved in a clique, there could be a set of m nodes where:

$$\{(a, x), (b, y), (c, z), \ldots\} \subseteq \bigcup_{i=0}^{m} \{(\alpha_{0,i}, \beta_{0,i}), \ldots, (\alpha_{t-1,i}, \beta_{t-1,i})\}$$

which results in a constraint violation even if all the edges where the constraints are involved are deleted when m is large enough the union could partially construct the factors fixed with the values that are involved in the set of constraints. □

	Set of columns	Set of values
1	(a, b)	$(0, 0)$
2	(b, d)	$(1, 0)$
3	(a, c)	$(0, 0)$
4	(a, c)	$(1, 0)$
5	(c, d)	$(1, 1)$
6	(b, c)	$(0, 1)$

Table 3.1: A set of constraints for a CA with parameters $t = 2, k = 4$, and $v = 2$.

Figure 3.5 shows an instance with parameters $t = 2, k = 4$, and $v = 2$ using constraints in Table 3.1. This instance without constraints has 24 nodes and 144 edges and constraint version has 18 nodes and 32 edges.

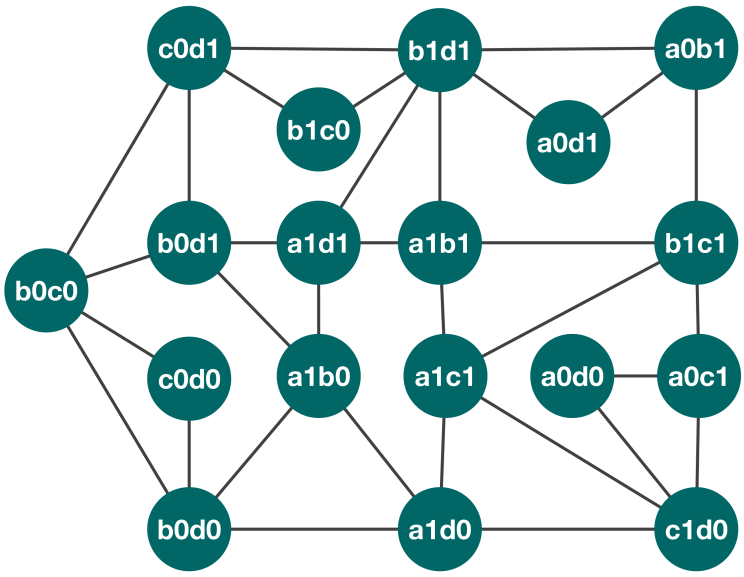


Figure 3.5: The graph generated for parameters $t = 2, k = 3$ and $v = 2$ by the representation CN with constraints.

3.2 Flexible positions using the coverage in nodes representation

Flexible positions, also known as wildcard positions, are elements of a CA that can be modified without decreasing its coverage provided. Using a vector or matrix of integers to represent a CA($N; t, k, v$), the flexible positions are identified by a value v, this flexible positions are studied in the state of the art (see [17, 25]) and have been used to post-optimize CAs in [44, 36, 62]. With the representation CN, the detection of flexible positions is performed in the mapping of the graph-based solution to the matrix integer domain. In the previous sections we described that the equivalent problem of the CACP in the graph domain is the minimum clique covering problem, where each clique represents a row of the CA. Since every node in the clique may fix values to a set of t factors, it is possible that some factors in a row are not fixed due to the number of nodes in the clique. If a factor is not fixed then this factor is a flexible position.

For example, Figure 3.7 presents a graph instance for the CA$(2, 5, 2)$, and Figure 3.8 shows a clique cover for the instance. To map this solution each clique is depicted to a row of the CA:

- The clique $\{a1b0, a1c0, a1d1, a1e1, b0c0, b0d1, b0e1, c0d1, c0e1, d1e1\}$ fixes values for factors a, b, c, d, e.
- The clique $\{a0b0, a0c1, a0d0, a0e0, b0c1, b0d0, b0e0, c1d0, c1e0, d0e0\}$ fixes values for factors a, b, c, d, e.
- The clique $\{a1b1, a1d0, a1e0, b1c0, b1d0, b1e0, c0d0, c0e0\}$ fixes values for factors a, b, c, d, e.
- The clique $\{a0b1, a0d1, a0e1, b1c1, b1d1, b1e1, c1d1, c1e1\}$ fixes values for factors a, b, c, d, e.
- The clique $\{a1c1, d1e0\}$ fixes values for factors a, c, d, e.
- The clique $\{a0c0, d0e1\}$ fixes values for factors a, c, d, e.

The cliques $1, 2, 3, 4$ fix values for all the parameters of the CA, the cliques 5 and 6 do not fix a

value to the b factor, so, the value for the factor b in the rows 5 and 6 is a flexible position, and its shown in Figure 3.6.

1	0	0	1	1
0	0	1	0	0
0	1	1	1	1
1	1	0	0	0
0	2	0	0	1
1	2	1	1	0

Figure 3.6: The covering array CA$(6; 2, 5, 6)$ with two flexible positions for the second factor in rows 5 and 6.

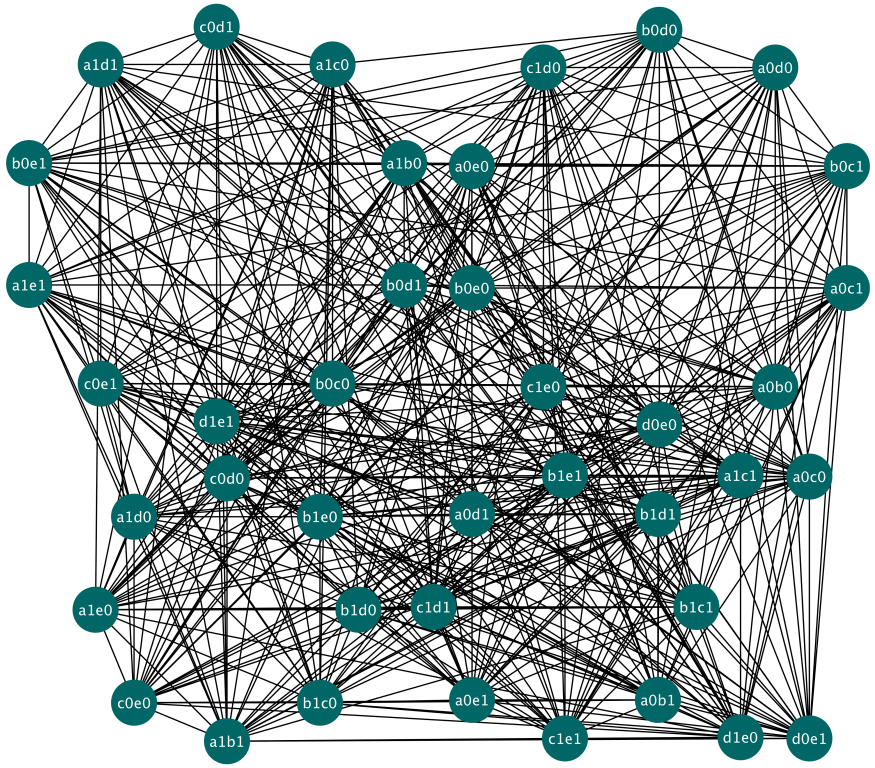


Figure 3.7: The graph generated for parameters $t = 2, k = 5$, and $v = 2$ by the coverage in nodes representation.

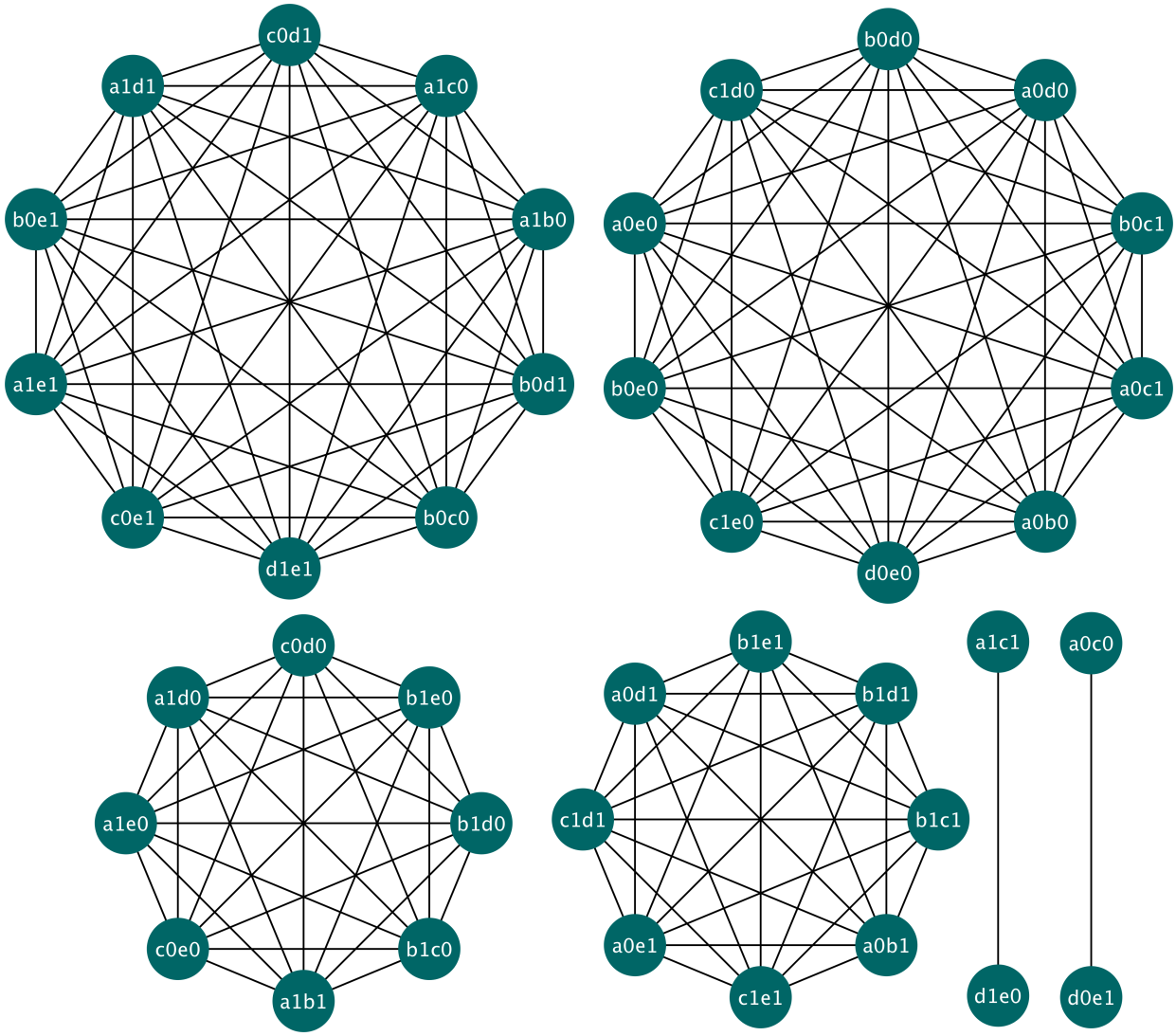


Figure 3.8: A clique cover for graph generated for the instance CA$(2, 5, 2)$ by the coverage in nodes representation.

3.3 Summary

This chapter presented the proposed representation of the characteristics of a covering array in a graph, is analyzed and described the general characteristics of the graph generated through the representation denoted as coverage in nodes (CN) of a covering array instance.

Furthermore, it is described the mapping of mixed covering arrays to the graph domain using the proposed representation and the way that constraints are handled. The representation CN allows to map the CACP to an equivalent problem in the graph domain, which is the minimum clique covering problem that is explained in the following chapter. The next chapter will describe the methodology applied to solve this equivalent problem with three distinct approaches: exact, greedy, and metaheuristic.

4

Methodology to solve the CACP in the graph domain

This chapter describes the methodology to solve CACP in the graph domain using the representation CN. This representation requires finding the minimum clique cover, which is an NP-hard problem and its decision version is NP-complete [21]. Each clique in the clique cover represents a row of a CA, therefore, the lesser cliques involved in the clique cover the better CA that is constructed. Section 4.1 presents an overview of the proposed methodology. Section 4.2 presents algorithms that are used by proposed exact, greedy, and metaheuristic implementations. Section 4.3 describes an exact algorithm to solve the clique cover problem. In Section 4.4 four greedy algorithms for the clique cover problem are explained. Section 4.5 shows a metaheuristic implementation to solve the clique cover problem. Finally, Section 4.6 shows a post-optimization algorithm using a graph representation to reduce the number of rows of CAs.

