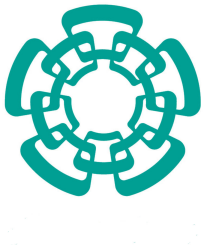


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Tamaulipas

Algoritmo Metaheurístico para Construir Sequence Covering Arrays de Fuerza 3

Tesis que presenta:

Oscar Iván Puga Sánchez

Para obtener el grado de:

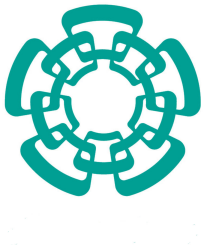
Maestro en Ciencias en Ingeniería y Tecnologías Computacionales

Director de tesis:

Dr. José Torres Jiménez

Cinvestav Tamaulipas. Parque Científico y Tecnológico TecnoTam – Km. 5.5 Carretera Cd. Victoria – Soto La Marina. C.P. 87130 Cd. Victoria, Tamps.

Cd. Victoria, Tamaulipas, México. Diciembre, 2021



Research Center for Advanced Study
from the National Polytechnic Institute

Cinvestav Unidad Tamaulipas

A Metaheuristic Algorithm to Construct Sequence Covering Arrays of Strength 3

Thesis by:

Oscar Iván Puga Sánchez

as the fulfillment of the
requirement for the degree of:

Master of Science in Engineering and Computational Technologies

Thesis Director:

Dr. José Torres Jiménez

Cinvestav Tamaulipas. Parque Científico y Tecnológico TecnoTam – Km. 5.5 Carretera Cd. Victoria – Soto La Marina. C.P. 87130 Cd. Victoria, Tamps.

Cd. Victoria, Tamaulipas, México. December, 2021

La tesis "Algoritmo Metaheurístico para Construir Sequence Covering Arrays de Fuerza 3" presentada por Oscar Iván Puga Sánchez fue aprobada por:

Dr. José Juan García Hernández

Dr. Said Polanco Martagón

Dr. José Torres Jiménez, Director

Cd. Victoria, Tamaulipas, México., 16 de Diciembre de 2021

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Resumen

Algoritmo Metaheurístico para Construir Sequence Covering Arrays de Fuerza 3

por

Oscar Iván Puga Sánchez
Unidad Tamaulipas
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2021
Dr. José Torres Jiménez, Director.

Un *Covering Array (CA)* es un objeto combinatorio ampliamente utilizado como conjunto de pruebas para evaluar el funcionamiento de componentes de software y hardware durante el proceso de elaboración de los mismos. Un CAs realiza un muestreo de las combinaciones de valores de las variables de entrada, a este tipo de pruebas se les denomina pruebas de interacción, en estas pruebas se muestrean todos los posibles valores de cada t variables de entrada de un componente de software o hardware. Si un componente tiene k variables de entrada donde cada variable cuenta con v posibles valores, un CA permite evaluar el funcionamiento de un componente de software y hardware utilizando el menor número de pruebas posible y garantizando la cobertura de cada v^t combinaciones de valores de variables de entrada. No obstante, un CA sólo evalúa las posibles combinaciones de valores de las variables de entrada y no el orden en el que son ingresados los valores de dichas variables al componente de software o hardware. Una solución a este problema es el uso de los *Sequence Covering Arrays (SCA)* como objetos combinatorios en los que cada renglón está definido por una permutación de k variables. Un SCA realiza un muestreo del orden de ocurrencia de cada t variables con el menor número de pruebas. El uso de un CA combinado con un SCA permite realizar una verificación más completa del funcionamiento de componentes de software o hardware, ya que, se evalúan tanto las combinaciones de valores de variables de entrada, así como el orden en el que son ingresados dichos valores al componente de hardware o software. Las dos características fundamentales de los SCAs son mínima cardinalidad y máxima cobertura, es decir, se cubren todas las $t!$ subsecuencias

de cada t variables usando el menor número de renglones. En esta tesis se reporta la construcción de SCAs de fuerza $t = 3$ con hasta $k = 200$ variables, dicha construcción fue realizada usando una metodología de construcción que hace uso de algoritmos de manipulación de SCAs y de un algoritmo metaheurístico basado en el paradigma de recocido simulado. Los resultados más importantes son 19 SCAs, cubriendo desde 4 hasta 200 variables, necesitando para esto desde 6 hasta 24 renglones. Gracias a estos resultados se logró mejorar o igualar todos los resultados previamente reportados y se establecieron un total de 118 nuevas cotas superiores en el tamaño de los SCAs.

Abstract

A Metaheuristic Algorithm to Construct Sequence Covering Arrays of Strength 3

by

Oscar Iván Puga Sánchez

Cinvestav Unidad Tamaulipas

Research Center for Advanced Study from the National Polytechnic Institute, 2021

Dr. José Torres Jiménez, Advisor.

A *Covering Array (CA)* is a combinatorial object widely used as a test set to evaluate the functionality of software and hardware components. A CA samples combination values of input variables, this type of sampling is called interaction testing, in this kind of testing the combination values between each t input variables is sampled at least once. However, a CA only enables the sampling of combinations of values of the input variables and not the order in which the values are input to the component under test. The combinatorial design the enables the sampling of the order in which are input the variables, is called *Sequence Covering Arrays (SCA)*. In an SCA each row is a permutation. The use of a CA combined with an SCA, allows a more complete verification of the functionality of software or hardware components, given that, the combination of values and the order in which the values are presented to a component under test are used. The two fundamental features of an SCA are minimum cardinality and maximum coverage, i.e., all the $t!$ subsequences of every t variables are covered using the minimum number of rows. In this investigation, the construction of SCAs of strength $t = 3$ with as much as 200 variables is reported, in order to do that, a construction methodology supported with construction and manipulation algorithms was used. The construction algorithm is an implementation based on the simulated annealing paradigm. Through the construction of 19 SCAs, it was possible to improve or match all previously reported SCAs, and a grand total of 118 new upper bounds on the size of SCAs was set.

1

Introducción

En este capítulo se presentan las motivaciones, antecedentes, justificación, objetivo general, objetivos particulares y la estructura del resto del documento de tesis. Esta tesis versa sobre la construcción de Sequence Covering Arrays (SCAs). Un SCA es un conjunto de permutaciones que garantiza que cada subsecuencia de un determinado tamaño está cubierta en al menos una permutación.

1.1 Antecedentes y Motivación

La verificación del funcionamiento de componentes de software o hardware es un punto de gran importancia en el proceso de su construcción, ya que es necesario que dichos componentes cumplan con el propósito para el cual fueron creados. La implementación de conjuntos de pruebas usando Covering Arrays (CAs)[53] ha permitido que se ejerciten todas las combinaciones de valores de cada t variables de entrada a componentes de hardware o software garantizando la funcionalidad de dichos componentes. Un inconveniente del uso de los CAs es que sólo permiten probar combinaciones de valores de variables de entrada, pero no permiten probar el orden en el que dichas variables son

alimentadas a los componentes de hardware o software. Para solventar este inconveniente, se propuso el uso de los Sequence Covering Arrays (SCAs)[31] para probar todas las subsecuencias de tamaño t de las variables de entrada a los componentes de software o hardware.

Carlo Ghezzi [23] menciona que, la capacidad de un sistema para realizar el trabajo para el cual fue creado de acuerdo a los requerimientos establecidos de manera correcta, se conoce como *correctitud*. Para un componente con k posibles variables de entrada y v posibles valores para cada una de ellas, un enfoque exhaustivo debe ejecutar v^k pruebas para evaluar su correctitud, pero para componentes con un gran número de variables de entrada, este tipo de enfoque no es el más indicado por el gran costo de recursos (computacionales y de tiempo) que se necesita para realizar dichas pruebas. Una solución al uso de pruebas exhaustivas, es el uso de pruebas que garanticen cubrir todas las interacciones entre cada t variables de entrada usando el menor número de pruebas. El diseño combinatorio que ha sido usado para estas pruebas de interacción es el Covering Array (CA)[53]. Los CAs se han utilizado para realizar pruebas a componentes de software ver[60, 7, 12, 16, 26, 30, 34, 33, 35, 56, 59] y a componentes de hardware ver [26, 49].

El conjunto de pruebas definido por un método exhaustivo puede ser visto como un $k-$hiperplano, donde cada dimensión tiene v posibles valores; un CA realiza un mecanismo de muestreo de todos los $t-$hiperplanos, donde cada dimensión tiene v posibles valores, es decir, se cubren todas las interacciones de tamaño t. Un *Covering Array* se representa como un objeto matemático de tamaño $N \times k$, donde N es el número de renglones (donde cada renglón representa una prueba a realizar), k es el número de columnas (variables de entrada del componente), v es el alfabeto (posibles valores para cada variable) y t es la fuerza de interacción (número de variables que se asume interactúan simultáneamente). Las características fundamentales de un CA son que: presenta mínima cardinalidad y máxima cobertura; es decir, que para cada t-tupla de columnas se cubren todas las v^t combinaciones de variables [53]. En la Tabla 1.1 se presenta un ejemplo de Covering Array con $N = 5$, $k = 4$, $v = 2$ y $t = 2$, donde se puede observar que para cada conjunto de dos columnas se encuentran las combinaciones (0,0), (0,1), (1,0) y (1,1), en particular para las primeras dos columnas se muestra en negritas la primer ocurrencia de cada combinación por cubrir.

