Metaheuristic algorithm for the construction of mixed covering arrays of different strength levels to be used in the design of Black Box software test cases

Algoritmo metaheurístico para la construcción de matrices de cobertura mixtas de distintos niveles de fuerza para su uso en el diseño de casos de prueba de software de Caja Negra

DOI: http://doi.org/10.17981/ingecuc.18.2.2022.18

Artículo de Investigación Científica. Fecha de Recepción: 30/08/2022. Fecha de Aceptación: 13/09/2022.

Andrés Rodrigo López Realpe
Universidad del Cauca - Information Technology R&D Group. Popayán (Colombia)
arlopez@unicauca.edu.co

Jorge Armando Muñoz Ordoñez
Universidad del Cauca - Information Technology R&D Group. Popayán (Colombia)
jorgemo@unicauca.edu.co

Carlos Alberto Cobos Lozada
Universidad del Cauca - Information Technology R&D Group. Popayán (Colombia)
ccobos@unicauca.edu.co

To cite this paper
A. López Realpe, J. Muñoz Ordoñez & C. Cobos Lozada, "Metaheuristic algorithm for the construction of mixed covering arrays of different strength levels to be used in the design of Black Box software test cases", *INGE CUC*, vol. 18, no. 2, pp. 223–237. DOI: http://doi.org/10.17981/ingecuc.18.2.2022.18

Resumen

Introducción— Actualmente, la calidad del software en una característica fundamental para asegurarse un espacio en el mercado global que día a día es más competitivo y exigente. Las pruebas de software permiten a las empresas desarrolladoras de software encontrar y corregir fallos y con ello elevar la calidad de sus productos. El costo de las pruebas hoy se estima en un 50% del costo total del desarrollo de software, por esto se hace necesario encontrar formas menos costosas que garanticen altos niveles de detección de fallos. En este escenario, las pruebas de caja negra tienen un papel fundamental y dentro de estas pruebas los enfoques combinatoriales son una de las mejores opciones. En las pruebas combinatorias se hace necesario que los usuarios probadores (testers) cuenten con una herramienta que les proporcione el menor número de casos de prueba con la mayor cobertura (detección de fallos) de acuerdo con los parámetros del método (procedimiento, función u otro) que desea probar, y este es el objetivo principal de este trabajo.

Objetivo— En este artículo se presenta un algoritmo que soporta la generación de casos de prueba en pruebas de caja negra basado en la creación de Arreglos de Cobertura Mixta (MCA). Estos arreglos permiten generar el menor número de casos de prueba requeridos para probar una unidad de código con la mayor cobertura requerida (mayor posibilidad de detectar fallos). El algoritmo propuesto construye una solución inicial basado en un algoritmo codicioso (greedy) y luego mejora esta solución a través de un proceso iterativo orientado por recocido simulado (algoritmo metaheurístico) y tres formas de definir soluciones vecinas.

Metodología— La investigación se realizó siguiendo el Patrón de Investigación Iterativa propuesto por Pratt. Primero se identificaron los principales problemas reportados en el estado del arte para la construcción de MCA, luego se realizó una revisión de las propuestas de solución a estos problemas. Después se creó un primer algoritmo y luego se fue modificando en forma iterativa este algoritmo, incluyendo y removiendo componentes de acuerdo con resultados experimentales de su funcionamiento. Cuando se obtuvo la versión deseada, se realizó un proceso de afinamiento de parámetros y se comparó con los mejores resultados presentados en la literatura, resultados obtenidos por diferentes algoritmos.

Resultados— El algoritmo propuesto obtiene MCA que son competitivos (en promedio 3 casos de prueba adicionales) frente a los mejores reportados en el estado del arte en un tiempo corto de ejecución, aspecto que es de especial interés para los probadores de software.

Conclusiones— Se confirmó que el enfoque codicioso y metaheurístico basado en recocido simulado es una buena alternativa para la construcción de un MCA. Los algoritmos de construcción de soluciones vecinas son claves para encontrar el MCA requerido y en un menor tiempo de ejecución.

Palabras clave— Arreglos de Cobertura; Arreglos de Cobertura Mixtos; Algoritmos Metaheurísticos; Algoritmos Codiciosos; Recocido Simulado

Abstract

Introduction— Currently, software quality is a fundamental feature to ensure a space in the global market that day by day is more competitive and demanding. Software testing allows software developers to find and fix bugs and, thereby, raise the quality of their products. The cost of testing is now estimated at 50% of the total cost of software development, so it is necessary to find less expensive ways to ensure elevated levels of fault detection. In this scenario, black box tests play a key role; within these tests, combinatorial approaches are one of the best options. In combinatorial tests, test users must have a tool that provides them with the least number of test cases with the greatest coverage (failure detection) according to the parameters of the method (procedure, function, or other) that they want to test, and this is the main objective of this work.

Objective— This paper presents an algorithm that supports the generation of test cases in black box tests based on the creation of Mixed Covering Arrays (MCA). These fixes allow you to generate the smallest number of test cases required to test a unit of code with the highest required coverage. The proposed algorithm builds an initial solution based on a greedy algorithm and then improves this solution through an iterative process oriented by simulated annealing (metaheuristic algorithm).

Methodology— The research was conducted following the Iterative Research Pattern proposed by Pratt. First, the main problems reported in the state of the art for the construction of CSF were identified, then a review of the proposed solutions to these problems was performed. Then a first algorithm was created and then iteratively modified this algorithm, including, and removing components according to the experimental results of its operation. When the desired version was obtained, a process of refinement of parameters was performed and compared with the best results presented in the literature, results obtained by different algorithms.

Results— The proposed algorithm obtains MCAs that are competitive (on average 3 additional test cases) against the best reported in the state of the art in an abbreviated execution time, an aspect that is of special interest for software testers.

Conclusions— It was confirmed that the greedy and metaheuristic approach based on simulated annealing is a suitable alternative for the construction of a CSF. Neighboring solution construction algorithms are key to finding the required MCA in a shorter execution time.