4.1 Methodology overview

Based on the specific objectives described in the Section 1.5, the proposed methodology consists in: a) the development of an algorithm to map a CA(t, k, v) instance to their respective graph, b) the development of exact, greedy and metaheuristic algorithms to solve the minimum clique cover problem according to the size of the instance, c) the development of an algorithm to map the clique cover into a CA and, d) the development of an algorithm to post-optimize CAs in the graph domain previously constructed by any method.

4.2 Utility algorithms

The exact algorithm and two versions of the greedy algorithm (to be discussed in the following sections) rely in knowing the list of all maximal cliques of the graph instance. To obtain the list, two algorithms are developed: a) an algorithm to list all the maximum cliques of a graph, and b) a greedy version for vertex coloring of a graph.

An instance of a CA without constraints using the representation CN stands that all maximal cliques of the graph are maximum. The algorithm to list all maximum cliques uses the greedy vertex coloring as a pruning method.

4.2.1 List all maximum cliques of a graph

This algorithm builds a clique in a recursive way, using a lexicographically order to avoid listing isomorphic cliques. Starting with each possible node of the graph, for example node v, then obtain the neighborhood of v (N_v) and call to a inner level of recursion. After that, the nodes available to append to the clique are the included in N_v, attach a node $u \in N_v$ to the clique, compute the new neighborhood as the intersection of N_v and N_u, and call to an inner level of recursion. Every element of a current neighborhood must be lexicographically greater than the current clique.

The current clique is included in the list when it reaches the size of the cliques already included, and if the current clique exceeds the size of the cliques in the list, all the elements of the list are deleted and the new maximum clique is added to the list. Algorithm 1 provides the pseudocode for this algorithm.

Algorithm 1 An exact algorithm to obtain all the maximum cliques of a graph generated by the CN representation.

Input: Graph $G = (V, E)$.

Output: A set of maximum cliques S.

$S \leftarrow \emptyset$, $max \leftarrow 0$
function GETMAXIMUMCLIQUES(graph G, current clique C, neighborhood of C N_C)
  if C is empty **then**
    for all $v \in V(G)$ **do**
      $C \leftarrow C \cup v$
      $GETMAXIMUMCLIQUES(G, C, N_v)$
      $C \leftarrow C \backslash v$
    end for
  else
    if $|C| > max$ **then** $S \leftarrow C$, $max \leftarrow |C|$
    else
      if $C = max$ **then** $S \leftarrow S \cup C$
      end if
    end if
    for all $v \in N_C$ **do**
      $C \leftarrow C \cup v$
      $V(G_{new}) \leftarrow \{u | u \in N_C \backslash v\}$
      $E(G_{new}) \leftarrow \{(i, j) | i, j \in V(G_{new}) \wedge (i, j) \in E(G)\}$

19: **if** $coloring(G_{new}) + |C| \leq max$ **then** $getmaximumcliques(G, C, N_C \cap N_v)$
20: **end if**
21: $C \leftarrow C \backslash v$
22: **end for**
23: **end if**
24: **end function**

4.2.2 Greedy vertex coloring of a graph

The greedy vertex coloring of a graph can be used as a pruning criterion for the tentative expansion of a clique. The chromatic number χ offers an upper-bound of growing for the clique expansion but the problem of determining the chromatic number of a graph is NP-complete [21] so the version implemented is a greedy one.

The strategy consists in coloring the vertices with higher degree first.

1. Sort in descending order the vertices of the graph according to their degree.

2. Color the first vertex of the list.

3. Color all the vertices not connected to the previously colored vertex with the same color if they do not have conflict coloring.

4. Repeat the process until every vertex is colored.

Algorithm 2 provides the pseudocode for the greedy vertex coloring algorithm.

Algorithm 2 A greedy algorithm for the vertex coloring of a graph.
Input: Graph $G = (V, E)$.

Output: Approximation of chromatic number.

```
1: function COLORING(G)
2:     Sort V in degree ascending order.
3:     current_color ← 0
4:     for all u ∈ V do
5:         if u is_not_colored then
6:             u_color ← current_color
7:             for all v ∈ V | (u, v) ∉ E do
8:                 if ∀w | (w ∈ V ∧ (v, w) ∈ E) (w_color ≠ current_color) then
9:                     v_color ← current_color
10:                end if
11:            end for
12:            current_color ← current_color + 1
13:        end if
14:    end for
15:    return color
16: end function
```

4.3 Exact algorithm for minimum clique covering

Exact algorithm for the minimum clique covering (ECC) requires the list of all cliques of the graph. With that, ECC obtains, in an optimal way, the set of cliques that include at least once each node of the graph. ECC models the problem as a decision of whether or not to include a clique in the set of the clique cover. Fixing cliques inside the cover may reduce the unique nodes provided by

others cliques but as they conserve at least one unique node they must be taken into account in the decision process.

The inclusion-exclusion of a clique in the solution can be represented as a decision tree shown in Figure 4.1.

– do not include in the clique.

+ include in the clique.

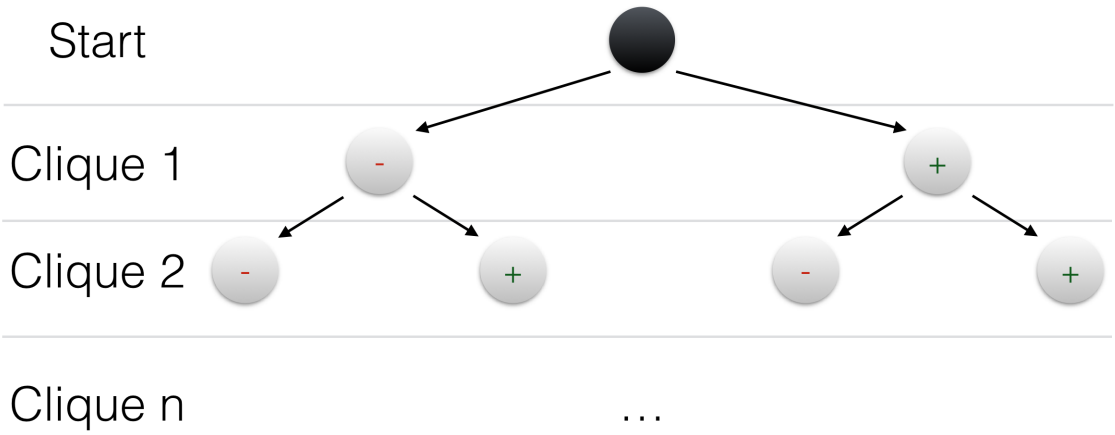


Figure 4.1: The decision tree to solve in an exact way the minimum clique covering.

A solution is achieved when the set of cliques included in the cover have at least one time each node of the graph. This solution could be used as a pruning criterion of the tree decision process. An strategy for parallelization of this algorithm consists in unwrapping the top of the decision tree to generate 2^n leaves, where n is the amount of fixed positions in the decision sequence.

For example, the string "– – –" could be mapped as a sequence where the initial three elements are not included in the clique cover. Having p processors, an approximate of fixed cliques in a sequence is $\log(p)$. Furthermore, solutions of size n that include only one clique to the cover are isomorphical, because the graph generated after the deletion of the nodes involved in the clique have the same characteristics no matter what clique is selected.

4.4 Greedy algorithm for minimum clique covering

Distinct approaches were tested as a greedy heuristic and four greedy versions were developed. The first and second versions require to list all the maximum cliques. The first version of the greedy algorithm evaluates the characteristics of the graph generated after the deletion of a desired clique. Every clique available in the list is tested, to determine what clique is going to be fixed, an objective function is maximized. This function consists in the summation of the square degree of the nodes that will remain available and the size of the clique that will be deleted, allowing to give priority to the deletion of cliques that would generate a graph with high probability to be covered with fewer cliques.

This criterion tries to avoid graphs with small degree nodes, a node with small degree may cause a clique with fewer nodes included that will be mapped to a row that covers a small number of t-way interactions.

Since the size of the list with all the maximum cliques grows exponentially, the second version of the greedy algorithm adds new criteria to fix a clique without exploring the whole list, these criteria fix a clique iff: a) the size of the clique is the largest coverage that a row provide, or b) the size of the clique is above the average of the previously added cliques to the clique cover. These new criteria in most instances may increase the number of cliques in the clique cover but the running time is decreased.

Despite the improvement in running time, iterating over the list with all the maximum cliques makes the overall process slower. The third version of the greedy algorithm does not use this list at all, it builds a maximum clique that only takes into account the neighborhood of a desired node. This algorithm Iteratively selects a base node v with a desired criteria, build a maximum clique including only the nodes $N_v = \{u | \{u, v\} \in E\}$ until all the nodes are included in a clique. The criteria for the selection of a node v could be the minimum or maximum of the summation of the degrees of the neighbors of node v, in practice better results are achieved when the base node minimizes

$\sum_{u \in N_v} deg(u)$. This algorithm provides a better running time but it a trade-off with the number of cliques generated, bigger number of cliques but in less time.

The last approach for the greedy algorithm tries to combine the ideas of the previous versions. This algorithm creates an induced graph G' with nodes randomly chosen of the ones that are not included in any clique yet. Select the maximum clique C of G' through a greedy heuristic, and perform the expansion of C using all the nodes of the original graph. This process can be repeated many times in order to obtain a set of tentative cliques to fix and determine which is the one that minimizes the *fixing heuristic*. The fixing heuristic combines the size of the clique and the degree of the nodes that are not included in any clique to calculate a representative value for the quality of the constructed clique. If the set of tentative cliques is large the constructed clique cover is better, but it is also slows the process of creation of the clique cover.

Since two processes are required to obtain a maximal clique it is not necessary to compute the maximum clique of the induced graph (line 9 of Algorithm 4) in an exact way. This heuristic construct a clique by iteratively appending the node that has the highest degree in the actual induced graph, recalculate the induced graph using the neighbors of the node selected and so on until there are no nodes to incorporate to the clique.

After the creation of a clique with the highest degree criterion, the algorithm performs an expansion of the clique using all the nodes available of the graph G (line 10 of Algorithm 4), the expansion computes all the possible ways of incorporate nodes of the set $CAND$ of candidates to the current clique in order to maximize the number of nodes in the clique. The set $CAND$ is calculated as the intersection of the neighborhood of each node in the clique.

This last greedy algorithm makes a trade-off between the running time invested to find a solution and the solution quality. It is possible to adjust this trade-off by increasing (for better quality) or decreasing (for faster execution) the value of $loop$ (in line 6 of Algorithm 4). This is not a deterministic algorithm despite the selected value for $loop$ several runs may not achieve the same result.

Algorithm 3 Pseudocode for the fixing heuristic.

1: **function** FIXING_HEURISTIC(C, G)

Input: A clique set C to fix and the graph G.

Output: Value for the fixing heuristic.

2: Put nodes C of G as unavailable.

3: $max \leftarrow 0$, $min \leftarrow \infty$, $valids \leftarrow 0$, $average \leftarrow 0$

4: **for all** $v \in G$ **do**

5: **if** v is available **then**

6: **if** $deg(v) < min$ **then** $min \leftarrow deg(v)$

7: **end if**

8: **if** $deg(v) > max$ **then** $max \leftarrow deg(v)$

9: **end if**

10: $valids \leftarrow valids + 1$

11: $average \leftarrow average + deg(v)$

12: **end if**

13: **end for**

14: Put nodes C of G as available.

15: **return** $|C| + (average/valids)/(max + 1)$

16: **end function**

Algorithm 4 A greedy algorithm for the minimum clique covering.

1: **function** GREEDY(G, $loop$)

Input: A graph $G = (V, E)$.

Output: A clique cover CC.

2: $missing \leftarrow |V|$

3: **while** $missing > 0$ **do**

4: $l \leftarrow 0$

5: $best \leftarrow 0$, $bestC = \emptyset$

6: **while** $l < loop$ **do**

7: Randomly select $min(missing, \binom{k}{t})$ vertices of V to build V'

8: $G' = (V', E' \leftarrow \{\{u, v\} | \{u, v\} \in E \wedge u, v \in V'\})$

9: $C \leftarrow greedyMaxClique(G')$

10: $C \leftarrow expand(C, G)$

11: **if** $best < fixing\_heuristic(C, G)$ **then**

12: $best \leftarrow fixing\_heuristic(C, G)$, $bestC \leftarrow C$

13: **end if**

14: **end while**

15: $CC \leftarrow CC \cup C$

16: Put nodes C as unavailable.