Tabla 1.1: Ejemplo de un Covering Array de 5 pruebas, 4 variables, 2 posibles valores e interacción de 2 variables simultáneas, donde se muestra en negritas, la primer aparición de las combinaciones (0,0), (0,1), (1,0) y (1,1) para las primeras dos columnas.

$CA(5;2,4,2)$
0 0 0 0
1 1 1 0
1 1 0 1
1 0 1 1
0 1 1 1

Existen distintos algoritmos para la construcción de CAs, entre los cuales podemos encontrar algoritmos *Deterministas* ver [12, 13, 36, 56, 35, 6] y *No-Deterministas* ver [29, 44].

La funcionalidad de componentes de software y hardware puede ser dependiente del orden en el que son ingresados los valores de las variables de entrada del componente bajo prueba, por esta razón el uso exclusivo de CAs para probar la funcionalidad de los componentes es incompleto, y requiere de un mecanismo que permita probar el orden en el que se ingresan los valores de las variables de entrada. El total de maneras de ingresar los valores de k variables es igual a $k!$, queda claro que probar la funcionalidad de un componente de software o hardware con un enfoque exhaustivo es prohibitivo, por este motivo se propuso el uso de Sequence Covering Arrays (SCA) [31] como diseño de pruebas de la funcionalidad de componentes de software o hardware dependientes del orden de ocurrencia de sus variables. En la literatura se han investigado diversos métodos de construcción de SCAs ver [3, 11, 25, 45, 46, 48].

El uso de un *Sequence Covering Array (SCA)* como diseño de pruebas que verifique la funcionalidad de componentes de software o hardware, permite que cada posible orden de ingresar una subsecuencia de t variables sea ejercitada al menos una vez. Un SCA es un objeto combinatorio de tamaño $M \times k$, donde M es el número de renglones representados por permutaciones de k, k es el total de variables del componente, y t es la fuerza de interacción (variables que interactúan simultáneamente). Las características fundamentales de los SCAs son: mínima cardinalidad y máxima cobertura, es decir que cubre todas las maneras de ordenar una subsecuencia de t variables usando el menor número de permutaciones [10].

En la Tabla 1.2 se presenta un ejemplo de un SCA con $M = 6$, $k = 4$ y $t = 3$ donde se pueden encontrar $\binom{4}{3} \cdot 3! = 24$ subsecuencias de tamaño 3 involucrando cuatro variables. Estas subsecuencias son: $\{(0,1,2),(0,2,1),(1,0,2),(1,2,0),(2,0,1),(2,1,0),(0,1,3),(0,3,1),(1,0,3),(1,3,0),(3,0,1),(3,1,0),(0,2,3),(0,3,2),(2,0,3),(2,3,0),(3,0,2),(3,2,0),(1,2,3),(1,3,2),(2,1,3),(2,3,1),(3,1,2),(3,2,1)\}$.

Tabla 1.2: Ejemplo de un Sequence Covering Array con 6 permutaciones, 4 variables de entrada y fuerza de interacción de 3, que cubre todas las formas de ordenar una subsecuencia de tamaño 3.

$SCA(6;3,4)$
0 1 2 3
0 3 2 1
1 3 0 2
2 1 0 3
2 3 0 1
3 1 2 0

En la literatura se han presentado una gran variedad de algoritmos para la construcción de SCAs, entre los cuales se encuentran algoritmos avaros[32, 31, 38, 20, 19, 1, 18], exactos[41, 37, 51, 10, 18], metaheurísticos[42, 18], y de post-optimización [43], con el objetivo de construir SCAs con el menor número de permutaciones.

Para llevar a cabo una prueba donde se ejerciten tanto combinaciones de valores de variables de entrada y el orden en el que dichas variables son alimentadas a un componente de software o hardware, es necesario combinar los renglones de un CA con los renglones de un SCA. Por ejemplo, sea un $CA(5;2,4,2)$ ver Tabla 1.1, y un $SCA(6;3,4)$ ver 1.2 la combinación de los valores y el orden en el que son alimentados se muestra en la Tabla 1.3, donde a partir del segundo renglón de la primera columna se presentan los valores que debe tener cada variable definido por el CA; y donde a partir de la segunda columna del primer renglon se presenta el orden en el que son ingresados los valores de las variables definidos por el SCA. En negrita se resaltan las pruebas resultantes de la combinación del CA con el SCA.

Tabla 1.3: Conjunto de $N \times M$ pruebas para evaluar el funcionamiento de un componente de $k = 4$ variables, $v = 2$ posibles valores e interacción de $t = 3$ variables simultaneas, usando el SCA de la Tabla 1.2 y el CA de la Tabla 1.1. En negrita se resaltan las pruebas resultantes. El conjunto $\mathbb{C} = \{c_0, c_1, c_2, c_3\}$ indica el conjunto de columnas del CA, y en rojo se resalta el orden de las columnas del CA definido por el SCA.

Orden definido por un SCA / Valores definidos por un CA	$c_0\ c_1\ c_2\ c_3$	$c_0\ c_3\ c_2\ c_1$	$c_1\ c_3\ c_0\ c_2$	$c_2\ c_1\ c_0\ c_3$	$c_2\ c_3\ c_0\ c_1$	$c_3\ c_1\ c_2\ c_0$
$c_0 = 0,\ c_1 = 0,\ c_2 = 0,\ c_3 = 0$	**0 0 0 0**	**0 0 0 0**	**0 0 0 0**	**0 0 0 0**	**0 0 0 0**	**0 0 0 0**
$c_0 = 1,\ c_1 = 1,\ c_2 = 1,\ c_3 = 0$	**1 1 1 0**	**1 0 1 1**	**1 0 1 1**	**1 1 1 0**	**1 0 1 1**	**0 1 1 1**
$c_0 = 1,\ c_1 = 1,\ c_2 = 0,\ c_3 = 1$	**1 1 0 1**	**1 1 0 1**	**1 1 1 0**	**0 1 1 1**	**0 1 1 1**	**1 1 0 1**
$c_0 = 1,\ c_1 = 0,\ c_2 = 1,\ c_3 = 1$	**1 0 1 1**	**1 1 1 0**	**0 1 1 1**	**1 0 1 1**	**1 1 1 0**	**1 0 1 1**
$c_0 = 0,\ c_1 = 1,\ c_2 = 1,\ c_3 = 1$	**0 1 1 1**	**0 1 1 1**	**1 1 0 1**	**1 1 0 1**	**1 1 0 1**	**1 1 1 0**

1.2 Justificación

La construcción de *Sequence Covering Arrays* con el menor número posible de permutaciones es un problema que se sigue investigando hoy en día. A pesar de que existen algoritmos para la construcción de SCAs se continúa la búsqueda de alternativas para construir SCAs con el mínimo número de permutaciones.

En este proyecto de tesis se propone la implementación de una metodología para construir SCAs de fuerza 3 de tamaño competitivo. Se llevará a cabo la construcción de SCAs de fuerza 3 ya que, entre mayor sea la fuerza de evaluación, mayor es la complejidad de construcción de un SCA, por otro lado, el tiempo y recursos computacionales disponibles serán utilizados para mejorar el mayor número de SCAs reportados en la literatura para SCAs de fuerza 3. El proceso de construcción se apoya en dos algoritmos de manipulación de SCAs (que permiten borrar y agregar, renglones o columnas) y de un algoritmo metaheurístico basado en el paradigma de recocido simulado. En [52] se presenta la construcción de CAs por medio del incremento de columnas y/o renglones de un CA óptimo; los algoritmos de manipulación de esta tesis tendrán como propósito la construcción de SCAs de tamaño competitivo mediante la adición de columnas y/o permutaciones a un SCA óptimo. Por otro lado, en [18] se presenta la implementación de un algoritmo de recocido simulado con el cual se mejoraron gran parte de los SCAs de fuerza 3 reportados en la literatura; el algoritmo de

recocido simulado de esta tesis será utilizado para perturbar un SCA con subsecuencias faltantes para ajustar la cobertura de tal manera que se genere un SCA con cobertura completa, utilizando el mismo número de renglones y columnas.

1.2.1 Preguntas de investigación

- ¿La implementación de la metodología de construcción soportada por algoritmos de manipulación de SCAs y de un algoritmo de recocido simulado podrá construir SCAs de fuerza 3 con un máximo de 200 variables y con un tamaño competitivo?

1.2.2 Hipótesis

La orquestación de los algoritmos de manipulación de SCAs y de un algoritmo de recocido simulado usando la metodología propuesta en esta tesis, permitirá mejorar o igualar todas las cotas superiores de SCAs de fuerza 3 hasta con 200 variables.

1.2.3 Objetivos generales y particulares del proyecto

Objetivo general

Construir un repositorio de SCAs de fuerza 3 y hasta con 200 variables que mejore o iguale todas las cotas superiores reportadas a la fecha.

Objetivos particulares

- Definición de una metodología que permita la orquestación de algoritmos de manipulación y de un algoritmo de recocido simulado para construir SCAs.
- Análisis, diseño, implementación y pruebas de un algoritmo que permita agregar y/o borrar renglones de un SCA.
- Análisis, diseño, implementación y pruebas de un algoritmo que agregue y/o quite columnas de un SCA.