Keywords— Covering Arrays; Mixed Covering Arrays; Metaheuristic algorithms; Greedy algorithms; Simulated Annealing

I. Introduction

The constant growth of the computer industry and the high data processing needs make the world of work increasingly intricate, so every day it becomes more necessary to have software tools to organize and facilitate the handling of information. Today, software systems control and manage huge-economic, health, and transportation systems, among others [1]. Failure to do so could cost substantial amounts of money and even the loss of human life. Therefore, errors in such systems are inadmissible and as a consequence companies currently invest large amounts of money in software testing to detect design and coding errors, among others, to ensure the reliability of the software that is delivered as the final product [2], [3].

The literature reports different techniques to test software, broadly organized into functional (unit, interface, regression, integration, among others) and non-functional (acceptance, performance, installation, reliability, security, among others). Non-functional tests focus on significant aspects of product behavior but are not related to its functions. In contrast, functional tests are defined considering the system requirements and validate and verify that the product complies with the specifications. Within the functional tests, there are a variety of techniques, and the black box tests are one of the most used.

Black box tests can be designed in various ways, highlighting the use of Covering Arrays (CA) and mixed Covering Arrays (MCA), which have successfully demonstrated a decrease in the number of test cases that need to be performed, which implies a lower cost and time in the execution of tests, ensuring maximum coverage, that is, the detection of the largest number of failures given a specific level of interaction of parameters or arguments that are sent to a method (for example a procedure, function or component) [4], [5].

A CA is represented as an matrix and constructed based on the N, k, t, and v values, where N is the number of rows (test cases), k is the number of columns in the matrix (number of parameters in the method), t represents the level of interaction between the different parameters or arguments (for example, if within a method to be tested loops or conditionals involving the same level or nested to 2 parameters, the value of t to be used for the test is 2), and v represents the number of different symbols or values that each parameter will take in the test (for example if you have a parameter that you want to evaluate the values of high, normal, or low, the alphabet of that column is of 3 and are represented by the values 0, 1, and 2).

On a CA each sub-array of size $N \times t$ contains each tuple of size t symbols at least once. The Table 1 shows the CA ($N = 11$, $k = 5$, $t = 2$, $v = 3$), where each column is made up of elements of set $Z_3 = \{0, 1, 2\}$ and strength $t = 2$ means that the sub-arrays made up of the combinations of values of the Z_3 alphabet, namely (0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2), are always present in the 10 combinations of two columns (combining 5 in 2 or combining from N in t), namely: (0, 1), (0, 2), (0, 3), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (2, 4) and (3, 4), based on 0 for the first column. The Table 1 highlights the nonrepeating combinations present for the combination of columns 0 and 1. The two unhighlighted combinations are repeated.

Table 1.
Covering Array (CA).

	0	1	2	3	4
0	2	2	1	0	1
1	2	1	0	0	0
2	2	0	2	1	1
3	2	0	0	2	2
4	1	2	2	0	2
5	1	2	0	1	0
6	1	1	2	2	1
7	1	0	1	2	0
8	0	2	0	2	1
9	0	1	1	1	2
10	0	0	2	0	0

Source: Authors.

An MCA is a CA that has in at least one column a different alphabet than the others, this causes the definition to be set to MCA (N, k, t, {v_1, v_2, ..., vk}), where each column defines the values that make up each column. If all the alphabets are the same, you are talking about a traditional CA, also known as uniform CA or homogeneous CA [6], [7]. The Table 2 shows the MCA ($N = 12$, $k = 5$, $t = 2$, $v = \{4, 3, 3, 2, 2\}$). In this MCA, the starting column consists of the alphabet $Z_4 = \{0, 1, 2, 3\}$, the next two columns consist of the alphabet $Z_3 = \{0, 1, 2\}$, and the final two columns consist of the alphabet $Z_2 = \{0, 1\}$.

Table 2.
Mixed Covering Array (MCA).

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

Source: Authors.

Considering that CA and MCA are the best alternatives to build combinatorial tests, which in turn are the best alternative to building black-box tests, this paper proposes a process of building MCA by adapting state-of-the-art algorithms seeking to optimize response times without losing the quality of the built MCA.

The algorithm consists of two stages, the first that Greedy constructs a matrix that seeks to be the basis of the required MCA and the second where this matrix is optimized based on an algorithm that uses the heuristic of ascent to the hill and the metaheuristic of simulated annealing, the latter performing local optimization based on three ways of defining the neighbors close to the current solution that is optimized, process that is repeated until the maximum number of interactions defined for the algorithm is finished or until the required MCA is found.

The rest of the article is organized as follows: in section II the theoretical bases of CA and MCA are presented and then the related works in the process of its construction are presented, later in section III the proposed algorithm is presented and a synthesis of its functioning, then in section IV the experimental results obtained with the algorithm are presented and finally in section V the conclusions of the work performed and the future work that is expected to develop are presented.

II. Related Works

The central theme of this proposal is black box testing as one of the tools available in the context of software testing, more specifically black box testing, where test cases are generated with a combinatorial approach based on the use of a MCA, and especially, the way these arrangements are built.

Building CAs and MCAs is a complex task. In Google Scholar, IEEE, Scopus, ScienceDirect and SpringerLink different methods are presented for its construction, namely: exact, algebraic, recursive, greedy and metaheuristic [8]. In addition, metaheuristic techniques have produced the best results so far and there are a wide variety of metaheuristic algorithms that have been used in the construction of CA and MCA [6], such as taboo search [9], simulated annealing (Simulated Annealing, SA) [10], genetic algorithms [11], the algorithm by ant colony [12], particle swarm optimization [4], the harmonic search [13], the swarm of birds [14], hybrid algorithms

combining the bee colony algorithm and harmonic search [15], the cuckoo hunt [15], the differential evolution [16], taboo search as a hyper-heuristic proposition [17], and a combination of simulated annealing with a greedy algorithm [18], among others.

Despite all these proposals, previous work shows a tendency to use metaheuristics that integrate or rely on simulated annealing with greedy algorithms and variable neighborhoods taking into account the Table 3 taken from the best results according to Charlie Colbourn (http://www.public.asu.edu/~ccolbou/src/tabby/catable.html), as they are the ones that report the best results. In addition, it was observed that the optimal covering array found by the researchers are not shared but must be purchased or established agreements with the authors of the articles to acquire them. On the other hand, most proposals that build large CAs or MCAs run on supercomputers for prolonged periods of time to obtain the optimal number (lowest theoretical or known) of rows for each built array. The main and most recent proposals are summarized below.