17: $missing \leftarrow missing - |C|$

18: **end while**

19: **return** CC

20: **end function**

4.5 Metaheuristic algorithm for minimum clique covering

To build clique covers with better quality, we proposed a simulated annealing algorithm for minimum clique covering (SACC). SACC receives a clique cover and perform operations to explore from the current solution to new ones in the search space in order to reduce the number of cliques needed to cover all the nodes of the graph.

To perform the exploration of the search space, it is necessary to develop neighborhood functions. These functions changes the composition of some cliques of the current solution to create a new solution that is evaluated to judge its quality, SA incorporates some criteria to allow the acceptance of solutions with lower quality depending on the current temperature of the system, at higher temperatures it accepts with higher probability a bad quality solution. The system is cooled down with the pass of iterations, forcing the acceptance of solutions that only makes improvement in their quality with regard to the current solution.

The quality of a clique cover is given by the following function:

$$f\_obj(CC) = |CC| + \left(1 - \frac{\sum_{c \in CC} |c|^2}{|CC| \cdot \binom{k}{t}^2}\right)$$

This functions is composed by an integer part, which evaluates the number of cliques of the clique cover and a decimal part in the range $[0, 1)$ that allows us to tear apart solutions with the same number of cliques but with different distribution of the sizes of each clique. The decimal part qualifies with a smaller number the clique covers that have cliques with small number of nodes that, in practice, are more suitable to be absorbed by another clique in the clique cover. Four neighborhood functions were developed to explore the search space that are described in the next subsections.

4.5.1 Neighborhood function 1: random coloring

This function gives randomness to the SACC, random coloring most of the time builds a larger number of cliques than the original but allows us to redistribute the nodes involved in some cliques in a different way. This function randomly selects 3 cliques O, P, Q that are in CC. The nodes included in that cliques forms the set of $missing$ that are used to generate a new set of cliques to substitute O, P, Q in the cover.

To re-include that nodes in cliques we take advantage of the equivalent problem of the clique cover, the *vertex coloring*. Using the graph complement, we randomly sort the set of missing to perform the vertex coloring of the nodes using a fast heuristic.

The coloring process uses a vector to keep tracking the available colors for each node, to coloring each node the algorithm verifies the previous colored nodes and marks as unavailable the colors that are used previously and are neighbors of the current node. If all the colors are unavailable a new color is assigned to the node.

To recreate the set of cliques, the nodes that are assigned to the same color form a clique to add to the cover, so the number of cliques added is equals to the greedy chromatic number.

4.5.2 Neighborhood function 2: greedy coloring

This function performs a quick coloring to perturb the current solution, that in most of the time generates a new solution with lesser number of cliques than the original or the same number but hardly generates a worst solution, it provides a slight change to the current solution.

This function randomly selects 5 cliques A, B, C, D, E that are in CC. The nodes included in that cliques forms the set of $missing$ that are used to generate a new set of cliques to substitute O, P, Q in the cover.

To re-include that nodes in cliques we take advantage of the equivalent problem of the clique cover, the *vertex coloring*. Using the graph complement, we perform the vertex coloring of the nodes

using the heuristic of highest degree first.

The coloring process consists in: a) Sort in descending order the vertices of the induced graph formed with the set of missing according to their degree, b) Color the first vertex of the list, c) Color all the vertices not connected to the previously colored vertex with the same color if they do not have conflict coloring, d) Repeat the process until every vertex is colored.

To recreate the set of cliques, the nodes that are assigned to the same color form a clique to add to the cover, so the number of cliques added is the greedy chromatic number.

4.5.3 Neighborhood function 3: disrupt a clique

This function attempts to redistribute all the nodes of one clique to a subset of cliques in the clique cover and provides a quick way to delete a clique that may fit partially in another clique in the clique cover.

This function randomly selects $\frac{|CC|}{2}$ cliques of the current cover to form the set Z.

A random clique of the set Z is selected as a $pivot$. The nodes included in the $pivot$ clique may fit in the remaining cliques of the set Z. The goal for this function is to decompose the $pivot$ clique to decrement its size. To do this, each node in the $pivot$ clique tries to fit in one clique in Z, if the node fits, it is deleted from the $pivot$ clique and added to the clique it fitted.

The new node configuration replaces the original set taken from CC.

4.5.4 Neighborhood function 4: fill a clique

This function redistributes the number of nodes of a subset of cliques in the clique cover. It tries to append the nodes included in this subset to only one clique, which performs a reduction in the number of nodes of some cliques but increases the number of nodes of only one clique.

This function randomly selects $\frac{|CC|}{4}$ cliques of the current cover to form the set Y.

A random clique of the set Y is selected as a $fixed$ clique. If the $fixed$ clique is not full (has $\binom{k}{t}$

nodes), it may accept some nodes that are included in the remaining cliques in Y. To do this, each node in each clique of the set Y is tested, if the node can expand the $fixed$ clique, it is translated from its original clique to the $fixed$ clique.

The new node configuration replaces the original set Y taken from CC.

4.5.5 Fine tuning for Simulated Annealing

Simulated annealing incorporate some parameters that have to be selected in the best possible way, such as the probability usage of the neighborhoods presented in the previous subsections, the cooling rate, and the number of iterations that the neighborhoods are tried for every temperature.

In order to tune the probabilities of usage of each neighborhood, we propose a tuning using the linear Diophantine equation:

$$F1 + F2 + F3 + F4 = 10$$

.

Where variables $F1, F2, F3$, and $F4$ represent the usage probability for each neighborhood with a granularity of 5.

For example, the values $F1 = 2, F2 = 2, F3 = 2, F4 = 4$ indicate that the function $F1$ has 20% of probability of being chosen, $F2$ has 20%, $F3$ has 20%, and $F3$ has 40%.

This tuning is done by selecting 3 representative instances, and testing 31 times each of the 286 different solutions of the above equation.

The proposed fine tuning leads to establish the probabilities for the usage of the neighborhood functions as: $F1 = 10$, $F2 = 10$, $F3 = 40$, and $F4 = 40$.

4.6 Graph Based Post-Optimization (GBPO)

Our graph-based post-optimization (GBPO) approach deletes, in an exhaustive way, a set of several rows of a CA or partial CA (PCA), maps the missing t-way interactions to a graph that represents the pairwise compatibility between them, and rebuilds a set of rows (with the same or less size than the deleted set) to provide a greater or equal coverage than the original CA (or PCA). The use of a graph provides an easy way to remember sets of t-way interactions that can be added in the same row of a CA.

This subsection describes GBPO approach, it is based on three processes: a) MAPPER process, that moves the problem to the graph domain; b) MAX process, that finds complete subgraphs with the maximal number of nodes, i.e. maximal cliques; c) HANDLER process, that orchestrates the post-optimization in the graph domain and provides the final result in the CA domain.

4.6.1 MAPPER process

Any covering array CA$(N; t, k, v)$ provides a total of $v^t\binom{k}{t}$ distinct *t-way interactions*. Every row provides at most $\binom{k}{t}$ t-way interactions, so a CA provides $N \cdot \binom{k}{t}$ interactions, a trivial case of optimality exists when $N = v^t$ (an orthogonal array of index unity). When $N > v^t$, some interactions could be covered in more than one rows, meanwhile some others could be covered only once. It is possible that regardless of the value of some cells the coverage still stands, those cells are known as wildcard or flexible positions. A PCA is a covering array with missing interactions.

The MAPPER process generates a graph $G = \{V, E\}$ where $|V|$ is the number of missing interactions in a PCA, and $|E|$ is the number of pairs of missing interactions that are compatible (see Definition 11). An edge between two nodes exists iff the corresponding missing interactions are compatible. This way the process of mapping a PCA to the graph domain is very easy to implement.

4.6.2 MAX process

The main work of this process involves the determination of a complete subgraph of maximal size (a complete subgraph between a set of nodes requires the existence of an edge for each pair of nodes belonging to the complete subgraph). A maximal complete subgraph is called a maximal clique. A maximal clique represents in the CA domain a row that covers all the missing interactions corresponding to its nodes. The MAX process receives as input parameters: a PCA; ι that indicates the number of input rows; ϕ that defines the number of output rows; and ϵ for the number of excluded rows.

At high level the operation of this process deletes from the input CA ι rows (excluding the first ϵ rows), and adds to the input CA ϕ new rows, so if the input is a CA$(N; t, k, v)$ the output will be a CA$(N + \phi - \iota; t, k, v)$ with less or same missing interactions than the input CA. This process requires the generation of a graph that covers the missing interactions in a CA$(N - \iota; t, k, v)$, the determination of ϕ maximal cliques in the graph and the construction of ϕ rows (corresponding to the ϕ maximal cliques) giving finally a CA$(N + \phi - \iota; t, k, v)$. As an important note, the reason to avoid searching the maximum clique is that getting the largest amount of missing interactions in one row naturally attempts to recreate the same rows that were previously deleted.

Given a PCA$(N; t, k, v)$ and ι, ϕ, and ϵ parameters, the MAX process does:

1. Find the missing interactions in the first PCA and construct the list O with them.
2. Let Ω be a set with every possible set of ι different rows of the PCA excluding the first ϵ rows.
3. For each element of $\omega \in \Omega$.
   (a) Analyze the interactions of each of the rows in ω to construct the list M with the interactions provided uniquely by the rows in ω.
   (b) With both lists M and O build a graph G where V is each element in both lists, and the set of edges is generated by checking the compatibility of the set of nodes.

(c) Generate the ϕ maximal cliques and construct ϕ rows

(d) Calculate ξ as the interactions that are not in any clique.

4. Choose the option that minimizes ξ (the number of missing interactions).

It is possible to improve the quality[1] of a CA by applying this process iteratively, with stop criterion such as the number of iterations without improvement.

For example, given the parameters $\iota = 1, \phi = 1$, $\epsilon = 0$ and the next PCA with parameters $t = 2$, $k = 4$, and $v = 2$

0	1	0	0
1	1	1	0
0	0	1	0
1	0	0	0
1	0	1	1

if the number of columns is denoted by the alphabet $a, b, c, \ldots$, there are three missing tuples: $a0d1$, $b1d1$, and $c0d1$.

Because $\iota = 1$ the process will try the $\binom{N}{1}$ ways to delete one row. Starting with the deletion of the first row, the t-way interactions provided uniquely by this row are $a0b1$, $a0c0$, and $b1c0$. Using the definition of *compatibility* it is possible to construct the edges and nodes of a graph using the original missing interactions plus the ones provided uniquely by the first row. Figure 4.2 shows the graph generated where the number of columns is represented with letters a, b, and c.

[1] Output CA with less or equal number of rows than input CA, and missing interactions equal or reduced.

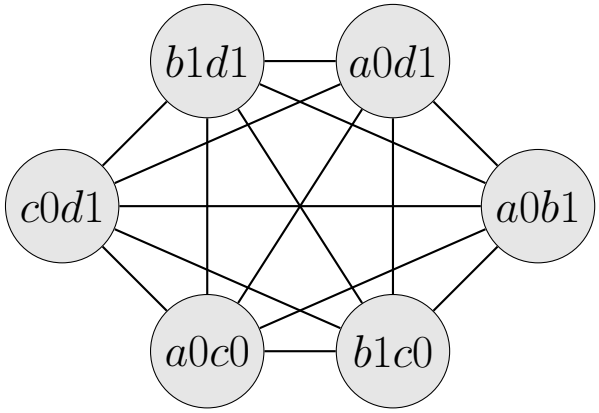


Figure 4.2: Graph generated in the MAX process of the post-optimization algorithm.

Since $\phi = 1$, just one maximal clique needs to be obtained. So, no matter what interaction was randomly selected the resulting maximal clique is $\{a0b1, a0c0, a0d1, b1c0, b1d1, c0d1\}$, leaving a remaining of zero missing interactions. The row generated with that maximal clique is $0, 1, 0, 1$, and forms the next valid CA

0	1	0	1
1	1	1	0
0	0	1	0
1	0	0	0
1	0	1	1

the process stops whenever it reaches zero missing interactions, after that, the HANDLER process (described in the next subsection) would try to delete another row, but the remaining 4 rows would reach the stop criterion without finding a CA$(4; 2, 4, 2)$, which does not exist.