- Análisis, diseño, implementación y pruebas de un algoritmo de recocido simulado usando tres métodos de perturbación.
- Construir SCAs de fuerza 3 con hasta 200 variables usando la metodología propuesta en esta tesis que permite orquestar a los algoritmos de manipulación de SCAs y a un algoritmo de recocido simulado.
- Redactar un artículo de revista relacionado con el tema de tesis.

1.3 Contenido del documento

El capítulo 2 presenta el marco teórico relacionado con los SCAs. En el capítulo 3 se cubren métodos de construcción de SCAs reportados en la literatura. El capítulo 4 se enfoca en presentar los algoritmos de manipulación de un SCA (para agregar/borrar renglones/columnas) y de un algoritmo de recocido simulado. El capítulo 5 se concentra en la presentación de la metodología que permite orquestar el funcionamiento de los algoritmos de manipulación de SCAs y de un algoritmo de recocido simulado para la construcción de SCAs. De manera adicional este capítulo sirve para presentar los resultados obtenidos en la construcción de SCAs de fuerza 3 y hasta con 200 variables. Finalmente, las conclusiones y trabajos futuros son presentadas en el capítulo 6.

1.4 Resumen

Este capítulo presentó la relevancia de realizar pruebas funcionales a los componentes de hardware o software a través del uso de CAs y de CAs combinados con SCAs. De la misma manera se tocaron aspectos relacionados con la construcción de CAs y SCAs. Además se introdujeron: las preguntas de investigación, la hipótesis, el objetivo general y los objetivos específicos de esta tesis. El siguiente capítulo hace una revisión del marco teórico relacionado a los SCAs.

2

Marco Teórico

En este capítulo se presentan conceptos relacionados con los diseños combinatorios denominados SCAs. De manera relevante se tratan los temas relacionados con: la construcción de SCAs óptimos; los isomorfismos presentes en los SCAs; representaciones posibles de los SCAs; y el concepto de elementos redundantes de un SCA.

2.1 Sequence Covering Array (SCA)

Richard Kuhn *et al.* [32] han propuesto el uso de Sequence Covering Array (SCA) usando la notación $SCA(M; t, k)$. Un SCA se representa como una matriz de tamaño $M \times k$ (donde cada renglón es una permutación de k variables). Las dos características fundamentales de un SCA son su mínima cardinalidad y su máxima cobertura. La máxima cobertura requiere que se cubra cada manera de ordenar una subsecuencia de tamaño t por lo menos un vez. La mínima cardinalidad implica que para satisfacer la propiedad de máxima cobertura se use el menor número de permutaciones. En la Tabla 2.1 se muestra un ejemplo de un SCA con $M = 6$, $t = 3$ y $k = 4$ el cual contiene todas las $t!$ subsecuencias de t variables, por ejemplo, la primera permutación (0,3,2,1) cubre las subsecuencias

(0,3,2),(0,3,1),(0,2,1) y (3,2,1).

Tabla 2.1: Ejemplo de un SCA con $M = 6$ permutaciones, $k = 4$ variables de entrada y fuerza de evaluación $t = 3$

$SCA(6; 3, 4)$
0 1 2 3
0 3 2 1
1 3 0 2
2 1 0 3
2 3 0 1
3 1 2 0

2.2 Definición de Cuasi-SCA

Un cuasi-SCA es un conjunto de permutaciones al cual le hacen falta cubrir un cierto número de subsecuencias para llegar a ser un SCA.

2.3 Definición de SCAN y SCAK

En esta sección se presentan las definiciones: del Sequence Covering Array Number (SCAN) que es el menor número de permutaciones requeridos para construir un SCA en particular; y del Sequence Covering Array K (SCAK) que es el máximo número de variables para las que se puede construir un SCA con un cierto número de permutaciones.

2.3.1 Sequence Covering Array Number (SCAN)

El Sequence Covering Array Number (SCAN) se denota por $SCAN(t, k) = M$ y es mínimo número de permutaciones requerido para construir un $SCA(M; t, k)$.

La cota inferior y superior en el valor del SCAN(t,k) es descrito en la Ecuación (2.1).

$$t! \leq SCAN(t, k) \leq \binom{k}{t} t! \tag{2.1}$$

Por otro lado, para un SCA de $t \geq 3$, se estiman cotas delimitadas por la Ecuación (2.2) [27, 50]:

$$max(t!, 1 + (\frac{2}{\log_2(e)})(t-1)!(\frac{k}{2k-t+1})\log_2(k-t+2)) \leq SCAN(t,k) \leq \frac{\log(k)t}{\log(\frac{t!}{t!-1})} \quad (2.2)$$

2.3.2 Definición de Sequence Covering Array K

El Sequence Covering Array K (SCAK), es el mayor número de variables con el que se puede construir un SCA con M renglones. La definición formal del SCAK puede ser visto en la Ecuación (2.3.2)

$$SCAK(M;t) = \{\text{máx } k \mid \exists SCA(M;t,k)\}$$

2.4 Cotas en el tamaño de SCAs

Esta sección presenta los dos casos óptimos conocidos en el tamaño de SCAs, es decir casos para los cuales el valor del SCAN es conocido, estos dos casos son: SCAs de fuerza 2 y SCAs de fuerza t con $t+1$ variables. De manera adicional se presentan referencias a las mejores cotas reportadas en el tamaño de SCAs.

2.4.1 Sequence Covering Arrays de fuerza 2

Los SCAs de fuerza $t = 2$ óptimos son construidos de una manera simple: se toma una permutación aleatoria y se agrega dicha permutación leída al revés[31]. En la Tabla 2.2 se muestra un ejemplo de $SCA(2;2,4)$.

Tabla 2.2: Ejemplo de SCA con $M = 2$ permutaciones, $k = 4$ variables de entrada y fuerza de evaluación $t = 2$

$SCA(2;2,4)$
0 1 2 3
3 2 1 0

2.4.2 SCAs de fuerza t con (t+1) variables

Levenshtein conjeturó que $SCAN(t, t+1) = t!$ para todo valor de t [37], sin embargo, Tran Van Trung *et al.* encontraron por búsqueda exhaustiva que es posible construir un $SCAN(t, t+2) = t!$ [40] demostrando que la conjetura de Levenshtein falla para al menos $t = 4$ y $k = 6$.

2.4.3 Mejores cotas superiores en el tamaño de SCAs

Enseguida presentaremos referencias donde se reportan las mejores cotas conocidas en el tamaño de SCAs. En [39] se presenta una recopilación de las mejores cotas reportadas por Oded Margalit, y en [10] se presentan las mejores cotas reportadas por Charles Colbourn *et. al* en el año 2013, por otro lado, en [18] se presentan las mejores cotas reportadas actualmente por Daniel O. Ramírez Acuña y José Torres Jiménez.

2.5 Representación inversa de un Sequence Covering Array.

La notación inversa de un SCA se denota como SCA^{-1} donde para cada permutación α del SCA se construye la permutación α^{-1} donde $\alpha^{-1}_{\alpha_i} = i$ para $0 \leq i < k$ [18]. La evaluación de las subsecuencias faltantes en un SCA por medio de su representación original es más costosa de ejecutar tanto en tiempo de ejecución como en el uso de memoria, por tal motivo, se utiliza la representación inversa SCA^{-1}, donde cada elemento de la permutación α^{-1} indica la posición en la que se encuentra la variable dentro de la permutación α del SCA. En la Tabla 2.3 se muestra un ejemplo de un $SCA(6; 3, 4)$ y su representación inversa $SCA^{-1}(6, 3, 4)$. En esta tesis se utiliza la notación inversa para la evaluación de la cobertura de un SCA para cada subsecuencia de t variables.

Como se observa en la Tabla 2.3, cada columna del SCA en su notación directa indica una posición y el contenido indica la variable que ocurre en dicha posición; por otro lado, cada columna de la notación inversa indica la variable que está ocurriendo y el contenido indica la posición en la que ocurre la variable.

Tabla 2.3: Ejemplo de un $SCA(6;3,4)$ y su representación inversa $SCA^{-1}(6;3,4)$. En rojo se resalta cada variable y en azul se resalta la posición en la que ocurre cada variable.