TABLE 3.
BEST RESULTS ALGORITHMS

K	N	T	Algorithm
4	16	3	Orthogonal array
5	19	3	Derive from strenth3
7	21	3	Simulated annealing
9	22	3	Simulated annealing
10	24	3	Simulated annealing
7	25	4	Simulated annealing
9	26	4	Simulated annealing
12	27	4	Simulated annealing
13	28	4	Simulated annealing
15	29	4	Simulated annealing
6	25	5	Orthogonal array
7	29	5	Fix 1 symbols
8	33	5	Fix 1 symbols
9	35	5	Simulated annealing
10	36	5	Search taboo
11	37	5	Simulated annealing
13	38	5	Simulated annealing

Source: Authors.

In 2018 a greedy and metaheuristic 3-stage approach was developed for CA construction, which has other very important approaches such as complement and post optimization [19]. In the first, they use a metaheuristic algorithm and create Covering Perfect Hash Families (CPHF), facilitating the construction of large matrices with little computational effort. The second stage reviews if the CPHFs are CA, if not they are completed (rows are added) to ensure that they are CA and finally in the third stage they are optimized, that is, rows are reduced looking for combinations of columns and values that are redundant in the CA. In general, the proposal obtained a positive result by reducing and improving a total of 21 217 CA and the execution time of the algorithm in general is much less than that reported in the state of the art.

Also that year (2018) the construction of CA is defined using a simulated annealing (SA) based algorithm to create CPHF [20], mentioning that one of the most successful algorithms for this work is proposed by UNL, ASU and UVM [21]. In this work, the target covering array is divided into smaller matrices or "components", after which, each component is constructed by a combinatorial technique or with SA, if previously there is no such construction [21]. According to this article, metaheuristic algorithms are one of the best solutions to build medium and small size CAs most successful, emphasizing simulated annealing as one of the best and most used algorithms for this task, due to the relationship between the execution time and the quality of the solution obtained [21].

For 2019, a new strategy for CA construction was developed based on a greedy algorithm and graph theory [22]. That investigation is based on a representation called Node Coverage (NC) and the algorithm called Graph Based Greedy Algorithm (GBGA) [22]. This work uses the representation through node coverage, to give solution to the construction of CA in the graph domain (transformation of a complete NP problem into another complete NP problem, where, the second problem is solved and the solution is moved to the first problem), this has made the presented algorithm competitive in terms of the generation of CA with the lowest row numbers. It should be noted that this method improved the CA and MAC reported in the state of the art in 80 of 98 cases [22].

In the year 2020 several CAs were improved through CPHF using a simulated annealing algorithm [23]. In this work they managed to improve the upper limits of 19 669 CA, that is, they reduced the number of parameters contained in these CAs [23]. These improvements could be achieved by restricting the inputs in the groups of rows of these arrays, a better one in the local search that consists of replacing the worst column of CPHF with a better solution found by the algorithm, this method shows its improvements in the creation of CA compared to that of 2018 [19].

Colbourn, one of the most important authors on the subject, makes available to the public a repository with a wide list of CA configurations and the optimal values (theoretical or known) of its rows, but does not make available the CAs themselves (http://www.public.asu.edu/~ccolbou/src/tabby/catable.html). In this repository it can be observed that for CA with $k <= 6$ the meta-heuristic algorithms that produce better CA are based on simulated annealing and have greedy approaches.

III. Proposed MCA Generator

In this paper, a generic algorithm is presented that constructs mixed covering arrays with acceptable characteristics reducing the response time, consequently with what the current software development market demands [24]. Specifically, this solution was arrived at in accordance with the results released by the NIST - National Institute of Standards and Technology (Fig. 1), where it could be determined that most of the errors or failures in the software are caused by the interaction (strength, parameter t of a MCA) of two, three, four, five or maximum six components. Considering the above, the simulated annealing algorithm was selected as the best option since it is more frequently found as a generator tool of the lower limits of covering arrays of dimensions not above the MCA configuration ($k = 6$, $t = 6$, $v = 6$).