4.6.3 HANDLER process

The HANDLER process orchestrates the CA post-optimization. This process defines the parameters used for calling the MAX process, and uses the obtained results for further calls. the flow chart in Figure 4.3 show the overall operation of the HANDLER process. The process receives a CA or PCA, the initial count of missing t-way interactions ξ, the number of iterations allowed in the external loop of the graph solver process tol, and the number of maximum iterations without improvement max

used as a stop criterion. The number of rows for ι and ϕ parameter values was fixed as $f = \{1, 2, 3\}$. Since the outer loop of the HANDLER process could take too much time to execute, a mechanism to control its execution time is needed. This mechanism uses the parameter tol to calculate the number ϵ of rows excluded from evaluation. For a specific value of rows involved f, ϵ is computed as follows:

1. Calculate the number of iterations of the external loop $num\_ops = \binom{N}{f}$.

   (a) if $num\_ops \leq tol$, $\epsilon = 0$.

   (b) if $num\_ops > tol$, $\epsilon = \left(N - \frac{N}{(num\_ops/tol)^{1/f}}\right)$.

This process has two variants: when there are no missing t-way interactions ($\xi = 0$) after updating the partial CA through the MAX process, the HANDLER immediately calls the MAX process to delete $\iota = 2$ rows and build $\phi = 1$ rows which generates a new PCA; and when $\xi > 0$ the HANDLER calls the MAX process trying to decrease the amount of missing interactions until the stop criterion is met.

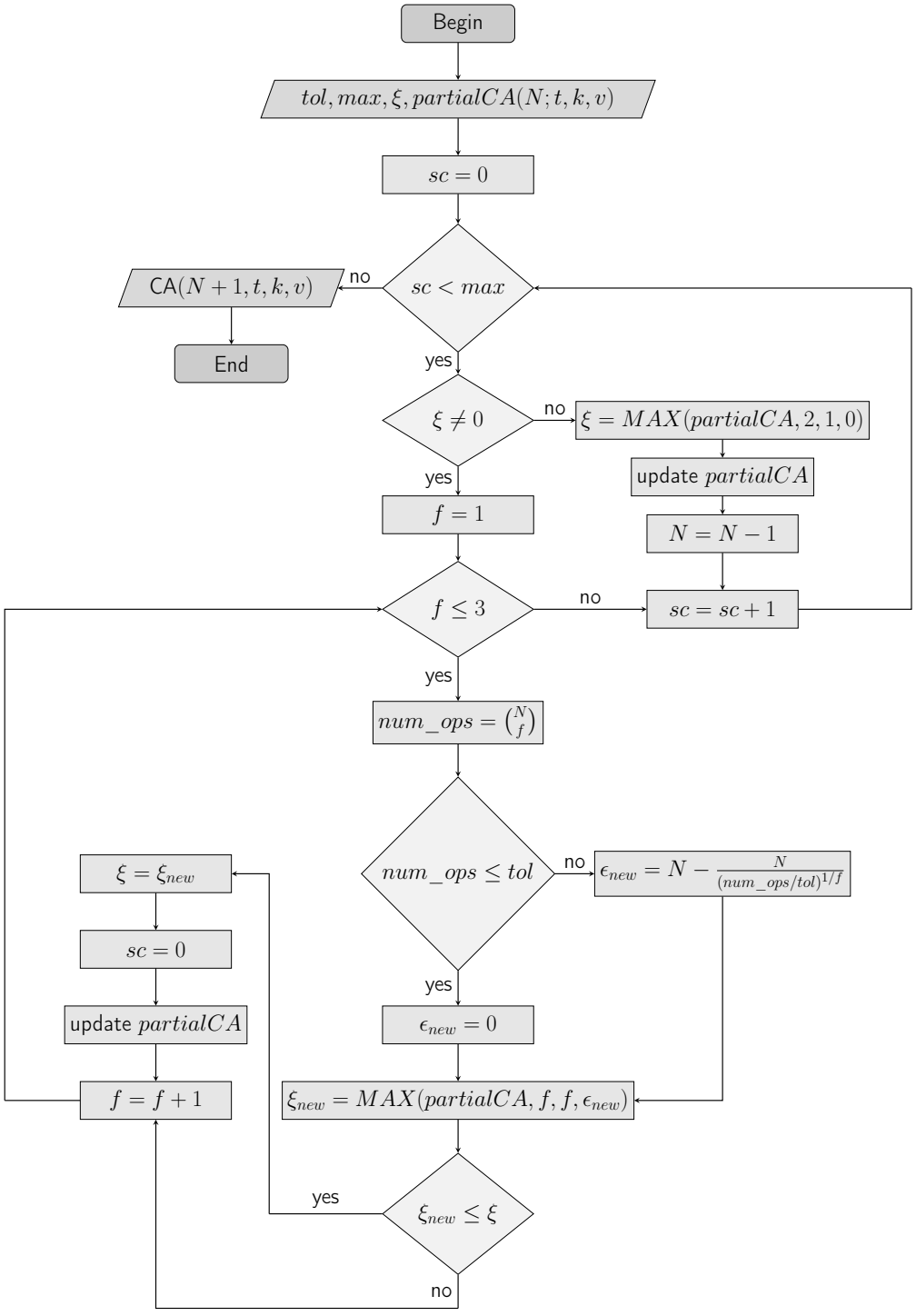


Figure 4.3: Flow diagram of the HANDLER process of the post-optimization algorithm.

4.7 Summary

This chapter presented the methodology used to solve the mapped problem of the CACP to the graph domain, three distinct approaches were developed: a) an exact algorithm to find optimal results for small graph instances; b) a set of four greedy algorithms to solve bigger graph instances in short time; c) a metaheuristic algorithm that improve in terms of quality the results obtained for the greedy algorithm. These approaches solve the minimum clique covering problem which is the equivalent problem to the CACP in the graph domain using the Coverage in Nodes representation, and consists in including every node into a clique where the number of cliques is minimal.

Also a graph-based post-optimization algorithm is presented to reduce the number of rows of a CA whenever its possible. In the next chapter we present the experimentation done for each algorithm proposed in this methodology and is analyzed the results obtained by our solution methods.

5

Experimentation and results

This chapter presents the experimentation done and results obtained with the proposed algorithms for the minimum clique covering, the proposed experimentation consists of: a) a set of 7 graph instances where $|V| = \binom{k}{t} v^t \leq 90$ for the exact algorithm, to prove that our exact approach construct optimal CAs, where we select small graph instances due to the fact that running time for this algorithm grows exponentially, b) a set of 95 medium size graph instances for the greedy algorithm, to show the performance of our greedy approach versus a greedy approach of the state-of-the-art, c) a set of 33 medium size graph instances for the metaheuristic algorithm, to show that our metaheuristic approach build clique covers with better quality than our greedy approach, and d) a set of 560 CAs previously constructed, to show that our post-optimization algorithm reduces the number of rows of the CAs. Results obtained are presented grouped by the algorithm used, and each approach applied is compared to a respective algorithm of the state-of-the-art.

5.1 Results with the exact algorithm

Experimentation for the exact algorithm was conducted for a small set of 7 instances of CAs of strength $t = 2$ where the amount of nodes $|V| = \binom{k}{t} v^t \leq 90$, results are shown in Table 5.1. The exact algorithm construct 7 CAs that matched the best-known upper bounds and these results are optimal.

Instance	|V|	|E|	Density	ECC	BUP[1] [10]
CA(2,3,2)	12	24	0.36363	4	4
CA(2,4,2)	24	144	0.52173	5	5
CA(2,5,2)	40	480	0.61538	6	6
CA(2,6,2)	60	1200	0.67796	6	6
CA(2,3,3)	27	81	0.23076	9	9
CA(2,4,3)	54	567	0.39622	9	9
CA(2,5,3)	90	2025	0.50561	11	11

Table 5.1: Results obtained for the experimentation conducted for the exact algorithm.

5.2 Results with the greedy algorithms

Experimentation for the first three greedy algorithms was conducted with a set of instances of CAs of strength $t = 2$ where the number of nodes is ≤ 540. Table 5.2 shows results for the four approaches. Despite the difference between the third approach in its two variants with the first and second approaches in terms of quality, the running time is substantially lower. G and Gv2 are competitive versus the state-of-the-art algorithms for $v = 2$ and for $v > 2$ outperforms IPOG-F [15] in most of the instances.

The last version of the greedy algorithm allows us to build solutions for larger instances, Tables 5.3 and 5.4 show the obtained results for 95 instances, where 18 instances matched the upper bounds of a state of the art greedy algorithm and 77 instances improved the upper bounds of the IPOG-F algorithm. As the strength and vocabulary grow, our greedy approach outperforms IPOG-F [15].

Results that match the best-known upper bounds are shown in bold.

v2t2

k	|V|	|E|	Density	G	Gv2	Gv3-Min	Gv3-Max	IPOG-F [15]
4	24	144	0.52173	6	6	6	6	6
5	40	480	0.61538	6	7	7	8	6
6	60	1200	0.67796	6	8	6	8	6
7	84	2520	0.72289	6	8	8	8	6
8	112	4704	0.75675	8	8	7	12	6
9	144	8064	0.78321	8	9	7	10	6
10	180	12960	0.80446	8	9	9	10	8
11	220	19800	0.82191	9	10	8	12	8
12	264	29040	0.83650	9	9	9	10	8
13	312	41184	0.84887	10	9	10	12	8
14	364	56784	0.85950	8	9	10	10	8
15	420	76440	0.86873	9	10	11	11	8

v3t2

k	|V|	|E|	Density	G	Gv2	Gv3-Min	Gv3-Max	IPOG-F [15]
3	27	81	0.23076	10	9	13	9	10
4	54	567	0.39622	12	12	13	17	12
5	90	2025	0.50561	15	13	13	18	13
6	135	5265	0.58208	14	15	15	19	15
7	189	11340	0.63829	15	16	14	15	15
8	252	2154	0.68108	17	17	17	21	15
9	324	37422	0.71517	16	17	20	23	17
10	405	60750	0.74257	19	18	18	23	19

v4t2

k	|V|	|E|	Density	G	Gv2	Gv3-Min	Gv3-Max	IPOG-F [15]
3	48	192	0.17021	17	16	25	23	17
4	96	1536	0.33684	20	20	23	30	20
5	160	5760	0.45283	22	16	24	29	22
6	240	15360	0.53556	23	28	23	29	24
7	336	33600	0.59701	26	25	25	28	27
8	448	64512	0.64429	27	28	29	31	27

v5t2

k	|V|	|E|	Density	G	Gv2	Gv3-Min	Gv3-Max	IPOG-F [15]
3	75	375	0.13513	25	26	39	36	26
4	150	3375	0.30201	29	30	34	47	30
5	250	13125	0.42168	31	31	34	43	33
6	375	35625	0.50802	35	34	39	40	38
7	525	78750	0.57251	39	36	41	44	41

v6t2

k	|V|	|E|	Density	G	Gv2	Gv3-Min	Gv3-Max	IPOG-F [15]
3	108	648	0.11214	38	38	57	51	38
4	216	6480	0.27906	44	43	49	69	43
5	360	25920	0.40111	48	46	50	60	46
6	540	71280	0.48979	49	49	58	57	52

Table 5.2: Results obtained of the experimentation conducted for the first three versions of the greedy algorithm.