SCA	SCA^{-1}
0 1 2 3	0 1 2 3
0 1 2 3	0 1 2 3
0 3 2 1	0 3 2 1
1 3 0 2	2 0 3 1
2 1 0 3	2 1 0 3
2 3 0 1	2 3 0 1
3 1 2 0	3 1 2 0

2.6 Normalización de un conjunto de t elementos

La normalización se interpreta como la transformación de una subsecuencia de t variables a una permutación de t variables. Para llevar a cabo la normalización de un conjunto $\mathbb{S}$ primero se realiza la identificación de los elementos $\mathbb{V}$ a normalizar, se utiliza $\mathbb{X}$ como auxiliar para realizar un reetiquetado y finalmente se define el conjunto normalizado $\mathbb{N}$. Sea $\mathbb{S} = (1,3,0)$, primero se define el vector $\mathbb{V}$ como el conjunto ordenado de $\mathbb{S}$, por lo que se obtiene el vector $\mathbb{V} = (0,1,3)$, después se realiza el reetiquetado utilizando $\mathbb{X}$ de tamaño k inicializado como $\forall\ i\ \in [0,...,t\text{-}1]\ \mathbb{X}_{\mathbb{V}_i} = i$; para este ejemplo se ha definido que $\mathbb{X}_0 = 0$, $\mathbb{X}_1 = 1$ y $\mathbb{X}_3 = 2$. Ahora bien, utilizando el vector $\mathbb{X}$ y el vector $\mathbb{S}$ se define el conjunto normalizado como $\forall\ i\ \in [0,...,t\text{-}1]\ \mathbb{N}_i = \mathbb{X}_{\mathbb{S}_i}$, para este caso, $\mathbb{N}_0 = \mathbb{X}_{\mathbb{S}_0} = \mathbb{X}_1 = 1$, $\mathbb{N}_1 = \mathbb{X}_{\mathbb{S}_1} = \mathbb{X}_3 = 2$ y $\mathbb{N}_2 = \mathbb{X}_{\mathbb{S}_2} = \mathbb{X}_0 = 0$. Finalmente calculamos que la normalización de la subsecuencia $\mathbb{S} = (1,3,0)$ es igual a $\mathbb{N} = (1,2,0)$.

El proceso de normalización es de gran ayuda al momento de llevar a cabo el registro de la cobertura de un SCA, ya que se define el conjunto de variables a evaluar y la normalización indica la manera en la que son ingresadas las variables. Para llevar a cabo el proceso de normalización de una subsecuencia $\mathbb{S}$ en la notación inversa se lleva a cabo el siguiente proceso: Se define $\mathbb{V}$ como la t-tupla del SCA^{-1}, se define $\mathbb{S}^{-1}$ como las posiciones definidas por el SCA^{-1} y $\mathbb{V}^{-1}$ como el conjunto ordenado de $\mathbb{S}^{-1}$, se realiza el reetiquetado utilizando $\mathbb{X}$ de tamaño k inicializado como $\forall\ i\ \in [0,...,t\text{-}1]\ \mathbb{X}_{\mathbb{V}_i^{-1}} = i$, la normalización del conjunto de posiciones se define como $\forall\ i\ \in [0,...,t\text{-}1]\ \mathbb{N}^{-1} = \mathbb{X}_{\mathbb{S}_i^{-1}}$, esto nos da como resultado la normalización de las posiciones, para obtener la normalización de la subsecuencia evaluada es necesario obtener la notación inversa de $\mathbb{N}^{-1}$ de la forma $\forall\ i\ \in [0,...,t\text{-}1]\ \mathbb{N}_{\mathbb{N}_i^{-1}} = i$.

Resumen de estructuras de normalización para las dos notaciones:

Normalización en notación directa:

- $\mathbb{S}$: Define la subsecuencia cubierta por una t-tupla del SCA.
- $\mathbb{V}$: Define el conjunto de t variables que conforman la subsecuencia $\mathbb{S}$.
- $\mathbb{N}$: Define la normalización de $\mathbb{S}$, es decir, define el orden el que ocurren las variables de $\mathbb{V}$.

Normalización en notación inversa:

- $\mathbb{V}$: t-tupla del SCA^{-1} que indica el conjunto de t variables que conforman una subsecuencia.
- $\mathbb{S}^{-1}$: Subsecuencia de posiciones que define el orden en el que ocurren las variables de $\mathbb{V}$ en una permutación de k variables.
- $\mathbb{V}^{-1}$: Conjunto ordenado de la subsecuencia de posiciones $\mathbb{S}^{-1}$.
- $\mathbb{N}^{-1}$: Normalización de la subsecuencia de posiciones.
- $\mathbb{N}$: Normalización de la subsecuencia cubierta.

2.7 Notación Factorádica

La notación factorádica de una subsecuencia de t variables, se define como la distancia a la que se encuentra cada elemento de la subsecuencia de su posición ordenada. La notación factorádica se define de izquierda a derecha, evaluando la distancia del elemento a su posición ordenada, intercambiando el elemento que se está ordenando por el actual elemento de la subsecuencia, y procediendo a evaluar el siguiente elemento. En la Tabla 2.4 se presenta un ejemplo de la representación de la subsecuencia (2,0,3,1) en notación factorádica, donde se define cada elemento en cada uno de los renglones de la tabla. En negrita se resalta el elemento que se está definiendo.

Tabla 2.4: Ejemplo del proceso de cálculo de la notación factorádica de la subsecuencia (2,0,3,1).

Subsecuencia	Notación factorádica
2,**0**,3,1	1,-,-
0,2,3,**1**	1,2,-
0,1,3,**2**	1,2,1
0,1,2,3	1,2,1

2.8 Evaluación de la cobertura de un SCA

Para que un SCA presente máxima cobertura, es necesario que cada subsecuencia de t variables ocurra en todos sus posibles órdenes por lo menos una vez, es decir, que para un SCA con los parámetros $t = 3$ y $k = 4$ se deben cubrir las subsecuencias:{ (0,1,2), (0,2,1), (1,0,2), (1,2,0), (2,0,1), (2,1,0), (0,1,3), (0,3,1), (1,0,3), (1,3,0), (3,0,1), (3,1,0), (0,2,3), (0,3,2), (2,0,3), (2,3,0), (3,0,2), (3,2,0), (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1) }.

2.8.1 Evaluación basada en la notación directa

Para cada conjunto de t columnas del SCA, se evalúan las subsecuencias cubiertas por cada permutación con el conjunto de columnas definido. El conjunto $\mathbb{V}$ de variables involucradas se calcula en cada permutación, y el orden cubierto $\mathbb{N}$ se evalúa normalizando la subsecuencia de cada permutación. En la Tabla 2.5 se presenta un ejemplo de cobertura de las primeras tres columnas del SCA de la Tabla 2.3 en notación directa.

Tabla 2.5: Cobertura de las primeras tres columnas del SCA en notación directa de la Tabla 2.3. La primera columna indica la subsecuencia cubierta, la segunda columna indica el conjunto de t variables, y la tercera columna indica la normalización de la subsecuencia.

Subsecuencia $\mathbb{S}$	conjunto de variables $\mathbb{V}$	Orden cubierto $\mathbb{N}$
0 1 2	0 1 2	0 1 2
0 3 2	0 2 3	0 2 1
1 3 0	0 1 3	1 2 0
2 1 0	0 1 2	2 1 0
2 3 0	0 2 3	1 2 0
3 1 2	1 2 3	2 0 1

2.8.2 Evaluación basada en la notación inversa

En el SCA inverso se realiza la evaluación de manera que cada conjunto de t variables $\mathbb{V}$ se define como cada t-tupla de la matriz inversa, y el conjunto de elementos $\mathbb{S}^{-1}$ conformado por cada permutación define una subsecuencia de posiciones, al normalizar $\mathbb{S}^{-1}$ se obtiene el ordenamiento de posiciones cubierto $\mathbb{N}^{-1}$, y al calcular la notación inversa de $\mathbb{N}^{-1}$ se obtiene el ordenamiento $\mathbb{N}$ que define el ordenamiento cubierto para el conjunto de variables $\mathbb{V}$, es decir, la subsecuencia cubierta por el SCA. En la Tabla 2.6 se presenta la cobertura de subsecuencias en base al SCA^{-1} de la Tabla 2.3 para las primeras tres columnas.

Tabla 2.6: Cobertura de las primeras tres columnas del SCA en notación inversa de la Tabla 2.3. La primera columna indica la subsecuencia cubierta, la segunda columna indica el conjunto de t variables, y la tercera columna indica la normalización de la subsecuencia.

Subsecuencia $\mathbb{S}^{-1}$	conjunto de variables $\mathbb{V}$	Orden de posiciones cubierto $\mathbb{N}^{-1}$	orden de $\mathbb{V}$ cubierto
0 1 2	0 1 2	0 1 2	0 1 2
0 3 2	0 1 2	0 2 1	0 2 1
2 0 3	0 1 2	1 0 2	1 0 2
2 1 0	0 1 2	2 1 0	2 1 0
2 3 0	0 1 2	1 2 0	2 0 1
3 1 2	0 1 2	2 0 1	1 2 0

2.9 Verificación de la cobertura de un SCA

Para validar la cobertura de un SCA se usa una matriz $\mathbf{P}$ denominada *matriz de cobertura* de tamaño $\binom{k}{t} \times t!$ que permite llevar el conteo de las subsecuencias cubiertas para cada subconjunto de t columnas. Dada una subsecuencia tomada de un SCA, para efectos de evaluar su cobertura es necesario determinar el renglón y la columna de la matriz $\mathbf{P}$ para controlar la cobertura. El renglón de la matriz $\mathbf{P}$ que corresponde a una subsecuencia de tamaño t se obtiene transformando las columnas ordenadas usando el **Algoritmo 1**. La columna de la matriz $\mathbf{P}$ a partir de una subsecuencia de tamaño t se calcula transformando la subsecuencia a la notación factorádica con el **Algoritmo 3**, y después transformando la notación factorádica a un número con el **Algoritmo 4** usando la regla de Ruffini.