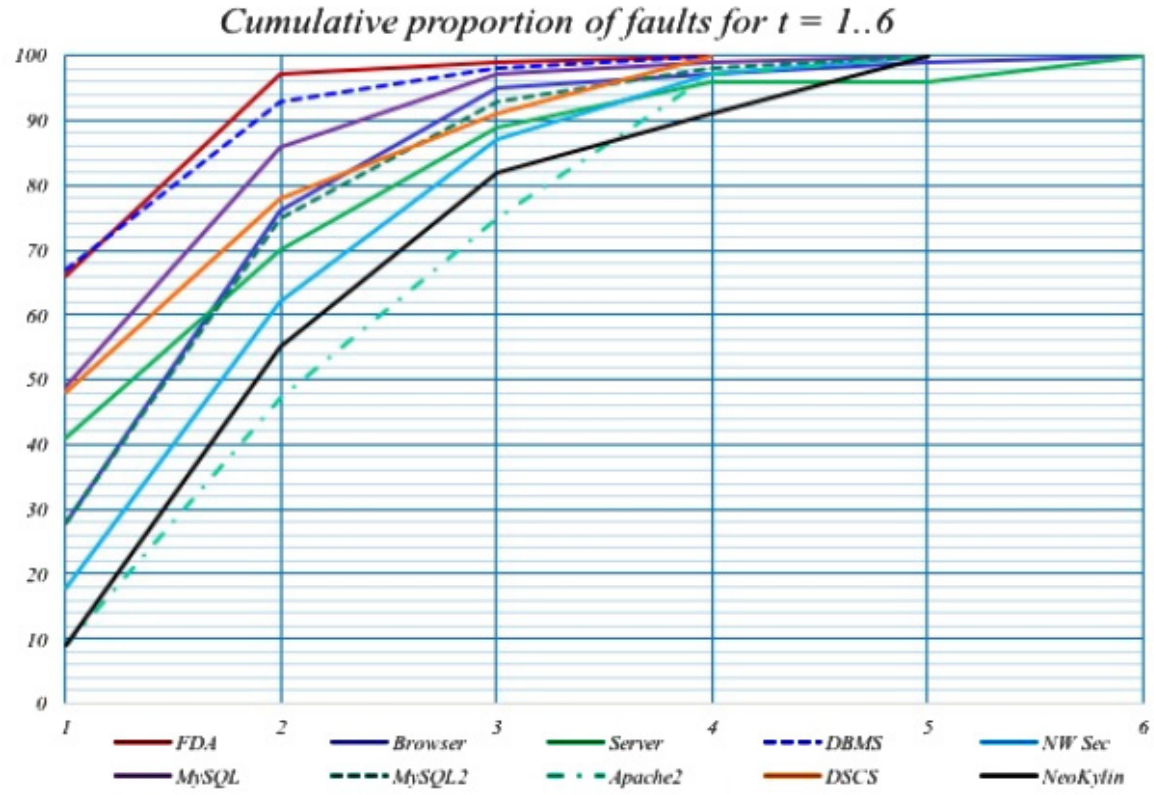

Fig. 1. Coverage of NIST software testing.
Source: Taken from https://csrc.nist.gov/projects/automated-combinatorial-testing-for-software.

Next, the components of the proposed algorithm are described as the basis not only of simulated annealing, but also of the ascent to the hill, a greedy initialization, and the use of three methods to define neighboring solutions to the one being optimized, also the order of execution of the steps established in the Generation algorithm is described.

A. *Primary Algorithm*

The Fig. 2 summarizes this algorithm, which is a descent of the hill because it was designed to minimize. The algorithm receives as input the parameters of the MCA that you want to build, the maximum number of iterations that will be performed within this algorithm and the maximum number of iterations that will be performed in simulated annealing, then (line 2) initializes a solution of that MCA with a greedy algorithm, which gets a matrix that approaches the required MCA and calculates the fitness of that solution.

Inputs: MCA required (N, K, T, v []), maxiter, maxiterSA Outputs: MCA found, Fitness (t-ads missing)	
1.	**Begin**
2.	Generate matrix (m) by greedy method and calculate fitness (f) corresponding to the missing t-ads
3.	**For** i=1 **Until** maxiter **Do**
4.	**If** (f = 0) **Then Break For**
5.	Copy m to m1 and f to f1
6.	Optimize with Simulated Annealing to m1 knowing f1 and using maxiterSA iterations
7.	**If** (f1 < f) **Then**
8.	Replace m with m1 and f with f1
9.	**End If**
10.	**End For**
11.	**Return** m and f
12.	**End**

Fig. 2. Primary Algorithm.
Source: Authors.

Fitness refers to the number of t-ads it needs (combinations of values in combinations of columns) to the matrix to be the required MCA. Then in an iterative process (lines 3 to 10) ask first if the solution to the required MCA (line 4) has already been found if this is the case, otherwise make a copy of the matrix and its fitness and seek to optimize the copy using Simulated Annealing for a maximum number of iterations or cycles (line 6).

If the result of the optimization process is better, in this case it seeks to minimize the value of fitness until it reaches zero (line 7), then the matrix and its fitness is replaced by the best new solution found. At the end (line 11) the matrix is returned and its fitness, if the fitness is zero the matrix corresponds to the required MCA otherwise it is not, i.e. it lacks t-ads to become the required MCA and this can be tried to solve in two ways, if the MCA can be found with the number N of rows requested, the number of interactions should be increased (maxiter and maxiterSA), if not, the algorithm should be run with a greater number of rows, for example, $N + 1$.

On line 2 of the Fig. 2, where Greedy is used, the algorithm summarized in the Fig. 3. This method on line 3 generates a row or row of the required MCA randomly considering the alphabets of each column (parameter) and copies it into the matrix which will eventually return. Then in the loop between lines 4 and 9 it looks to generate a set of random lines (the size of this set depends on a parameter called candidate size or cansize) based on the alphabet of the columns, then looks at which is the most different from the lines already copied in the matrix and the most different is copied in the matrix. After the process is finished, a matrix of $N \times k$ is obtained which is returned to the main algorithm and corresponds to the initial matrix that is expected to be optimized. Unless the required MCA is quite easy to obtain, this initial matrix does not match the required MCA.

The process of fitness calculation involves recording in a matrix, which is called a P-Matrix (an example of this matrix is shown below), where you have at the row level the combinations of columns according to the number of columns and the strength of the MCA, at the column level the combinations of values according to the alphabet of the columns and in the contents of each cell the times that these combinations of columns and combinations of values appear.

If all cells in this matrix have a value ≥ 1, the required MCA is already present, i.e., no t-ada is missing. Otherwise, the fitness value of that matrix corresponds to the number of missing t-ads in the P-Matrix.

Inputs: MCA required (N, K, T, v []), cansize Output: N x K Initial Matrix	
1.	**Begin**
2.	Define a m matrix of N rows by K columns
3.	Randomly generate a row based on the alphabet of each column and copy to m [0]
4.	**For** i=1 **Until** N-1 **Do**
5.	Generate cansize randomized rows based on the alphabet of each column
6.	Calculate the average hamming distance of the previously generated cansize lines against the lines already copied in the matrix m, i.e., from 0 to i-1
7.	Select the best line of the generated cansize, that is, the one with the highest average hamming distance, if there is a tie select the first
8.	Copy the best selected line to m[i]
9.	**End For**
10.	**Return** m
11.	**End**

Fig. 3. Greedy Algorithm for Matrix Initialization.
Source: Authors.

B. *Simulated Annealing based algorithm*

This algorithm is summarized in the Fig. 4. The algorithm receives from the main algorithm the configuration of the required MCA, the m1-matrix and the f1 fitness of that matrix and the maximum number of iterations to be performed in the execution of simulated annealing. In lines 2 to 10 are included the optimization iterations, in each one of them it starts by calculating the temperature that at the beginning is high (one) and decreases linearly in each iteration to almost zero. Then, a copy of the matrix and its fitness is made on line 4. These copies are passed to the local optimization phase that is performed on line 5 by running one of three neighborhood motion algorithms (Algorithm1, Algorithm2 and Algorithm3) which are performed based on p1, p2 and p3 probabilities, respectively.