v3t2							v5t2						
k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta	k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta
3	27	81	0.23077	9	10	1	3	75	375	0.135135	26	26	0
4	54	567	0.39623	9	12	3	4	150	3375	0.302013	29	30	1
5	90	2025	0.50562	12	13	1	5	250	13125	0.421687	31	33	2
6	135	5265	0.58209	14	15	1	6	375	35625	0.508021	34	38	4
7	189	11340	0.63830	15	15	0	7	525	78750	0.572519	38	41	3
8	252	21546	0.68127	15	15	0	8	700	152250	0.622318	40	41	1
9	324	37422	0.71517	16	17	1	9	900	267750	0.661846	42	44	2
10	405	60750	0.74257	16	19	3	10	1125	438750	0.693950	44	45	1
11	495	93555	0.76518	18	19	1	11	1375	680625	0.720524	46	46	0
12	594	138105	0.78415	18	20	2	12	1650	1010625	0.742874	48	48	0
13	702	196911	0.80029	18	20	2	v6t2						
14	819	272727	0.81418	19	20	1	k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta
15	945	368550	0.82627	19	20	1	3	108	648	0.112150	37	38	1
16	1080	487620	0.83689	20	20	0	4	216	6480	0.279070	42	43	1
17	1224	633420	0.84628	20	20	0	5	360	25920	0.401114	45	46	1
18	1377	809676	0.85465	20	20	0	6	540	71280	0.489796	50	52	2
19	1539	1020357	0.86216	21	21	0	7	756	158760	0.556291	52	54	2
20	1710	1269675	0.86893	21	21	0	8	1008	308448	0.607746	56	57	1
v4t2							9	1296	544320	0.648649	59	59	0
k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta	10	1620	894240	0.681902	62	63	1
3	48	192	0.170213	16	17	1	v2t3						
4	96	1536	0.336842	16	20	4	k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta
5	160	5760	0.452830	17	22	5	4	32	96	0.193548	8	9	1
6	240	15360	0.535565	23	24	1	5	80	960	0.303797	11	11	0
7	336	33600	0.597015	25	27	2	6	160	4960	0.389937	12	14	2
8	448	64512	0.644295	26	27	1	7	280	17920	0.458781	12	16	4
9	576	112896	0.681739	28	29	1	8	448	51520	0.514541	14	17	3
10	720	184320	0.712100	28	29	1	9	672	126336	0.560358	16	17	1
11	880	285120	0.737201	31	31	0	10	960	275520	0.598540	16	18	2
12	1056	422400	0.758294	31	31	0	11	1320	549120	0.630781	18	18	0
13	1248	604032	0.776263	33	33	0	12	1760	1019040	0.658329	18	19	1
14	1456	838656	0.791753	33	33	0							
15	1680	1135680	0.805241	34	34	0							

Table 5.3: Results obtained of the experimentation conducted for the last version of the greedy algorithm. Part 1 of 2

v3t3							v3t4						
k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta	k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta
4	108	486	0.08411	29	34	5	7	2835	433755	0.10797	156	164	8
5	270	6075	0.16729	39	42	3	v4t4						
6	540	36450	0.25046	45	49	4	k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta
7	945	144585	0.32415	49	52	3	5	1280	10240	0.01251	292	323	31
8	1512	442260	0.38716	54	56	2	6	3840	245760	0.03334	415	442	27
9	2268	1132866	0.44067	58	62	4	7	8960	2652160	0.06608	491	530	39
v4t3							v5t4						
k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta	k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta
4	256	1536	0.04706	67	76	9	5	3125	31250	0.00640	742	793	51
5	640	23040	0.11268	86	94	8	6	9375	890625	0.02027	974	1059	85
6	1280	156160	0.19077	103	109	6	v6t4						
7	2240	663040	0.26440	118	124	6	k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta
v5t3							5	6480	77760	0.00370	1548	1628	80
k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta	v2t5						
4	500	3750	0.03006	138	148	10	k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta
5	1250	65625	0.08407	173	181	8	6	192	960	0.05236	32	42	10
6	2500	493750	0.15806	202	209	7	7	672	20160	0.08942	52	57	5
v6t3							8	1792	206080	0.12842	63	68	5
k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta	9	4032	1370880	0.16869	76	77	1
4	864	7776	0.02086	238	259	21	v3t5						
5	2160	155520	0.06670	296	310	14	k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta
6	4320	1283040	0.13753	343	361	18	6	1458	10935	0.01030	273	318	45
v2t4							7	5103	306180	0.02352	438	467	29
k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta	v4t5						
5	80	320	0.10127	16	22	6	k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta
6	240	4800	0.16736	25	26	1	6	6144	61440	0.00326	1283	1377	94
7	560	35840	0.22898	29	32	3	v2t6						
8	1120	179200	0.28597	32	34	2	k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta
9	2016	685440	0.33747	34	37	3	7	448	2688	0.02685	65	79	14
v3t4							8	1792	75264	0.04690	115	118	3
k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta	v3t6						
5	405	2430	0.02970	87	98	11	k	|V|	|E|	Density	Gv4	IPOG-F [15]	Delta
6	1215	47385	0.06425	133	140	7	7	5103	45927	0.00353	895	990	95
7	2835	433755	0.10797	156	164	8							

Table 5.4: Results obtained of the experimentation conducted for the last version of the greedy algorithm. Part 2 of 2

5.3 Results with the metaheuristic algorithm SACC

To conduct the experimentations for the SACC algorithm, 33 CA instances of $t = 2$ and $v = \{3, 4, 5, 6\}$ were selected. The parameters for usage probability of each function were settled as: $F1 = 10\%$, $F1 = 10\%$, $F3 = 40\%$, $F4 = 40\%$, and the cooling rate factor to $\alpha = 0.9$. Table 5.5 show that 17 of the 33 instances matched the best-known upper bounds and 4 instances are improved. Results that matched or improved the best-known bounds are shown in bold.

v3t2							v5t2						
k	|V|	|E|	Density	SACC	Best-known [10]	Delta	k	|V|	|E|	Density	SACC	Best-known [10]	Delta
3	27	81	0.23077	9	9	0	3	75	375	0.13514	25	25	0
4	54	567	0.39623	9	9	0	4	150	3375	0.30201	25	25	0
5	90	2025	0.50562	11	11	0	5	250	13125	0.42169	25	25	0
6	135	5265	0.58209	11	11	0	6	375	35625	0.50802	25	25	0
7	189	11340	0.63830	12	12	0	7	525	78750	0.57252	33	29	-4
8	252	21546	0.68127	14	12	-2	8	700	152250	0.62232	36	33	-3
9	324	37422	0.71517	15	13	-2	9	900	267750	0.66185	39	35	-4
10	405	60750	0.74257	15	14	-1	10	1125	438750	0.69395	40	36	-4
11	495	93555	0.76518	15	15	0	11	1375	680625	0.72052	43	37	-6
v4t2							v6t2						
k	|V|	|E|	Density	SACC	Best-known [10]	Delta	k	|V|	|E|	Density	SACC	Best-known [10]	Delta
3	48	192	0.17021	16	16	0	3	108	648	0.11215	36	36	0
4	96	1536	0.33684	16	16	0	10	1620	894240	0.68190	48	48	0
5	160	5760	0.45283	16	16	0	11	1980	1389960	0.70945	50	51	1
6	240	15360	0.53556	19	19	0	12	2376	2067120	0.73263	52	53	1
7	336	33600	0.59701	22	21	-1	13	2808	2965248	0.75240	54	55	1
8	448	64512	0.64430	23	21	-2	14	3276	4127760	0.76947	56	57	1
9	576	112896	0.68174	25	22	-3	18	5508	12426048	0.81932	57	57	0
10	720	184320	0.71210	26	22	-4							

Table 5.5: Results of the experimentation conducted for the SACC algorithm.

5.4 Results with the GBPO algorithm

To conduct the experimentation for the GBPO algorithm we selected a set of 30 CAs with number of columns in the range of $3 \leq k \leq 32$ for the alphabets $2 \leq v \leq 6$ of strength $t = 2$, a set of 30 CAs with number of columns in the range of $4 \leq k \leq 33$ for the alphabets $2 \leq v \leq 6$ of strength $t = 3$, a set of 20 CAs with number of columns in the range of $5 \leq k \leq 24$ for the alphabets $2 \leq v \leq 6$ of strength $t = 4$, a set of 20 CAs with number of columns in the range of $6 \leq k \leq 25$ for the alphabets $2 \leq v \leq 6$ of strength $t = 5$, and a set of 15 CAs with number of columns in the range of $7 \leq k \leq 16$ for the alphabets $2 \leq v \leq 5$ of strength $t = 6$, for a total of 560 CAs generated by the algorithm IPOG-F developed by Forbes et al. [15] . In Tables 5.6 to 5.10 the obtained results of our graph based post-optimization approach (GBPO) are shown, results that match the best-known bounds results registered in Colbourn tables [10] are shown in bold. Compared to the IPOG-F, every tested instance is improved.

Additionally, a set of 32 instances of strength $t = 2$ were selected to perform a comparison with the results obtained by the algorithm proposed by Li et al. [36]. Table 5.11 shows that every result is matched and 16 results are improved. The parameters selected for both experimentations were, $tol = 200000$, $f = 3$, $max = 200$. Since in some cases the parameter tol limits the number of rows immerse in the graph solving process, the performing of a random selection of the rows that constitute part of the process of elimination-addition of rows pays off with better results.

The Figures 5.1 to 5.5 shows the behavior of the obtained results by the GBPO of CAs with strength $2 \leq t \leq 6$, the y-axis represents the number of rows in logarithmic scale, and the x-axis represents the number of columns of the CAs.

$t = 2$

	$v = 2$			$v = 3$			$v = 4$			$v = 5$			$v = 6$		
k	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ
3	4	**4**	0	10	**9**	1	17	**16**	1	26	**25**	1	38	**36**	2
4	6	**5**	1	12	**9**	3	20	**16**	4	30	**25**	5	43	38	5
5	6	**6**	0	13	**11**	2	22	**16**	6	33	**25**	8	46	43	3
6	6	**6**	0	15	**12**	3	24	20	4	38	**25**	13	52	45	7
7	6	**6**	0	15	**12**	3	27	22	5	41	34	7	54	48	6
8	6	**6**	0	15	**13**	2	27	23	4	41	35	6	57	51	6
9	6	**6**	0	17	**13**	4	29	24	5	44	37	7	59	53	6
10	8	**6**	2	19	15	4	29	25	4	45	38	7	63	56	7
11	8	**7**	1	19	15	4	31	26	5	46	41	5	64	56	8
12	8	**7**	1	20	15	5	31	27	4	48	41	7	67	59	8
13	8	**7**	1	20	15	5	33	27	6	49	43	6	68	62	6
14	8	**7**	1	20	16	4	33	28	5	50	44	6	68	63	5
15	8	**7**	1	20	16	4	34	29	5	51	45	6	73	65	8
16	8	**8**	0	20	16	4	35	29	6	51	47	4	75	66	9
17	10	**8**	2	20	17	3	36	30	6	54	47	7	75	68	7
18	10	**8**	2	20	17	3	36	30	6	54	48	6	75	68	7
19	10	**8**	2	21	18	3	37	31	6	54	49	5	75	70	5
20	10	**8**	2	21	18	3	37	32	5	56	49	7	79	71	8
21	10	**8**	2	21	18	3	37	32	5	56	51	5	79	72	7
22	10	**8**	2	21	18	3	38	33	5	57	52	5	80	73	7
23	10	**8**	2	21	19	2	40	33	7	59	52	7	83	75	8
24	10	**8**	2	21	19	2	40	34	6	59	53	6	83	75	8
25	10	**8**	2	21	19	2	40	34	6	59	54	5	83	76	7
26	10	**8**	2	23	19	4	41	35	6	59	54	5	85	78	7
27	10	**8**	2	23	19	4	41	35	6	62	54	8	85	78	7
28	10	**8**	2	23	20	3	41	35	6	62	56	6	85	79	6
29	10	**8**	2	23	20	3	41	36	5	62	56	6	85	80	5
30	11	**8**	3	23	20	3	41	36	5	63	57	6	86	80	6
31	11	**8**	3	23	21	2	41	36	5	63	57	6	89	82	7
32	11	**8**	3	23	21	2	41	37	4	63	58	5	90	83	7

Table 5.6: Results obtained of the experimentation conducted for the GBPO algorithm with instances of strength $t = 2$.