Ejemplo. La subsecuencia (2,3,0,1) involucra a las columnas 0,1,2 y 3 por lo que, al utilizar el **Algoritmo 1** con $t = 4$ se obtiene el valor 0. La notación factorádica (2,2,0,0) (generada por el **Algoritmo 3** con la subsecuencia (2,3,0,1)) se utiliza en el **Algoritmo 4** y se obtiene el valor 8. Entonces, a la subsecuencia (2,3,0,1) le corresponde el contador de matriz de cobertura $\mathbf{P}_{0,8}$.

Enseguida se presenta el **Algoritmo 1** el cual recibe un conjunto de columnas de tamaño t y devuelve la sumatoria de los coeficientes binomiales de cada uno de los elementos del conjunto de columnas dado.

Algoritmo 1 DirectGTP($\mathbb{V}$,t)

Entrada: $\mathbb{V}$: Conjunto de Columnas.
Entrada: t: Longitud del conjunto.
Salida: α: Valor entero positivo.

```
1: a := 0;
2: ∀ i ∈ [0, ..., t-1]{
3:     a := a + Binomial(V_i, i + 1);
4: }
5: devolver a;
```

El **Algoritmo 2** calcula el número de veces que se pueden seleccionar t elementos de un total de n elementos (valor equivalente a $\binom{n}{t}$).

Algoritmo 2 Binomial(n,t)

Entrada: n: Total de variables.
Entrada: t: Variables a tomar del total.
Salida: b : Coeficiente binomial de n en t.

```
1: b := 1
2: si (n < t) devolver 0;
3: si (n = t) devolver 1;
4: ∀ i ∈ [1, ..., t]{
5:     b := (b*(n-i+1))/i;
6: }
7: devolver b;
```

Usando el **Algoritmo 3** se convierte una subsecuencia $\mathbb{S}$ a su notación factorádica $\mathbb{F}$. Se define $\mathbb{S}^{-1}$ como la notación inversa de la subsecuencia $\mathbb{S}$ para realizar el ordenamiento de la subsecuencia dada, y de esa manera, definir la notación factorádica. Si un elemento de la subsecuencia ya se

encuentra ordenado, entonces se define 0 en la notación factorial para la posición evaluada, de lo contrario, se define el elemento de la notación factorádica como la resta de $\mathbb{S}_i^{-1}$-$\mathbb{S}_{\mathbb{S}_i}^{-1}$ donde i define el elemento de la notación factorádica que se está definiendo.

Algoritmo 3 notacion_factoradica($\mathbb{S}, t$)

Entrada: $\mathbb{S}$: Secuencia cubierta.
Entrada: t : Fuerza de evaluación.
Salida: $\mathbb{F}$: Notación factorial de $\mathbb{S}$.

```
1: ———————— Variables y Estructuras de ejecución ————————————————
2: S^-1 ← Notación inversa de S.
3: ———————————————— Procedimiento ————————————————————————————
4: ∀ i ∈ [0, ..., t-1] S^-1_{S_i} = i;
5: ∀ i ∈ [0, ..., t-2]{
6:     si (S_i = i) F_i := 0;
7:     si (S_i ≠ i){
8:       F_i := S_i^-1 - S_{S_i}^-1;
9:       temp := S_i;
10:      S_i := S_{i+F_i};
11:      S_{i+F_i} := temp;
12:      temp := S_i^-1;
13:      S_i^-1 := S^-1_{Si+F_i};
14:      S^-1_{Si+F_i} := temp;
15:    }
16: }
17: devolver F;
```

En el **Algoritmo 4** se convierte una notación factorádica en un valor entero utilizando una variación de la regla de Ruffini para valores con base variable.

Algoritmo 4 Ruffini($\mathbb{F}$,r)

Entrada: $\mathbb{F}$: Vector que contiene una notación factorial de longitud t.
Salida: a: Valor que representa la notación factorial como un entero positivo.

1: $a := \mathbb{F}_0$;
2: $\forall\ i\ \in\ [1, ..., r\text{-}1]\{$
3: $\quad a := a * (t\text{-}i) + \mathbb{F}_i$;
4: $\}$
5: **devolver** a;

2.10 Función de evaluación de los SCAs

La función de evaluación está basada en el número de subsecuencias faltantes en la matriz de cobertura, i.e., $|\mathbf{P}_{i,j} = 0|$, un SCA satisface $\forall(i,j)P_{i,j} > 0$.

2.11 Isomorfismos de un SCA

En esta sección se presenta los isomorfismos inherentes a un SCA a través de la definición de las simetrías válidas.

2.11.1 Simetría de renglones

Dado un SCA $\mathbf{A}$ con M renglones, se pueden generar $M!$ matrices isomorfas (denotadas por $\mathbf{A}'$) permutando sus renglones ver un ejemplo en la Tabla 2.7.

Tabla 2.7: Ejemplo de isomorfismo por simetría de renglones para un SCA de $M = 6$ permutaciones, fuerza $t = 3$ y $k = 4$ variables.

SCA $\mathbf{A}$	SCA $\mathbf{A}'$
0 1 2 3	3 1 2 0
0 3 2 1	0 1 2 3
1 3 0 2	2 3 0 1
2 1 0 3	0 3 2 1
2 3 0 1	2 1 0 3
3 1 2 0	1 3 0 2

2.11.2 Simetría de inversión de columnas

Dada una matriz SCA $\mathbf{A}$ se puede generar una matriz isomorfa $\mathbf{A}'$ con las columnas columnas de $\mathbf{A}$ leídas en orden inverso. En la Tabla 2.8 se presenta un ejemplo de la simetría de inversión de columnas con un SCA con $M = 6$ permutaciones, fuerza $t = 3$ y $k = 4$ variables.

Tabla 2.8: Ejemplo de isomorfismo por simetría de inversión de columnas para un SCA de $M = 6$ permutaciones, fuerza $t = 3$ y $k = 4$ variables.

SCA $\mathbf{A}$	SCA $\mathbf{A}'$
0 1 2 3	3 2 1 0
0 3 2 1	1 2 3 0
1 3 0 2	2 0 3 1
2 1 0 3	3 0 1 2
2 3 0 1	1 0 3 2
3 1 2 0	0 2 1 3

2.11.3 Simetría de variables

Dada una matriz SCA $\mathbf{A}$ con k variables se pueden generar $k!$ SCAs isomorfos denotados genéricamente por $\mathbf{A}'$. Esta simetría requiere permutar las variables en todo el SCA. En la Tabla 2.9 se presenta un ejemplo de isomorfismo de un SCA de $M = 6$ permutaciones, fuerza $t = 3$ y $k = 4$ variables por medio de simetría de variables aplicando la permutación de variables $\{3, 0, 2, 1\}$.

Tabla 2.9: Ejemplo de isomorfismo por simetría de variables para un SCA de $M = 6$ permutaciones, fuerza $t = 3$ y $k = 4$ variables, usando la permutación $\{3, 0, 2, 1\}$.

SCA $\mathbf{A}$	SCA $\mathbf{A}'$
0 1 2 3	3 0 2 1
0 3 2 1	3 1 2 0
1 3 0 2	0 1 3 2
2 1 0 3	2 0 3 1
2 3 0 1	2 1 3 0
3 1 2 0	1 0 2 3

2.12 Representación en el dominio de permutaciones a/desde el dominio de Grafos Acíclicos Dirigidos

En esta sección trataremos la transformación del dominio de permutaciones a un Grafo Acíclico Dirigido (GAD) y viceversa.

2.12.1 Transformación de una permutación a un GAD

Para transformar una permutación que pertenece a un SCA de fuerza t con k variables a un Grafo Acíclico Dirigido (GAD), es necesario transformar cada $\binom{k}{t}$ subsecuencias de tamaño t en $t-1$ arcos involucrando t nodos. Una subsecuencia $\mathbb{S}$ de t variables se representa como un subgrafo $G = (\mathbb{V}, \mathbb{E})$ donde: $|\mathbb{V}| = t$, $|\mathbb{E}| = (t-1)$, $\mathbb{V} = \{\mathbb{S}_0, \mathbb{S}_1, ..., \mathbb{S}_{t-1}\}$ y $\mathbb{E} = \{(\mathbb{S}_0, \mathbb{S}_1), (\mathbb{S}_1, \mathbb{S}_2), ..., (\mathbb{S}_{t-2}, \mathbb{S}_{t-1})\}$ [18].

Dado un GAD G es posible agregarle una subsecuencia sii, el subgrafo correspondiente a la subsecuencia al ser integrado en el GAD G sigue conservando la propiedad de ser un grafo acíclico [18].

Dadas las subsecuencias (0,4,2), (0,5,1), (4,2,1), (4,2,5), (2,5,1), (3,2,1), (3,2,5) generan el GAD mostrado en la Figura 2.1.

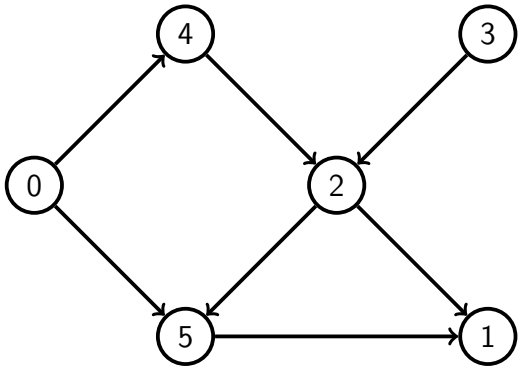


Figura 2.1: Grafo acíclico dirigido correspondiente a las subsecuencias (0,4,2), (0,5,1), (4,2,1), (4,2,5), (2,5,1), (3,2,1), (3,2,5).