Inputs: MCA required (N, K, T, v []), cansize Output: N x K Initial Matrix	
1.	**Begin**
2.	Define a m matrix of N rows by K columns
3.	Randomly generate a row based on the alphabet of each column and copy to m [0]
4.	**For** i=1 **Until** N-1 **Do**
5.	Generate cansize randomized rows based on the alphabet of each column
6.	Calculate the average hamming distance of the previously generated cansize lines against the lines already copied in the matrix m, i.e., from 0 to i-1
7.	Select the best line of the generated cansize, that is, the one with the highest average hamming distance, if there is a tie select the first
8.	Copy the best selected line to m[i]
9.	**End For**
10.	**Return** m
11.	**End**

Fig. 4. Simulated Annealing based algorithm.
Source: Authors.

These probabilities must be refined according to an experimental process; to date the best values found are p1 = 0.5, p2 = 0.3 and p3 = 0.2. Already with the m2-matrix modified according to the logic of one of these three algorithms the fitness value is calculated, if this new fitness (f2) is better (lower value) than that of f1 or according to the equation of Boltzmann raised in the simulated annealing is a little less good but acceptable as a strategy to get out of local optimum (line 6), this new solution replaces the current (line 7). It should be clarified that in line 6, the Rnd value corresponds to a pseudo-random number between zero and one. If the required CSF is already found, the iterative cycle (line 9) is broken. At the end, the m1-matrix returns and its fitness f1.

C. *Algorithm1*

This proposed algorithm has three main steps:

1. Calculates the row that contributes least to the matrix, considered the worst row.
2. A matrix named U is constructed that allows you to identify missing combinations and construct a row that is expected to contribute more to the solution than the worst row.
3. Finally, fitness is recalculated with the set change and if this improves the row is replaced.

In step 1 the m2-matrix is taken as input and the worst row is calculated based on the process of creating or updating the P-Matrix. The worst row is set according to a punishment counter per row, which considers the number of extra occurrences contained in that row, i.e., a row that has two or more repeated combinations does not contribute to the goal of the covering array, this row will be replaced by a better one, calculated according to the next step.

For step 2, the U-Matrix is created according to the number of missing combinations determined in the P-Matrix. The following is an example of the process from the m2-matrix shown in the Fig. 5. This figure includes a column called a punishment that is incremented each time the algorithm that computes the P-Matrix encounters a repeating combination. For example, the last row, (0, 0, 0), is 3 times repeated, that is, the combination of values in columns (0, 1), (0, 2), and (1, 2) that are the same as in the first row.

MCAs			Punishment
0	0	0	0
2	1	0	0
1	1	1	0
0	1	1	1
0	0	1	2
0	0	0	3

Fig. 5. m2-matrix with punishment column per row.
Source: Authors.

The P-Matrix shown in the Fig. 6 shows that there are 4 missing t-ads (zeros in green cells) and 5 repeated t-ads (values greater than one in green cells).

		0		1		2		3		4		5	
0 1	0	0 0	3	0 1	1	1 0	0	1 1	1	2 0	0	2 1	1
0 2	1	0 0	2	0 1	2	1 0	0	1 1	1	2 0	1	2 1	0
1 2	2	0 0	2	0 1	1	1 0	1	1 1	2				

Fig. 6. P-Matrix corresponding to the m2-matrix previously shown.
Source: Authors.

Based on the information presented in the Fig. 6, a summary of missing data is constructed and presented in the Fig. 7. It's interpreted like this. The first row in the first column says 0 2, this means that at the crossing of row 0 of P-Matrix that corresponds to the combination of columns 0 1 and column 2 of the P-Matrix that corresponds to the set of missing values 1 0 there is a zero (a missing t-ada). The other rows are read in the same way.

Matrix List		Matrix Values
Matrix of Coordinates P	Combination of Columns	Missing values
0 2	0 1	1 0
0 4	0 1	2 0
1 2	0 2	1 0
1 5	0 2	2 1

Fig. 7. Data pulled from P-Matrix.
Source: Authors.

To construct the U-Matrix, the blue column of the missing data summary (Fig. 7) and you see that column 0 is in all the rows and the first value of the final column in those rows is (1 2 1 2), this becomes the first row of U-Matrix. Then you take column 1 which is in the first two rows of the summary and on the right side you see that the missing values are (0 0) which becomes the second row of U-Matrix and finally you take column 2 and on the right side the missing values are (0 1) and become the third row of U-Matrix (Fig. 8). In this sense, the U-Matrix is constituted by the columns of the m2-matrix that require combinations, and by the value that they require.

According to the U-Matrix it is known that for column 1 the missing values are mostly 0 so its mode is 0, for columns 0 and 2 the algorithm approximates the mode to the largest near value, in this case 2 and 1, respectively.

MCA column	Missing Values				Mode
0	1	2	1	2	2
1	0	0			0
2	0	1			1

Fig. 8. U-Matrix, most missing values to cover for each MCA column.
Source: Authors.

Finally, the third step is executed, which corresponds in changing the worst row for the constructed or optimized row (Fig. 9), and then proceed to recalculate the P-Matrix.

K	0	1	2
N	MCA		
0	0	0	0
1	2	1	0
2	1	1	1
3	0	1	1
4	0	0	1
5	0	0	0

K	0	1	2
N	MCA		
0	0	0	0
1	2	1	0
2	1	1	1
3	0	1	1
4	0	0	1
5	2	0	1

Fig. 9. Row change in m2-matrix per optimized row.
Source: Authors.

When recalculating the P-Matrix (Fig. 10), it can be observed that there are still 2 t-ads missing, but with the change made it was possible to reduce 2 t-ads, since originally 4 were missing.

		0		1		2		3		4		5	
0 1	0	0 0	3	0 1	1	1 0	0	1 1	1	2 0	1	2 1	1
0 2	1	0 0	2	0 1	2	1 0	0	1 1	1	2 0	1	2 1	1
1 2	2	0 0	2	0 1	1	1 0	1	1 1	2				

Fig. 10. P-Matrix calculated with the new m2-matrix.
Source: Authors.