$t = 3$

	$v = 2$			$v = 3$			$v = 4$			$v = 5$			$v = 6$		
k	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ
4	9	**8**	1	34	**27**	7	76	68	8	148	133	15	259	243	16
5	11	**10**	1	42	**33**	9	94	86	8	181	165	16	310	291	19
6	14	**12**	2	49	35	14	109	100	9	209	192	17	361	340	21
7	16	**12**	4	52	44	8	124	110	14	236	219	17	400	384	16
8	17	**12**	5	56	50	6	135	121	14	257	239	18	437	419	18
9	17	**12**	5	62	53	9	146	132	14	276	255	21	476	457	19
10	18	**12**	6	66	56	10	155	139	16	299	279	20	507	485	22
11	18	**12**	6	68	59	9	163	150	13	312	295	17	537	511	26
12	19	**15**	4	71	63	8	170	156	14	329	315	14	564	545	19
13	20	17	3	76	66	10	177	163	14	344	329	15	589	567	22
14	21	18	3	77	69	8	183	169	14	355	341	14	613	593	20
15	21	18	3	80	71	9	188	175	13	367	354	13	633	613	20
16	22	18	4	82	73	9	194	182	12	380	361	19	652	630	22
17	24	19	5	85	75	10	199	187	12	393	377	16	670	650	20
18	24	20	4	88	77	11	202	193	9	401	389	12	689	677	12
19	24	20	4	91	79	12	211	197	14	412	396	16	707	692	15
20	25	21	4	92	81	11	215	203	12	420	406	14	724	704	20
21	25	22	3	93	85	8	218	206	12	426	417	9	739	720	19
22	26	22	4	93	85	8	223	209	14	435	424	11	756	737	19
23	26	23	3	94	86	8	227	217	10	444	431	13	769	750	19
24	26	24	2	98	87	11	232	218	14	454	440	14	785	765	20
25	27	24	3	98	90	8	236	223	13	461	449	12	798	777	21
26	27	25	2	99	91	8	239	228	11	468	456	12	812	797	15
27	28	25	3	100	93	7	243	231	12	480	463	17	825	804	21
28	28	25	3	101	94	7	247	235	12	482	467	15	840	823	17
29	28	26	2	103	95	8	252	238	14	486	477	9	848	832	16
30	28	26	2	106	96	10	255	244	11	495	482	13	859	839	20
31	29	27	2	106	97	9	259	245	14	505	489	16	870	856	14
32	31	27	4	107	98	9	259	248	11	511	492	19	882	865	17
33	31	27	4	108	99	9	262	253	9	516	498	18	891	878	13

Table 5.7: Results obtained of the experimentation conducted for the GBPO algorithm with instances of strength $t = 3$.

$t = 4$

	$v = 2$			$v = 3$			$v = 4$			$v = 5$			$v = 6$		
k	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ
5	22	**16**	6	98	86	12	323	301	22	793	739	54	1628	1604	24
6	26	**21**	5	140	124	16	442	408	34	1059	1012	47	2165	2124	41
7	32	**24**	8	164	148	16	530	500	30	1280	1222	58	2628	2564	64
8	34	**24**	10	188	168	20	599	566	33	1460	1430	30	3007	2974	33
9	37	**24**	13	211	189	22	669	646	23	1638	1611	27	3374	3330	44
10	41	**24**	17	228	210	18	725	702	23	1791	1755	36	3713	3676	37
11	43	38	5	248	226	22	784	755	29	1941	1909	32	4011	3955	56
12	47	40	7	262	244	18	835	819	16	2068	2031	37	4295	4249	46
13	49	42	7	277	256	21	893	870	23	2193	2168	25	4553	4506	47
14	52	44	8	288	276	12	933	914	19	2321	2276	45	4800	4757	43
15	53	47	6	302	286	16	976	962	14	2424	2398	26	5024	4991	33
16	56	48	8	316	300	16	1018	1004	14	2528	2486	42	5248	5199	49
17	57	51	6	328	311	17	1060	1045	15	2627	2590	37	5449	5396	53
18	60	53	7	336	325	11	1098	1077	21	2725	2697	28	5650	5603	47
19	62	55	7	347	336	11	1131	1110	21	2805	2793	12	5841	5800	41
20	64	58	6	358	346	12	1165	1149	16	2892	2870	22	6015	5972	43
21	68	59	9	369	354	15	1201	1183	18	2982	2964	18	6186	6144	42
22	69	60	9	380	364	16	1234	1218	16	3066	3025	41	6352	6320	32
23	70	62	8	388	376	12	1266	1250	16	3133	3122	11	6508	6474	34
24	71	64	7	396	383	13	1298	1281	17	3203	3165	38	6662	6623	39

Table 5.8: Results obtained of the experimentation conducted for the GBPO algorithm with instances of strength $t = 4$.

$t = 5$

	$v = 2$			$v = 3$			$v = 4$			$v = 5$			$v = 6$		
k	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ
6	42	**32**	10	318	266	52	1377	1276	101	4195	4018	177	10407	10112	295
7	57	**42**	15	467	432	35	1966	1890	76	5942	5718	224	14712	14025	687
8	68	**52**	16	557	513	44	2406	2334	72	7349	7135	214	18266	17571	695
9	77	65	12	652	607	45	2795	2723	72	8629	8383	246	21474	20894	580
10	87	74	13	738	689	49	3181	3117	64	9796	9645	151	24394	23911	483
11	95	83	12	815	771	44	3521	3471	50	10862	10712	150	27151	26682	469
12	105	91	14	885	850	35	3854	3787	67	11889	11775	114	29645	29340	305
13	111	101	10	957	931	26	4158	4094	64	12851	12739	112	32076	31806	270
14	119	104	15	1021	993	28	4454	4384	70	13748	13593	155	34309	34102	207
15	127	104	23	1080	1049	31	4718	4673	45	14578	14457	121	36429	36216	213
16	134	104	30	1140	1106	34	4970	4925	45	15379	15223	156	38408	38248	160
17	140	130	10	1190	1170	20	5214	5176	38	16128	15997	131	40334	40207	127
18	144	134	10	1241	1214	27	5458	5408	50	16843	16738	105	42102	42019	83
19	148	137	11	1293	1259	34	5679	5621	58	17516	17420	96	43833	43763	70
20	155	142	13	1342	1306	36	5884	5828	56	18171	18039	132	45425	45376	49
21	160	147	13	1386	1359	27	6085	6032	53	18779	18720	59	46970	46933	37
22	163	152	11	1432	1407	25	6273	6226	47	19387	19265	122	48479	48462	17
23	168	158	10	1475	1450	25	6443	6406	37	19941	19841	100	49924	49908	16
24	175	165	10	1513	1490	23	6629	6600	29	20482	20385	97	51287	51280	7
25	181	168	13	1547	1530	17	6805	6759	46	21004	20902	102	52604	52580	24

Table 5.9: Results obtained of the experimentation conducted for the GBPO algorithm with instances of strength $t = 5$.

$t = 6$

	$v = 2$			$v = 3$			$v = 4$			$v = 5$		
k	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ	IPOG-F	GBPO	Δ
7	79	**64**	15	990	861	129	5761	5472	289	22100	21299	801
8	118	**85**	33	1490	1432	58	8579	8060	519	32822	31282	1540
9	142	122	20	1847	1774	73	10724	10239	485	41210	40447	763
10	165	148	17	2190	2130	60	12713	12350	363	49111	48505	606
11	192	170	22	2512	2448	64	14661	14232	429	56615	56164	451
12	215	195	20	2815	2765	50	16446	16106	340	63620	63245	375
13	237	214	23	3106	3048	58	18136	17865	271	70190	69855	335
14	256	234	22	3358	3310	48	19739	19485	254	76390	76113	277
15	276	256	20	3623	3553	70	21215	21029	186	82139	81944	195
16	292	275	17	3863	3824	39	22608	22432	176	87559	87338	221
17	309	291	18	4095	4048	47	23947	23736	211	92701	92543	158
18	327	309	18	4310	4254	56	25212	25095	117	97605	97473	132
19	343	323	20	4509	4469	40	26392	26264	128	102208	101970	238
20	363	346	17	4701	4658	43	27534	27374	160	110842	106558	4284
21	375	356	19	4890	4866	24	28625	28521	104	114775	110703	4072

Table 5.10: Results obtained of the experimentation conducted for the GBPO algorithm with instances of strength $t = 6$.

$t = 2$

$v = 3$				$v = 4$				$v = 5$				$v = 6$			
k	Li et al.[36]	GBPO	Δ	k	Li et al.[36]	GBPO	Δ	k	Li et al.[36]	GBPO	Δ	k	Li et al.[36]	GBPO	Δ
10	16	15	1	15	30	29	1	9	39	37	2	8	53	51	2
20	19	18	1	27	36	35	1	17	49	47	2	23	75	75	0
33	21	21	0	35	39	38	1	38	62	60	2	42	90	88	2
48	23	23	0	57	44	43	1	43	63	63	0	56	97	95	2
92	26	26	0	66	45	45	0	54	67	67	0	65	100	100	0
101	27	27	0	72	46	46	0	77	74	73	1	82	107	105	2
122	28	28	0	83	47	47	0	96	77	77	0	104	113	111	2
146	29	29	0	139	54	54	0	111	80	80	0	124	118	117	1

Table 5.11: Comparative of results between Li et al. [36] and our GBPO algorithm for instances of strength $t = 2$.

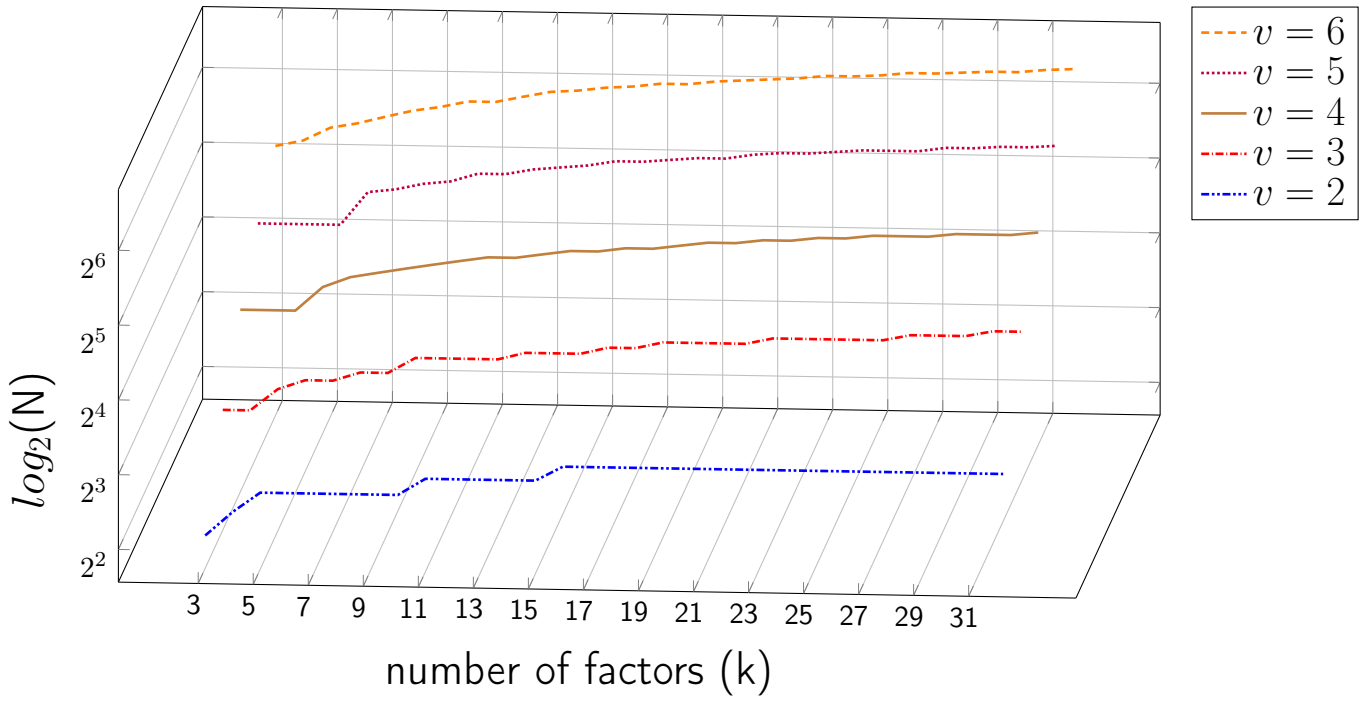


Figure 5.1: Results obtained of the experimentation for the GBPO algorithm with instances of $t = 2$.