2.12.2 Transformación de un GAD a una permutación

Para realizar la transformación del dominio de GAD al dominio de permutaciones, es necesario usar un algoritmo de ordenamiento topológico. Un ordenamiento topológico de un GAD $G = (\mathbb{E}, \mathbb{V})$ es un ordenamiento lineal de los vértices de un grafo G donde para cada arista (u, w), u aparece antes de w en el ordenamiento [17]. Para construir una permutación en base a un GAD se realiza un ordenamiento topológico donde se van eliminando nodos del grafo de tal manera que se elimine el nodo al cual no apuntan otros nodos, el nodo eliminado se agrega como variable en la permutación que se está generando, en la Figura 2.2 se muestra un ejemplo de ordenamiento topológico, con el cual, se obtiene la permutación (2,1,4,3,5,0) o la permutación (2,4,1,3,5,0).

El proceso de conversión de un GAD al dominio de las permutaciones puede generar diferentes resultados debido a que el ordenamiento topológico no es único [18].

2.13 Elementos redundantes en un SCA

Cada subsecuencia en un SCA debe aparecer al menos en una permutación. Cuando las subsecuencias aparecen más de una vez, es posible que el SCA tenga elementos redundantes. Un elemento en el renglón i y la columna j es redundante sii pueden ser ignoradas las subsecuencias en las que participa sin afectar las propiedades de cobertura del SCA. Un elemento redundante se

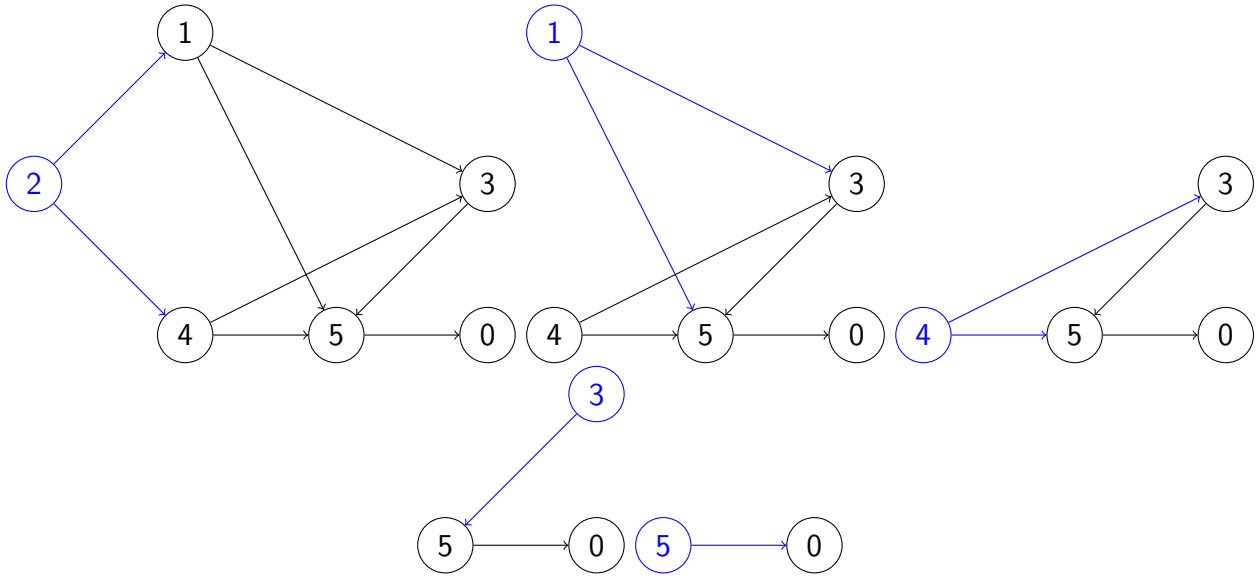


Figura 2.2: Eliminación de los nodos del GAD para la construcción de permutaciones, donde se resalta el nodo eliminado en color azúl, las permutaciones resultantes pueden ser (2,1,4,3,5,0) o la permutación (2,4,1,3,5,0).

representa con el símbolo "-" en la matriz SCA[18].

Un uso de los elementos redundantes es la eliminación de permutaciones redundantes, dado que algunas subsecuencias cubiertas por algún renglón pueden ser cubiertas usando los elementos redundantes de otro(s) renglón(es) [18].

En la Tabla 2.10 en la primer columna se presenta un ejemplo de un SCA con $M = 10$ permutaciones, fuerza $t = 3$ y $k = 8$ variables, en la segunda, tercera y cuarta columna se presentan 3 posibles SCAs con elementos redundantes.

Como se puede observar en la segunda, tercera y cuarta columna de la Tabla 2.10, la detección de elementos redundantes puede tener múltiples soluciones. Dado que un renglón de un SCA con k variables tiene r elementos redundantes, esto puede generar $\binom{k}{r} \cdot r!$, esto es porque los elementos redundantes pueden acomodarse en $\binom{k}{r}$ maneras y los valores de esos elementos redundantes pueden organizarse en $r!$ formas. Sea la permutación $--234$ con $k = 5$ y $r = 2$, la Tabla 2.11 presenta todas las posibles permutaciones equivalentes. Respecto a la Tabla 2.11, la primer columna presenta las posibles $\binom{5}{2} = 10$ posiciones en las que se acomodan los elementos redundantes, la segunda

Tabla 2.10: Ejemplo de SCA de $M = 10$ permutaciones, fuerza $t = 3$ y $k = 8$ variables y tres opciones de variables redundantes para la matriz dada.

$SCA(10; 3, 8)$	Opción 1	Opción 2	Opción 3
1 4 0 7 6 3 5 2	1 4 0 7 6 3 5 2	1 4 0 7 6 3 5 2	1 4 0 7 6 3 5 2
6 7 2 0 5 3 1 4	6 7 2 0 5 3 1 4	6 7 2 0 5 3 1 4	6 7 2 0 5 3 1 4
5 2 7 4 1 3 6 0	5 2 7 4 1 3 6 0	5 2 7 4 1 3 6 0	5 2 7 4 1 3 6 0
3 0 4 2 1 5 6 7	3 0 4 2 1 5 6 7	3 0 4 2 1 5 6 7	3 0 4 2 1 5 6 7
3 7 6 4 1 2 5 0	3 7 6 4 1 2 5 0	3 7 6 4 1 2 5 0	3 7 6 4 1 2 5 0
5 6 0 1 2 4 3 7	5 6 0 1 2 4 3 7	5 6 0 1 2 4 3 7	5 6 0 1 2 4 3 7
1 7 5 0 3 2 6 4	1 7 5 0 3 2 6 4	1 7 5 0 3 2 6 4	1 7 5 0 3 2 6 4
2 4 0 6 7 3 5 1	2 4 0 6 7 3 5 1	2 4 0 6 7 3 5 1	2 4 0 6 7 3 5 1
6 3 5 0 7 1 4 2	6 3 5 0 7 - 4 2	6 3 5 0 7 4 2 -	- 6 3 5 0 7 4 2
4 2 5 6 7 0 3 1	4 2 5 - 7 0 3 -	- 4 2 - 5 7 0 3	4 2 5 7 0 3 - -

columna asigna los valores 0 y 1 a los elementos redundantes, y la tercer columna asigna los valores 1 y 0 a los elementos redundantes.

Tabla 2.11: Posibles permutaciones derivadas usando elementos redundantes. La primer columna presenta las posiciones de los elementos redundantes, la segunda y tercera columnas presentan la asignación de valores a los elementos redundantes.

Posiciones de los elementos redundantes	Asignación (0,1)	Asignación (1,0)
- - 2 3 4	**0 1** 2 3 4	**1 0** 2 3 4
- 2 - 3 4	**0** 2 **1** 3 4	**1** 2 **0** 3 4
- 2 3 - 4	**0** 2 3 **1** 4	**1** 2 3 **0** 4
- 2 3 4 -	**0** 2 3 4 **1**	**1** 2 3 4 **0**
2 - - 3 4	2 **0 1** 3 4	2 **1 0** 3 4
2 - 3 - 4	2 **0** 3 **1** 4	2 **1** 3 **0** 4
2 - 3 4 -	2 **0** 3 4 **1**	2 **1** 3 4 **0**
2 3 - - 4	2 3 **0 1** 4	2 3 **1 0** 4
2 3 - 4 -	2 3 **0** 4 **1**	2 3 **1** 4 **0**
2 3 4 - -	2 3 4 **0 1**	2 3 4 **1 0**

2.14 Resumen

Además de definir de manera formal el concepto de SCAs, se definieron los conceptos de SCAN, y SCAK. Los dos casos óptimos conocidos ($t = 2$ y $(M = t!) \wedge (k = t + 1)$) fueron presentados. De manera relevante se presentaron tres posibles representaciones de una permutación: como permutación, como permutación inversa y como un grafo acíclico dirigido. La evaluación, verificación, y como medir las subsecuencias faltantes fueron presentadas. El concepto de isomorfismos a través

de la presentación de las tres simetrías (de renglón, de columnas, y de variables) fue presentado. Este capítulo finaliza abordando el tema de elementos redundantes en un SCA.