D. *Algorithm2*

This algorithm first defines the number of zeros in the P-Matrix and the characteristics of these zeros (their combinations of columns with their missing values), then selects a combination that will try to solve, that is, at the end of the process stop being zero in the P-Matrix. To achieve this, for each of the rows of m^2 selected in random order the selected combination of columns and missing values is forced to be placed there and its fitness is calculated. Changes to the line are rolled back so that the calculations in the following lines are not affected.

At the end, the line in which the change obtains the best fitness (lower value of missing t-ads) is selected. If the fitness improved in relation to the original is returned that line, otherwise a Shake operation () is performed on that best line, which consists of making a change to a randomly selected column of the line, this change is made to complement the value it has in relation to its maximum alphabet, for example, if the maximum alphabet is 4 and the current value is 3, the resulting value will be 1 (4 - 3).

Below is an example of the algorithm to clarify its operation. Starting from the m2-matrix presented in the Fig. 5, and their respective P-Matrix presented in the Fig. 6, where beforehand you know that your fitness is 4. When you enter algorithm 2, you initially randomly select one of the zeros (0) in the P-Matrix, for example, the number 1 in the list (Fig. 11).

List	Coordinate Matrix P
0	0 2
1	0 4
2	1 2
3	1 5

Fig. 11. Array with the list of zeros in the P-Matrix.
Source: Authors.

Subsequently, a row of the m2-matrix is selected randomly, in this case row 4 (starting with base index 0 for the rows), to which some modifications will be made. According to the coordinates of the P-Matrix of the selected zero, you can know which columns (0 1) and which values (2 0) are needed to cover (Fig. 12). This information changes the selected line from (0 0 1) to (2 0 1). That is, column 0 was assigned the value 2, and column 1 was assigned the value 0.

		0		1		2		3		4		5	
0 1	0	0 0	3	0 1	1	1 0	0	1 1	1	2 0	0	2 1	1
0 2	1	0 0	2	0 1	2	1 0	0	1 1	1	2 0	1	2 1	0
1 2	2	0 0	2	0 1	1	1 0	1	1 1	2				

Fig. 12. Missing values according to P-Matrix.
Source: Authors.

As result, m^2 remains as seen in the Fig. 13 and its P-Matrix as shown in the Fig. 14. It is evident that this change causes 2 missing t-ads to be solved and that the fitness of this solution drops from 4 to 2. This process is repeated with the rest of the lines of the m2-matrix and at the end the change that further decreases the value of fitness is selected. If fitness cannot be reduced, Shake (or) is applied.

MCA		
0	0	0
2	1	0
1	1	1
0	1	1
2	0	1
0	0	0

Fig. 13. m2-matrix with change of line 4.
Source: Authors.

		0		1		2		3		4		5	
0 1	0	0 0	3	0 1	1	1 0	0	1 1	1	2 0	1	2 1	1
0 2	1	0 0	2	0 1	2	1 0	1	1 1	1	2 0	1	2 1	1
1 2	2	0 0	2	0 1	1	1 0	0	1 1	2				

Fig. 14. P-Matrix after applying algorithm2.
Source: Authors.

E. *Algorithm3*

This algorithm initially randomly selects a row (row) of m^2, a copy is made to that row to have it in its original form, then the row for each of its columns is evaluated as change fitness if each of the other options of the alphabet of such column, if there is a change that has a better fitness, this is kept on the solution and advanced to the next column, otherwise, the original value of the column is left in the row. At the end, they can change zero, one or even the columns of the line, but when no change is made a Shake operation () is done on the line.

Starting from the m2-matrix (Fig. 5) and its corresponding P-Matrix (Fig. 6), you can see that the fitness of m2 is 4 so it needs to be improved. If you select the algorithm 3, first you randomly choose a line, in this case line 4 (Fig. 15).

MCA		
0	0	0
2	1	0
1	1	1
0	1	1
0	0	1
0	0	0

Fig. 15. Random selection of a line in the m2-matrix.
Source: Authors.

The algorithm takes that line and initially changes the value of the first column, as that column have alphabet 3, try with the values 1 and 2 and not 0 since that is already assigned in the column. Evaluate fitness with a value of 1 and then evaluate fitness with a value of 2. If there is improvement in fitness the value with which greater improvement was achieved is preserved and continued with the next column which in this case also has a value of 0 and then with the last column which already has a value of 1. In this example, the vector at the end is left with the values (2 0 1). The Fig. 16 shows the m2-matrix resulting from the change made by algorithm 3 and the Fig. 17 displays the P-Matrix result of the change. This algorithm also reduces 2 missing t-ads.

MCA		
0	0	0
2	1	0
1	1	1
0	1	1
2	0	1
0	0	0

Fig. 16. m2-matrix after applying algorithm3.
Source: Authors.

		0		1		2		3		4		5	
0 1	0	0 0	3	0 1	1	1 0	1	1 1	1	2 0	1	2 1	1
0 2	1	0 0	2	0 1	2	1 0	0	1 1	1	2 0	1	2 1	1
1 2	2	0 0	2	0 1	1	1 0	0	1 1	2				

Fig. 17. P-Matrix after applying algorithm3.
Source: Authors.

As can be seen with the examples, the three algorithms allow to improve fitness using different strategies and the results depend on the solution (m2 matrix) being processed, the difficulty of the required MCA and the chance in algorithms 2 and 3.

IV. Analysis and Results

To evaluate the proposed algorithm a set of tests was designed that executes the algorithm and asks it to generate a total of 142 MCA configurations with $k = 6$, $t = 2$, $N = 50$ and alphabets ranging from v (3, 2, 2, 2, 2, 2) to v (6, 6, 3, 3, 2, 2) going through the possible combinations in that interval.

A. *Tuning the Odds*

To establish the appropriate execution percentages for the operation of the algorithms that define a possible better neighbor, the proposed algorithm was executed by modifying the percentages according to the values presented in the Table 4.