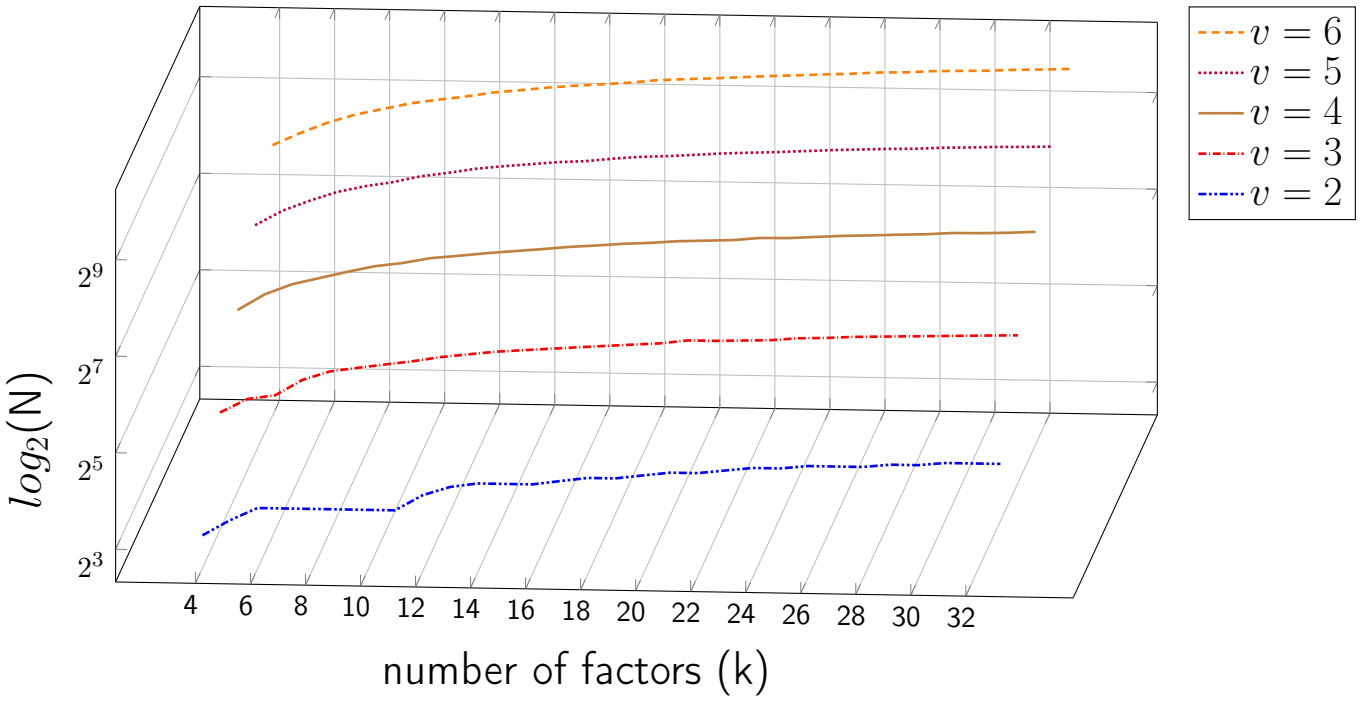


Figure 5.2: Results obtained of the experimentation for the GBPO algorithm with instances of $t = 3$.

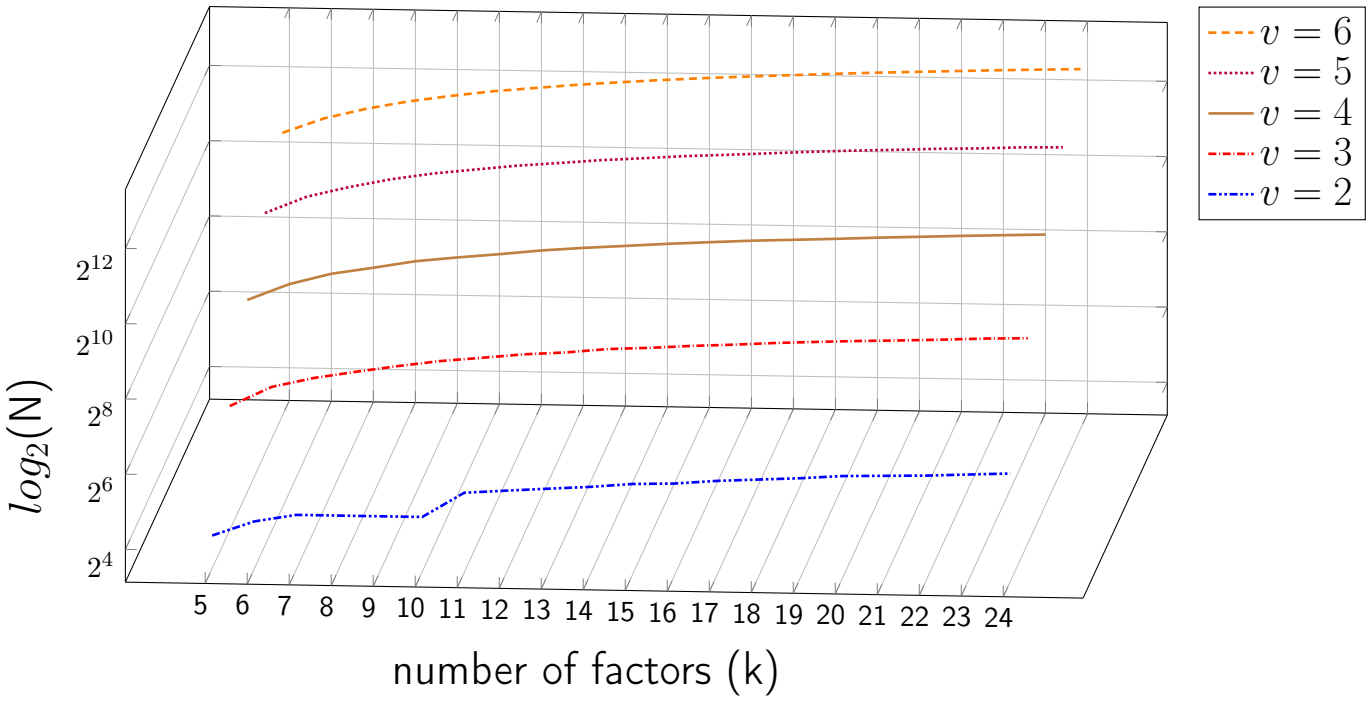


Figure 5.3: Results obtained of the experimentation for the GBPO algorithm with instances of $t = 4$.

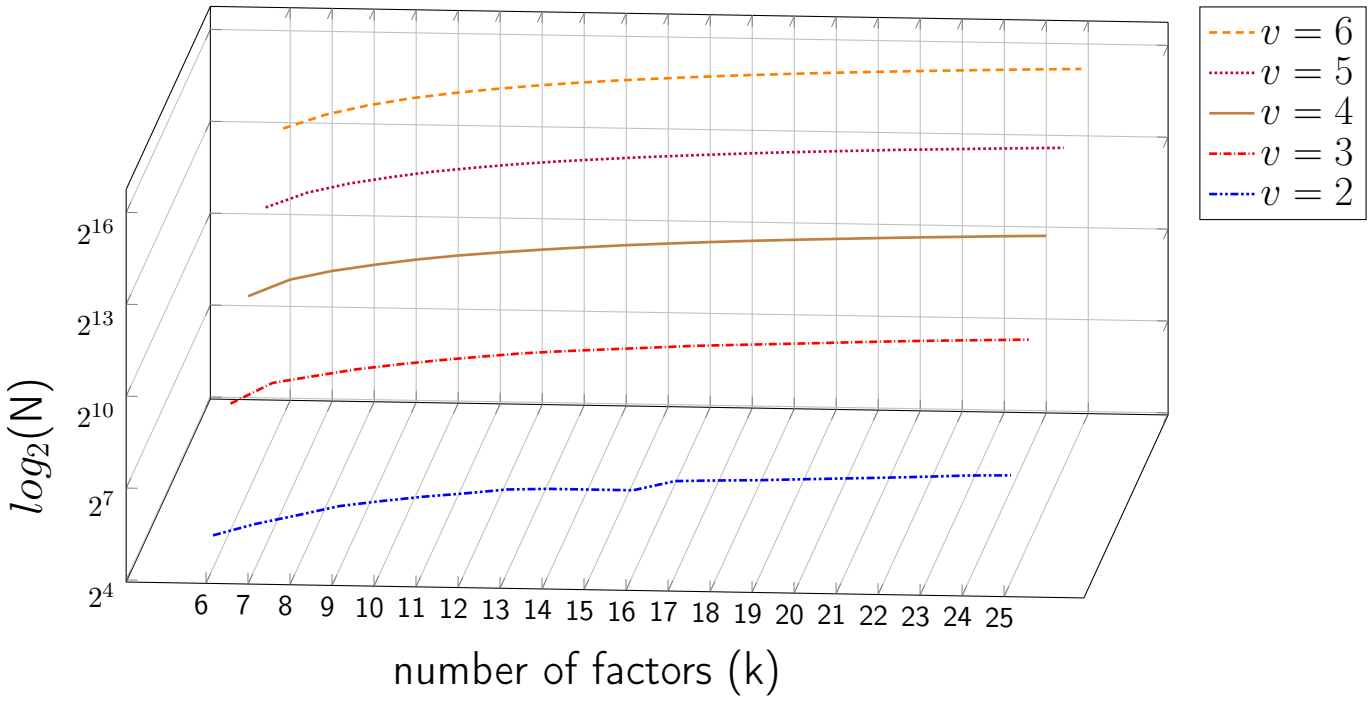


Figure 5.4: Results obtained of the experimentation for the GBPO algorithm with instances of $t = 5$.

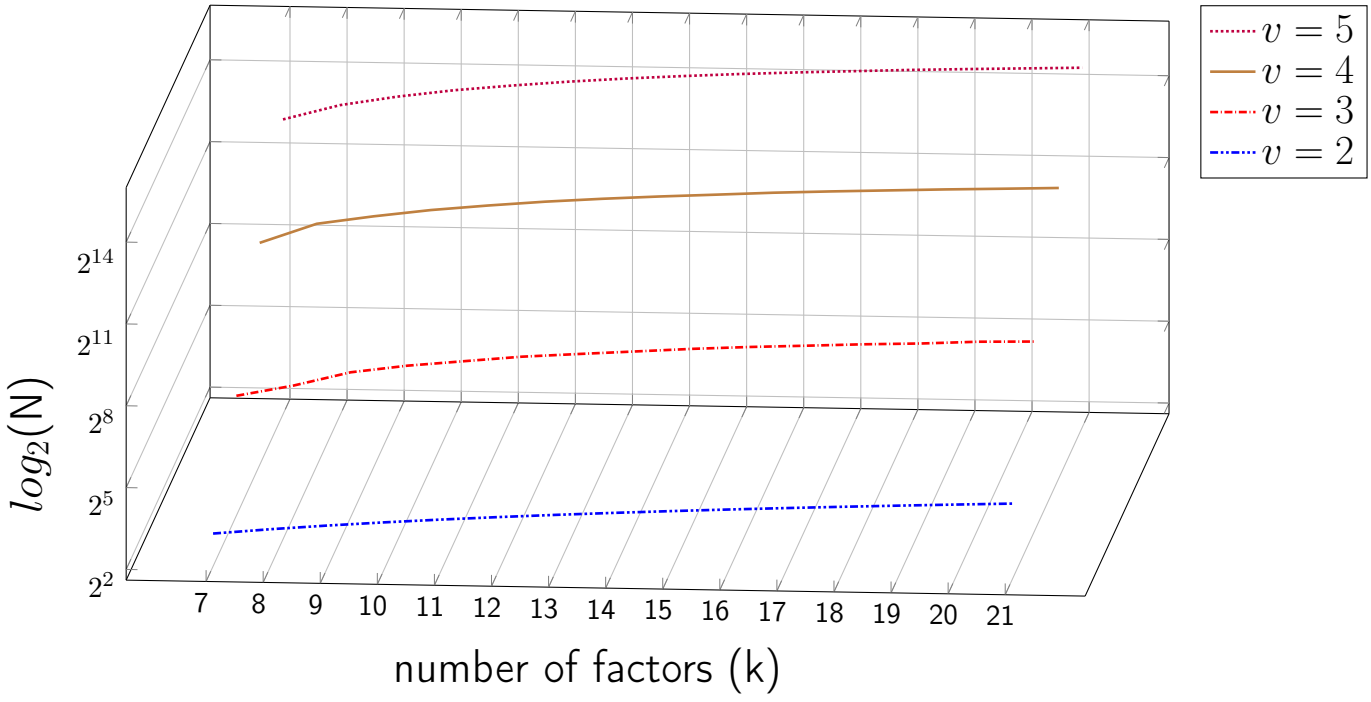


Figure 5.5: Results obtained of the experimentation for the GBPO algorithm with instances of $t = 6$.