3

Estado del Arte

En este capítulo se revisa el estado del arte relacionado a la construcción de SCAs organizados de acuerdo al tipo de algoritmo en: algoritmos avaros, algoritmos exactos, algoritmos de post-optimización y algoritmos metaheurísticos.

3.1 Algoritmos avaros

Los algoritmos avaros son simples de implementar y entregan resultados de una manera rápida. Estos algoritmos trabajan de manera incremental, inician con una solución aleatoria y la van modificando a través de la generación de un conjunto de elementos candidatos y la selección del mejor de ellos. En ocasiones los algoritmos avaros producen la solución óptima para un determinado problema [5]. Existen algoritmos avaros utilizados para la construcción de CAs como los que se presentan en [8, 21, 58] que son implementados de igual manera para la construcción de SCAs. En esta sección se presentan algunos de los algoritmos avaros desarrollados para la construcción de SCAs.

3.1.1 Algoritmo T-SEQ

Kuhn *et al.* presentan en [32] un algoritmo avaro para construir SCAs. Este algoritmo fue motivado por el problema de evaluar el comportamiento del software en pruebas de sistemas que aceptan múltiples conexiones de sensores y generan salida a varios enlaces de interfaz. En este problema es importante el orden en el que ocurren las k variables, pensando en una relación donde t variables son evaluadas para cada posible $t!$ orden. El Algoritmo funciona implementando los siguientes pasos:

1. Generación de un conjunto de candidatos (permutaciones aleatorias de k variables).
2. Selección del candidato que cubra más subsecuencias faltantes.
3. Adición del candidato seleccionado al cuasi-SCA.
4. Inversión del candidato seleccionado.
5. Adición del candidato invertido al cuasi-SCA.

Con este algoritmo se construyeron SCAs para $k = \{5, 6, ..., 80\}$ variables y con una fuerza de interacción de $t = \{3, 4\}$.

3.1.2 Algoritmo avaro de iteración simple

Kuhn *et al.* presentan en [31] la construcción de SCAs por medio de un algoritmo avaro simple por medio de un enfoque similar al presentado en [32, 28]. Este algoritmo funciona generando un conjunto de pruebas candidatas a ser agregadas al SCA, luego se agrega la prueba que cubra el mayor número de subsecuencias faltantes, repitiendo este proceso hasta que se haya cubierto todas las subsecuencias posibles. Richard Kuhn *et al.* mencionan que "*Este algoritmo produce SCAs de menor tamaño que otros desarrollados hasta ahora*", este algoritmo se desarrolló en el año 2017. Este algoritmo fue utilizado para construir SCAs de $k = \{5, 6, ..., 80\}$ variables y con una fuerza de $t = \{3, 4\}$.

3.1.3 Uso de Answer-Set Programing (ASP) para la construcción de SCAs

Answer-Set Programing (ASP) es una forma de programación declarativa, este tipo de programación se orienta a problemas de búsqueda difíciles, se basa en ideas que han llevado a la creación de solucionadores rápidos para el problema de satisfactibilidad [38]. Existen diversos usos del ASP para la construcción de SCAs como se muestra en [20], por otro lado, Esra Erdem *et al.* en [19] construyen SCAs utilizando ASP solver CLASP [22] con una codificación SCA en específico, utilizando un enfoque que permita establecer criterios de cobertura complejos de manera que pequeñas variaciones de la especificación de un problema requieren solo pequeñas modificaciones de la representación ASP y al hacer esto de manera concisa, logran la construcción de SCAs de fuerza $t = 3$.

Por otro lado, Mutsunori Banbara *et. al* presentan en [1] un método de construcción de SCAs por medio de un programa ASP del modelo de matriz de incidencia y Programación Lineal Entera (CP). Mutsunori describe el procedimiento de su método de la siguiente manera:

- Primero se presentan tres modelos de CP desde diferentes puntos de vista de matrices de cobertura de secuencias: 1. Modelo de matriz ingenuo, 2. Modelo de matriz de posición de variable y 3. Modelo de matriz de incidencia.

- Se representa un SCA mediante una matriz de incidencia (0, 1) denominada matriz de incidencia de la matriz, en esta matriz las restricciones de cobertura del SCA se pueden expresar de forma concisa.

- Se presenta un programa ASP del modelo de matriz de incidencia, compacto que refleja fielmente las limitaciones originales del modelo de matriz de incidencia.

Utilizando este enfoque se lograron mejorar resultados para fuerza $t = 3$ y se lograron encontrar resultados óptimos para fuerza $t = 4$.

3.1.4 Algoritmo de adición de permutaciones basadas en la cobertura

Ramírez Acuña presenta en [18] un algoritmo avaro para la construcción de SCAs donde dado un conjunto de permutaciones candidatas generadas por métodos de construcción aleatorios, se selecciona la permutación que cubra el mayor número de subsecuencias faltantes. En el **Algoritmo 5** se presenta el pseudocódigo para la construcción de SCAs implementada en [18]. El funcionamiento consta de generar candidatos aleatorios a ser agreagados, los candidatos se generan con una función que construye un grafo en base a las subsecuencias faltantes y por medio de un ordenamiento topológico genera una permutación. Se selecciona la permutación con la cual se cubra el mayor número de subsecuencias faltantes y el programa termina cuando no falten subsecuencias por cubrir.

Algoritmo 5 Algoritmo avaro de adición de permutaciones basadas en la cobertura para fuerza t y k variables

1: $C \leftarrow$ número de candidatos.
2: $F \leftarrow$ Umbral de fallos.
3: $\mathbb{S} \leftarrow$ Subsecuencia faltante.
4: $\mathbf{G} \leftarrow$ Grafo acíclico dirigido.
5: $\mathbf{SCA} \leftarrow$ Sequence Covering Array generado.
6: **hacer**{
7: $mejor\_candidato := \{\emptyset\}$;
8: $intentos := 0$;
9: $\forall\ i\ \in\ [0, ..., C\text{-}1]$\{
10: $\mathbf{G} := \{\emptyset\}$;
11: **hacer**{
12: $\mathbb{S} :=$ Subsecuencia faltante;
13: **si** $(\mathbf{G} \cup \mathbb{S} =$ GAD$)$ $\mathbf{G} := \mathbf{G} \cup \mathbb{S}$;
14: **si no** $intentos := intentos + 1$;
15: }**mientras** $(intentos < F)$;
16: $candidato := ordenamiento\_topologico(\mathbf{G})$;
17: **si** $(mejor\_candidato = \{\emptyset\})$ $mejor\_candidato := candidato$;
18: **si no. si** $(candidato._{faltantes} < mejor\_candidato._{faltantes})$\{
19: $mejor\_candidato := candidato$;
20: }
21: }
22: $\mathbf{SCA} := \mathbf{SCA} \oplus mejor\_candidato$;
23: }**mientras** $(faltantes > 0)$;

3.2 Algoritmos Exactos

Los algoritmos exactos en general aseguran la convergencia a resultados óptimos, están basados en el uso de técnicas analíticas y matemáticas, garantizan obtener la solución óptima. Estos algoritmos son diseñados bajo características especificas como continuidad, espacio de búsqueda, diferenciabilidad, entre otros [41].

3.2.1 Construcción de SCAs de fuerza t y (t+1) variables

Levenshtein [37] desarrolló un algoritmo basado en la idea presentada en [9, 57] tal que, dado el conjunto de permutaciones de tamaño $t+1$, se logran construir $t+1$ subconjuntos de tamaño $t!$, donde cada uno de estos subconjuntos es un SCA. Una permutación $\mathbb{R}$ pertenece al SCA α si se cumple la ecuación (3.1):

$$(\sum_{i=0}^{t-1}(i+1)\zeta(\mathbb{R}_i,\mathbb{R}_{i+1})) \equiv \alpha \mod (t+1), \text{ donde } \zeta(x,y)=\begin{cases}1 & \textbf{si } x>y\\ 0 & \textbf{si } x\leq y\end{cases} \tag{3.1}$$

Utilizando la ecuación (3.1) se desarrolla un algoritmo capaz de construir $SCAN(t,t+1)=t!$, en el **Algoritmo 6** se presenta el pseudocódigo para la construcción de los SCAs óptimos de fuerza t, $k=t+1$ variables y $t!$ permutaciones.

Algoritmo 6 Algoritmo para la construcción de SCA óptimos de fuerza t y $k=t+1$ variables.

1: $\mathbb{R} \leftarrow$ Permutación a clasificar.
2: $\forall\ i\ \in\ [0,...,t\text{-}2]\ SCA_i := \{\emptyset\}$;
3: $\forall\ \mathbb{R} \in \{(t+1)!\}\{$
4: $\quad s := 0$;
5: $\quad \forall\ i\ \in\ [0,...,t\text{-}2]\{$
6: $\quad\quad s := s + (i+1)\zeta(\mathbb{R}_i,\mathbb{R}_{i+1})$;
7: $\quad\}$
8: $\quad SCA_{s \mod (t+1)} = SCA_{s \mod (t+1)} \oplus \mathbb{R}$;
9: $\}$

En la Tabla 3.1 se presentan los SCAs óptimos resultantes del conjunto de permutaciones de

fuerza $t = 3$ y $k = 4$ variables, lo cual da como resultado 4 $SCAN(3, 4) = 6$ óptimos.