As evidenced in the table of results and according to the response time and number of rows, the best results were selected for the operation of the generation algorithm that in this case corresponds to those obtained with the configuration of test 2, where it is appreciated that the deterministic algorithm (algorithm 1), considerably optimizes the response time which translates into a considerable reduction in the consumption of computational resources, but that its value cannot be very high because it slightly increases the number of rows in the MCA.

TABLE 4.
TEST SET WITH DIFFERENT PROBABILITIES.

Test	Algorithm1	Algorithm2	Algorithm3	Rows average	Time Average
1	10%	60%	30%	27.6	34.8
2	50%	30%	20%	27.8	14.2
3	70%	20%	10%	28.2	15.1
4	10%	20%	70%	27.4	23.0
5	0%	70%	30%	27.8	32.0
6	33%	33%	33%	28.1	16.6

Source: Authors.

B. *Comparison with Colbourn results*

In this section, the results obtained in terms of Number of rows (N) per CSF obtained with the proposed algorithm were compared with the results reported in the Charlie Colbourn repository to determine the quality of these. It should be noted that this repository contains the best results reported so far by a wide range of state of the art algorithms, not a single algorithm. The results are shown in the Table 5.

TABLE 5.
COMPARISON OF RESULTS AGAINST COLBOURN.

Columns	Strength	Alphabet	Rows	Rows (Colbourn)	Δ Delta
4	2	3, 3, 3, 3	9	9	0
5	2	3, 3, 3, 3, 3	11	11	0
7	2	3, 3, 3, 3, 3, 3, 3, 3	13	12	1
9	2	3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3	15	13	2
10	2	3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3	15	14	1
5	2	4, 4, 4, 4, 4	16	16	0
6	2	4, 4, 4, 4, 4, 4	19	19	0
7	2	4, 4, 4, 4, 4, 4, 4, 4	23	21	2
9	2	4, 4, 4, 4, 4, 4, 4, 4, 4, 4	25	22	3
6	2	5, 5, 5, 5, 5, 5, 5	33	25	8
7	2	5, 5, 5, 5, 5, 5, 5, 5	35	29	6
8	2	5, 5, 5, 5, 5, 5, 5, 5, 5	38	33	5
9	2	5, 5, 5, 5, 5, 5, 5, 5, 5, 5	39	35	4
10	2	5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5	41	36	5
3	2	6.6.6	36	36	0
4	2	6, 6, 6, 6	38	37	1
5	2	6, 6, 6, 6, 6	43	39	4
6	2	6, 6, 6, 6, 6, 6	48	41	7
8	2	6, 6, 6, 6, 6, 6, 6, 6, 6	54	42	12
9	2	6, 6, 6, 6, 6, 6, 6, 6, 6, 6	56	46	10
10	2	6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6	60	48	12
Percentage			31.7619	27.8095	3.9523

Source: Authors.

As can be seen in the Table 5, there is a difference between the number of rows of the Colbourn repository with which it was possible to generate with the algorithm proposed in this work, which on average increased by 4 lines (http://www.public.asu.edu/~ccolbou/src/tabby/catable.html).

However, the result is not so high considering that the data of the arrangements provided by the Colbourn repository yields the best possible results and each one of them is built with different algorithms, specialized in finding the best possible solution to that particular arrangement, on the contrary, the algorithm proposed in this work serves general purposes, different values of k, t and alphabets.

V. Conclusions

According to the research carried out, it is clear the importance of quality analysis to be performed on software and hardware products, contributing to the constant improvement of the methods that can facilitate this work contributes on different fronts to the construction of better tests and better tools for developers, therefore, in this work focused efforts on improving the performance of the proposed algorithms taking into account the needs of the market, since a rapid response means savings in processing capacity and time, which in the end will directly impact on the quality of software or hardware products.

During the development of this proposal, important aspects were found along the way that could lead to related work, focused on the MCA, which continue to contribute to the constant growth of the improvement of the tests, in this case it is considered the option of building a third stage that from an already built array can extract one of smaller dimensions without clearly affecting the coverage of this, in order to further optimize the response time of the system.

The greedy and metaheuristic approach allowed to find an appropriate balance between the quality of the results and the execution time required for the construction of the MCA (Table 5). The process of tuning parameters is crucial for the use of the algorithm since the results show execution times that are different. Other neighborhood-building mechanisms that allow better MCAs (fewer rows) while keeping response times (execution times) low need to be further explored.

As for the experimental results concerning the generation of MCA are considered to be generally good according to their ranks with respect to the best of the state of the art (Table 5) since the objective of the research was to adapt the solutions found in the state of the art to a generic algorithm that can be used as an input for different purposes oriented to the needs of the current market. Finally, the importance and contribution of related works that were of major influence to guide the development of this research is highlighted.

As a future work is expected to integrate this algorithm in a web solution of generation of test cases based on microservices that is already developed and that also has a post optimization algorithm where the response time is prioritized, and the algorithm presented here seeks to serve to fill an initial repository and generate on demand new MCA required by users of the solution.

Acknowledgements

This work was partially funded by the Universidad del Cauca (Popayán, Cauca, Colombia).

References

[1] A. Kovačević, “The Top 4 Trends That Will Change Software Development in the 2020s”, *IEEE Computer Society*, Mar. 2, 2020. [Online]. Available in: https://www.computer.org/publications/tech-news/trends/the-top-4-trends-that-will-change-software-development-in-the-2020s

[2] A. Tripathi, K. Mishra, S. Tiwari, & N. Kumar, “Improved software cost estimation models: A new perspective based on evolution in Dynamic Environment,” *J Intell Fuzzy Syst*, vol. 35, no. 2, pp. 1707–1720, Jan. 2018. https://doi.org/10.3233/JIFS-169707

[3] H. Wu, C. Nie, F. Kuo, H. Leung, & C. Colbourn, “A Discrete Particle Swarm Optimization for Covering Array Generation,” *IEEE Trans Evol Composite*, vol. 19, No. 4, pp. 575–591, Aug. 2015. https://doi.org/10.1109/TEVC.2014.2362532

[4] T. Mahmoud & B. Ahmed, “An efficient strategy for covering array construction with fuzzy logic-based adaptive swarm optimization for software testing use,” *Expert Syst Appl*, vol. 42, no. 22, pp. 8753–8765, Dec. 2015. https://doi.org/10.1016/j.eswa.2015.07.029