5.5 Summary

This chapter presented the results obtained for the experimentation of each one of the proposed algorithms to solve the CACP in the graph domain. The exact algorithm construct 7 CAs that matched the best-known upper bounds and results are optimal. The last version for the greedy algorithm generated 18 instances that matched the results obtained by a greedy algorithm of the state-of-the-art and improved 77 instances versus the same algorithm. The metaheuristic algorithm matched 17 instances of the current best-known upper bounds and improves 4 instances that will be now the best-known upper bounds. The GBPO algorithm improves 560 CAs of the IPOG-F instances repository.

In the next chapter the main contributions of the thesis will be summarized and we will provide directions for further research in the solution of CACP in the graph domain.

6

Conclusions and future works

This chapter finalizes the thesis document. Section 6.1 summarizes the main contributions of the thesis; Section 6.2 gives directions for further research in the CACP in the graph domain.

6.1 Main contributions

A novel and original graph representation for CAs was presented that maps the covering array construction problem to the minimum clique covering problem in the graph domain.

We have presented three types of algorithms to solve CACP in the graph domain, an exact algorithm (ECC), a greedy algorithm (GCC), and a metaheuristic algorithm (SACC) to find the minimum clique cover. And a post-optimization approach to improve previously constructed CAs by mapping them to the graph domain.

The proposed representation allowed us to build CAs one row at a time.

The exact algorithm constructed small optimal results (where $\binom{k}{t}v^t \leq 90$) that matched the best-known upper bounds for 7 CAs.

The first three greedy versions provided competitive results versus the IPOG-F algorithm but takes too much time to process bigger instances. The fourth version of the greedy algorithm improved the size of every instance versus the IPOG-F algorithm.

The metaheuristic algorithm provided competitive results versus the best-known bounds, and improved 4 of the best-known upper bounds by a row.

We have presented GBPO a novel post-optimization approach that takes advantages of moving the missing interactions in a CA to the graph domain. In the graph domain, the coverage of missing interactions is equivalent to finding a set of maximal cliques, and each maximal clique is equivalent to a row in the CA domain. In order to show the validity and advantages of our approach two sets of CAs were used: a) the first set consists of 560 CAs of strength $2 \leq t \leq 6$, alphabet $2 \leq v \leq 6$, and parameters $3 \leq k \leq 32$, generated by IPOG-F, GBPO improved all CAs and 37 cases matched the best-known upper bounds [10]; and b) the second set consists of 32 CAs of strength $t = 2$, alphabet $3 \leq v \leq 6$, and number of parameters $8 \leq k \leq 146$ to compare with a state-of-art, in this set 16 cases were improved and 16 cases were matched [36]. This work were published in [46].

The improvement of the best-known upper bounds for CAs reduces the number of test cases required to validate hardware and software components, which is translated into saving time and money aimed at the hardware and software testing. Moreover, the development of methods that are capable of construct one row at a time, like our greedy approach, allow us to perform testing on-the-fly, which also is translated into saving time by starting the testing progress when the test suite is not complete.

6.2 Future works

Despite the running time reduction in the last version of the greedy algorithm, it is possible to parallelize it to improve its current running time, by unwrapping the search of the candidate clique to be added to the cover. Also, it is possible to incorporate some vocabulary knowledge to the

algorithm, allowing it to fix v cliques at a time instead of one by one.

SACC algorithm is deeply dependent of the neighborhood functions, and the running time of each neighborhood is highly dependent of the instance that is processed, it is possible to add a controller to auto-adapt the probabilities of using each neighborhood on the fly, and to vary the size of the cliques set used in each neighborhood.

GBPO also is suitable to be parallelized to process larger CAs, by decomposing the exhaustive work done in the MAPPER process.

In this thesis is explored a representation to map the characteristics of a CA. It is possible to explore additional graph mappings that generates a distinct problem to solve in the graph domain.

Bibliography

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

[2] Avila-George, H., Torres-Jimenez, J., and Hernández, V. (2012b). Parallel simulated annealing for the covering arrays construction problem. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

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

[4] Becci, G., Dhadyalla, G., Mouzakitis, A., Marco, J., and Moore, A. D. (2013). Robustness testing of real-time automotive systems using sequence covering arrays. *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.*, 6(1):287–293.

[5] 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 Berlin Heidelberg.

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

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

[8] 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.

[9] Chateauneuf, M. A., Colbourn, C. J., and Kreher, D. L. (1999). Covering arrays of strength three. *Designs, Codes and Cryptography*, 16(3):235–242.

[10] Colbourn, C. (2017). Covering arrays tables. `http://www.public.asu.edu/~ccolbou/src/tabby/catable.html`. Accesed on: 24-Apr-2017.

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

[12] 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.

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

[14] D.M. Cohen, S.R. Dalal, A. K. and Patton, G. (1994). The automatic efficient test generator (AETG) system. In *5th International Symposium on Software Reliability Engineering, ISSRE 1994, Monterey, CA, USA, November 6-9, 1994*. Institute of Electrical & Electronics Engineers (IEEE).

[15] Forbes, M., Lawrence, J., Lei, Y., Kacker, R., and Kuhn, D. (2008). Refining the in-parameter-order strategy for constructing covering arrays. *Journal of Research of the National Institute of Standards and Technology*, 113(5):287.

[16] Glover, F. (1989). Tabu search-part I. *ORSA Journal on computing*, 1(3):190–206.

[17] Gonzalez-Hernandez, L., Torres-Jiménez, J., and Rangel-Valdez, N. (2011). An exact approach to maximize the number of wild cards in a covering array. In *MICAI 2011: Advances in Artificial*

Intelligence. 10th Mexican International Conference on Artificial Intelligence, Puebla, Mexico, November 26 - December 4, 2011, Proceedings, Part I, pages 210–221. Springer Berlin Heidelberg.

[18] Hartman, A. (2004). Software and hardware testing using combinatorial covering suites. In *Graph Theory, Combinatorics and Algorithms*, pages 237–266. Springer Science + Business Media.

[19] Holland, J. H. (1973). Genetic algorithms and the optimal allocation of trials. *SIAM J. Comput.*, 2(2):88–105.

[20] Kacker, R. N., Kuhn, D. R., Lei, Y., and Lawrence, J. F. (2013). Combinatorial testing for software: An adaptation of design of experiments. *Measurement*, 46(9):3745 – 3752.

[21] Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA.

[22] Karp, R. M. (1982). Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters*, 1(2):49–51.

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

[24] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN95 - International Conference on Neural Networks*. Institute of Electrical and Electronics Engineers (IEEE).

[25] Kim, Y., Jang, D.-H., and Anderson-Cook, C. M. (2017). Selecting the best wild card entries in a covering array. *Quality and Reliability Engineering International*.

[26] Kirkpatrick, S., Gelatt, C. D., Vecchi, M., et al. (1983). Optimization by simmulated annealing. *science*, 220(4598):671–680.

[27] Kitsos, P., Simon, 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*, pages 69–76.

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

[29] Kokkala, J. I. (2017). Computational methods for classification of codes; laskennallisia mentelmiä koodien luokitteluun.

[30] Kuhn, D., Kacker, R., and Lei, Y. (2010). Practical combinatorial testing. Technical report, National Institute of Standards and Technology.

[31] Kuhn, D. and Okum, V. (2006). Pseudo-exhaustive testing for software. In *30th Annual IEEE / NASA Software Engineering Workshop (SEW-30 2006), 25-28 April 2006, Loyola College Graduate Center, Columbia, MD, USA*. IEEE.

[32] Kuhn, D. and Reilly, M. (2002). An investigation of the applicability of design of experiments to software testing. In *27th Annual NASA Goddard/IEEE Software Engineering Workshop, 5-6 December 2002, Greenbelt, MD, USA.* IEEE Comput. Soc.

[33] 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.

[34] Lei, Y., Kacker, R., Kuhn, D. R., Okun, V., and Lawrence, J. F. (2007). IPOG: A general strategy for t-way software testing. In *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS2007)*. Institute of Electrical & Electronics Engineers (IEEE).

[35] Lei, Y. and Tai, K. (2005). In-parameter-order: a test generation strategy for pairwise testing. In *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (HASE 2005), 12-14 October 2005, Heidelberg, Germany*. Institute of Electrical & Electronics Engineers (IEEE).

[36] Li, X., Dong, Z., Wu, H., Nie, C., and Cai, K.-Y. (2014). Refining a randomized post-optimization method for covering arrays. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*. Institute of Electrical and Electronics Engineers (IEEE).

[37] 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.

[38] Lopez-Escogido, D., Torres-Jimenez, J., Rodriguez-Tello, E., and Rangel-Valdez, N. (2008). *Strength Two Covering Arrays Construction Using a SAT Representation*, pages 44–53. Springer Berlin Heidelberg, Berlin, Heidelberg.

[39] Luce, R. D. and Perry, A. D. (1949). A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116.

[40] Mahmoud, T. and Ahmed, B. S. (2015). An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use. *Expert Systems with Applications*, 42(22):8753–8765.

[41] 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.

[42] Martinez-Pena, J., Torres-Jimenez, J., Rangel-Valdez, N., and Avila-George, H. (2010). A heuristic approach for constructing ternary covering arrays using trinomial coefficients. In *Advances*

in Artificial Intelligence – IBERAMIA 2010 12th Ibero-American Conference on AI, Bahía Blanca, Argentina, November 1-5, 2010. Proceedings, pages 572–581. Springer Science Business Media.

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

[44] Nayeri, P., Colbourn, C. J., and Konjevod, G. (2013). Randomized post-optimization of covering arrays. *European Journal of Combinatorics*, 34(1):91–103.

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

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

[47] Quistorff, J. and Schlage-Puchta, J. (2011). On generalized surjective codes. *Studia Scientiarum Mathematicarum Hungarica*, 48(1):75–92.

[48] Quiz-Ramos, P. (2010). Maximización de renglones constantes para covering arrays. Master's thesis, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional.

[49] R. Kuhn, D. R. Kacker, Y. L. and Hunter, J. (2009). Combinatorial software testing. *Computer*, 42(7):94–96.

[50] Roux, G. (1987). *K-proprietes dans des tableaux de n colonnes : cas particulier de la k-surjectivite et de la k-permutivite*. PhD thesis, University of Paris. Thèse de doctorat dirigée par Cohen, Gerard Mathématiques pures. Combinatoire Paris 6 1987.

[51] Sarkar, K. (2016). *Covering Arrays*: *Algorithms and Asymptotics*. PhD thesis, Arizona State University. Phd dissertation.

[52] 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.

[53] 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.

[54] Skiena, S. S. (1998). *The algorithm design manual: Text*, volume 1. Springer Science & Business Media.

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

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

[57] 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.

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

[59] 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. In *Advances in Artificial Intelligence and Soft Computing. 14th Mexican International Conference on Artificial Intelligence, MICAI 2015, Cuernavaca, Morelos, Mexico, October 25-31, 2015, Proceedings, Part I*, pages 158–177. Springer International Publishing.

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

[61] Torres-Jimenez, J., Rangel-Valdez, N., Gonzalez-Hernandez, 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.

[62] 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.

[63] Torres-Jimenez, J. and Rodriguez-Tello, E. (2010). Simulated annealing for constructing binary covering arrays of variable strength. In *IEEE Congress on Evolutionary Computation*. IEEE.

[64] 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.

[65] Williams, A. W. and Probert, R. L. (2002). Formulation of the interaction test coverage problem as an integer program. In *Testing of Communicating Systems XIV, Applications to Internet Technologies and Services, Proceedings of the IFIP 14th International Conference on Testing Communicating Systems - TestCom 2002, Berlin, Germany, March 19-22, 2002*, TestCom '02, pages 283–, Deventer, The Netherlands, The Netherlands. Kluwer, B.V.

[66] Wu, H., Nie, C., Kuo, F., Leung, H., and Colbourn, C. J. (2015). A discrete particle swarm optimization for covering array generation. *IEEE Transactions on Evolutionary Computation*, 19(4):575–591.

[67] 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).

[68] Younis, M., Zamli, K., and Mat Isa, N. (2008). IRPS – an efficient test data generation strategy for pairwise testing. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 493–500. Springer.