[5] J. Timaná-Peña, C. Cobos-Lozada, & J. Torres-Jimenez, “Metaheuristic algorithms for building Covering Arrays A review,” *Rev Fac Ing*, vol. 25, no. 43, pp. 31–45, Ene. 2016. https://doi.org/10.19053/01211129.v25.n43.2016.5295

[6] J. Timaná, C. Cobos, & J. Torres-Jimenez, "Memetic algorithm for constructing covering arrays of variable strength based on global-best harmony search and simulated annealing," presented at *Mexican International Conference on Artificial Intelligence*, MICAI, Guad., MX, 22-27 Oct. 2018. https://doi.org/10.1007/978-3-030-04491-6_2

[7] S. Sabharwal, P. Bansal, & N. Mittal, "Construction of t-way covering arrays using genetic algorithm," *Int J Syst Assur Eng Manag*, vol. 8, no. 2, pp. 264–274, Mar. 2016. https://doi.org/10.1007/s13198-016-0430-6

[8] X. Yang, *Nature-inspired Metaheuristic Algorithms*, 2 ed., Cambr., UK: Luniver Press, 2010.

[9] R. Walker & C. Colbourn, "Tabu search for covering arrays using permutation vectors," *J Stat Plan Inference*, vol. 139, no. 1, pp. 69–80, Jan., 2009. https://doi.org/10.1016/j.jspi.2008.05.020

[10] S. Esfandyari & V. Rafe, "A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy," *Inf Softw Technol*, vol. 94, pp. 165–185, Feb. 2018. https://doi.org/10.1016/j.infsof.2017.10.007

[11] G. Demiroz & C. Yilmaz, "Using simulated annealing for computing cost-aware covering arrays," *Appl Soft Comput*, vol. 49, pp. 1129–1144, Dec. 2016. https://doi.org/10.1016/j.asoc.2016.08.022

[12] X. Chen, Q. Gu, A. Li, & D. Chen, "Variable strength interaction testing with an ant colony system approach," presented at *Asia-Pacific Software Engineering Conference*, APSEC, PG, MY, 1-3 Dec. 2009. https://doi.org/10.1109/APSEC.2009.18

[13] X. Bao, S. Liu, N. Zhang, & M. Dong, "Combinatorial Test Generation Using Improved Harmony Search Algorithm," *Int J Hybrid Inf Technol*, vol. 8, no. 9, pp. 121–130, Sep. 2015. https://doi.org/10.14257/ijhit.2015.8.9.13

[14] Y. Zhang, L. Cai, & W. Ji, "Combinatorial testing data generation based on bird swarm algorithm," presented at *2nd International Conference on System Reliability and Safety*, ICSRS, MIL, IT, Jan. 2018. https://doi.org/10.1109/ICSRS.2017.8272871

[15] B. Ahmed, T. Abdulsamad & M. Potrus, 'Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the Cuckoo Search algorithm,' *Inf Softw Technol*, vol. 66, pp. 13–29, Oct. 2015. https://doi.org/10.1016/j.infsof.2015.05.005

[16] Y. Wang, M. Zhou, X. Song, M. Gu & J. Sun, "Constructing Cost-Aware Functional Test-Suites Using Nested Differential Evolution Algorithm," *IEEE Trans Evol Comput*, vol. 22, no. 3, pp. 334–346, Jun. 2018. https://doi.org/10.1109/TEVC.2017.2747638

[17] K. Zamli, B. Alkazemi & G. Kendall, "A Tabu Search hyper-heuristic strategy for t-way test suite generation," *Appl Soft Comput*, vol. 44, pp. 57–74, Jul. 2016. https://doi.org/10.1016/j.asoc.2016.03.021

[18] J. Torres-Jimenez, H. Avila-George & Izquierdo-Marquez, "A two-stage algorithm for combinatorial testing," *Optim Lett*, vol. 11, no. 3, pp. 457–469, Feb. 2016. https://doi.org/10.1007/s11590-016-1012-x

[19] I. Izquierdo-Marquez, J. Torres-Jiménez, B. Acevedo-Juárez & H. Avila-George, "A greedy-metaheuristic 3-stage approach to construct covering arrays," *Inf Sci*, vol. 460-461, pp. 172–189, Sep. 2018. https://doi.org/10.1016/j.ins.2018.05.047

[20] J. Torres-Jiménez & I. Izquierdo-Márquez, "A Simulated Annealing Algorithm to Construct Covering Perfect Hash Families," *Math Probl Eng*, pp. 1–15, Jan. 2018. https://doi.org/10.1155/2018/1860673

[21] M. Cohen, C. Colbourn & A. Ling, "Constructing strength three covering arrays with augmented annealing," *Discrete Math*, vol. 308, no. 13, pp. 2709–2722, Jun. 2003. https://doi.org/10.1016/j.disc.2006.06.036

[22] J. Torres-Jimenez & J. Perez-Torres, "A greedy algorithm to construct covering arrays using a graph representation," *Info Sci*, vol. 477, pp. 234–245, Mar. 2019. https://doi.org/10.1016/j.ins.2018.10.048

[23] J. Torres-Jiménez & I. Izquierdo-Márquez, "Improved covering arrays using covering perfect hash families with groups of restricted entries," *Appl Math Composite*, vol. 369, pp. 1–10, Mar. 2020. https://doi.org/10.1016/j.amc.2019.124826

[24] M. Martin, "SOLID: los 5 principios que te ayudarán a desarrollar software de calidad," *profile*, *22 Nov. 2018* [Online]. Available in: https://profile.es/blog/principios-solid-desarrollo-software-calidad/

Andrés Rodrigo López Realpe. Universidad del Cauca - Information Technology R&D Group (Popayán, Colombia). https://orcid.org/0000-0002-2936-4022

Jorge Armando Muñoz Ordoñez. Universidad del Cauca - Information Technology R&D Group (Popayán, Colombia). https://orcid.org/0000-0002-2734-1222

Carlos Alberto Cobos Lozada. Universidad del Cauca - Information Technology R&D Group (Popayán, Colombia). https://orcid.org/0000-0002-6263-1911