Tabla 3.1: SCAs óptimos generados para $t = 3$ y $k = 4$ utilizando el algoritmo de casos óptimos $SCAN(t, t + 1) = t!$.

Clase 0	Clase 1	Clase 2	Clase 3
0 2 1 3	0 3 2 1	0 1 2 3	0 1 3 2
0 3 1 2	1 0 2 3	1 0 3 2	0 2 3 1
1 2 0 3	1 3 2 0	2 0 3 1	1 2 3 0
1 3 0 2	2 0 1 3	2 1 3 0	2 1 0 3
2 3 0 1	2 3 1 0	3 0 2 1	3 1 0 2
3 2 1 0	3 0 1 2	3 1 2 0	3 2 0 1

3.2.2 Construcciones de SCAs de fuerza 3

Tauri en [51] reporta un algoritmo similar con conjuntos de t valores completamente desordenados. Con este algoritmo se logran reportar SCAs de fuerza $t = 3$. Por otro lado, en [10] inspirado en los trabajos de [15, 14] se presenta la construcción de SCAs utilizando la representación inversa del SCA, y se logra demostrar que para la existencia de un $SCA^{-1}(M; 3, k)$ y $SCA^{-1}(R; 3, w)$ se puede construir un $SCA^{-1}(M + R; 3, kw)$.

3.2.3 Construcción de SCAs de fuerza t y (t+2) variables

Para reproducir los resultados reportados en [40] Ramírez Acuña implementó en [18] un método para la construcción de SCAs de fuerza t y $k = t + 2$ mediante un algoritmo exacto de ramificación y poda que realiza búsqueda no-isomorfa.

3.3 Algoritmos de post-optimización

Los algoritmos de post-optimización son utilizados para reducir el número de permutaciones en un SCA, basándose en la sobre cobertura de subsecuencias, es decir que existen elementos redundantes.

Charles J. Colbourn *et. al* [43] presentan un algoritmo de post-optimización de SCAs, donde se hace uso de un elemento llamado *cobertura efectiva*, que indica el número de subsecuencias

que cubre únicamente la permutación evaluada. Se hace una evaluación de la cobertura efectiva de cada permutación y se ordenan las permutaciones de manera tal que las permutaciones que cubran el menor número de subsecuencias únicas son posicionadas al final de la matriz. Si una de las permutaciones evaluadas contiene cero subsecuencias únicas, entonces es eliminada automáticamente y se actualiza la matriz de cobertura. Al terminar la evaluación y el ordenamiento, se elimina una de las permutaciones con las que se pierda el menor número de subsecuencias y se cubren las subsecuencias perdidas usando las permutaciones restantes que contengan elementos redundantes. Con este algoritmo se logró mejorar cotas para SCAs con fuerza $t = \{4, 5\}$ y se definieron nuevas cotas para fuerza $t = 6$.

3.4 Algoritmos metaheurísticos

El término metaheurística es un término compuesto por las palabras "meta"(más allá; a un nivel superior) y "heurística"(encontrar). Glover introduce en [24] el concepto de metaheurística como metodologías generales de nivel superior (plantillas) que se pueden utilizar como estrategias de guía en el diseño de heurísticas subyacentes para resolver problemas de optimización específicos. Por otro lado, en [2] este tipo de algoritmos se definen como un proceso de generación iterativo que guía una heurística combinando conceptos para explorar y explotar el espacio de búsqueda. Las metaheurísticas engloban algunos algoritmos como: Optimización de colonias de hormigas, Computación evolutiva, incluidos algoritmos genéticos, búsqueda local iterada, recocido simulado y búsqueda tabú. En esta sección se detallan algoritmos metaheurísticos para construir SCAs.

3.4.1 Algoritmo de colonia de abejas

Los algoritmos de colonia de abejas funcionan iniciando con determinado número de abejas ubicadas aleatoriamente en el espacio de búsqueda, y después siguen la estructura de:

- Evaluar la aptitud de la población.
- Mientras no se cumpla el criterio de terminación, se genera una nueva población.

- Seleccionar sitios de búsqueda vecinos.
- Reclutar abejas para sitios selectos, evaluar aptitud.
- Seleccionar la abeja más apta de cada grupo.
- Asignar nuevos sitios de búsqueda a las abejas restantes y evaluar la aptitud.

Estos pasos son los utilizados para realizar un algoritmo de colonia de abejas ver [47].

Zabill *et. al* presentan en [42] un algoritmo metaheurístico basado en una colonia de abejas para construir SCAs. Como ya se había mencionado, este tipo de algoritmo simula el trabajo que realizan las abejas en la búsqueda de comida, donde la comida es la solución óptima que se está buscando. Se logran construir soluciones SCAs para fuerza $t = \{3, 4, 5\}$ y con un alfabeto $k = \{t, ..., 10\}$ pero no se logran reportar resultados que mejoren los resultados obtenidos por otros algoritmos.

3.4.2 Algoritmo de recocido simulado

Los algoritmos de recocido simulado (Simulated annealing) como su nombre lo dice, son algoritmos de búsqueda que simulan el proceso de recocido de metales disminuyendo lentamente la temperatura de un metal inicialmente a altas temperaturas. El proceso de recocido consiste en fundir el sistema que se desea optimizar a una temperatura efectiva, luego bajar por etapas lentas la temperatura hasta que el sistema termine congelado, es decir que ya no ocurran más cambios. Por cada temperatura, la simulación continua el tiempo suficiente para que el sistema alcance un estado estable [29]. En [18] se presenta un ejemplo de algoritmo de recocido simulado para la construcción de SCAs, donde se recibe un SCA con subsecuencias faltantes, y realizando determinadas modificaciones en las permutaciones del SCA logra cubrir las subsecuencias faltantes. Primero se evalúa una de las subsecuencias faltantes y se intenta agregar a una permutación por medio de la representación de grafos, donde si el grafo resultante de unir el grafo de la permutación con los arcos de la subsecuencia da como resultado un grafo acíclico dirigido, entonces se acepta la adición de la subsecuencia faltante. Los criterios de aceptación del recocido simulado implementado se basan en la reducción de las subsecuencias faltantes, de lo contrario, se basa en el criterio de aceptación de la distribución de

Boltzmann. Los parámetros de ejecución del recocido simulado fueron tomados de [55], donde se define una temperatura inicial de 4000; una temperatura final de $1{,}0E-10$; factor de congelamiento de $0,9$; $L=(Mk)^2$ soluciones vecinas máximas por temperatura donde M es el total de renglones, k el número de variables; un factor de congelado de 11; y un total de $2(Mk)^2$ movimientos.

3.5 Resumen

En este capítulo se presentaron diversos algoritmos para la construcción de SCAs. Los algoritmos presentados fueron clasificados en: avaros, exactos, de post-optimización y metaheurísticos. A través de la utilización de los algoritmos de manipulación de SCAs para agregar/borrar renglones/columnas y el algoritmo de recocido simulado se realizará la construcción de SCAs de fuerza 3, este es el tema tratado en el siguiente capítulo.

4

Desarrollo de algoritmos para la construcción de SCAs

En este capítulo se presentan los algoritmos desarrollados para la construcción de SCAs. Se detallan dos algoritmos para manipular SCAs: *moveRowSCA* y *moveColSCA*, de manera adicional se presenta un algoritmo de recocido simulado para poder reducir a cero las subsecuencias faltantes en un cuasi-SCA. Finalmente se presenta el algoritmo testSCA para verificar un SCA.

4.1 moveRowSCA: Algoritmo para agregar/quitar renglones de un SCA

El algoritmo *moveRowSCA* fue desarrollado con el propósito de manipular un SCA/cuasi-SCA a través de la eliminación y/o adición de renglones con el criterio de que el SCA/cuasi-SCA resultante tenga el menor número de subsecuencias faltantes. Sea $\mathbf{A}$ un SCA/cuasi-SCA, $\mathbf{P}$ su matriz de cobertura de tamaño $\binom{k}{t} \times t!$, DEL el número de renglones por eliminar, ADD el número de renglones por agregar, e $ITERS$ el número de veces que se realiza el proceso de eliminación y/o adición de

renglones (con el objetivo de generar el SCA/cuasi-SCA con el menor número de $faltantes$. En la operación del *moveRowSCA* pueden ocurrir los siguientes casos:

- Cuando $(DEL = 0) \wedge (ADD > 0)$. Esto demanda que se construya una lista $\mathbf{L}$ de tamaño $||(\mathbf{P}_{i,j} = 0)|| \times t$, en la cual se almacenan las subsecuencias faltantes del SCA/cuasi-SCA inicial. Se realiza la construcción de ADD grafos acíclicos dirigidos (GADs) en los cuales se intentarán agregar las subsecuencias faltantes de la lista $\mathbf{L}$ (si una subsecuencia no puede ser agregada a uno de los GADs se intenta agregar en el siguiente). Al final se realiza el ordenamiento topológico usando los GADs y se generan las ADD permutaciones equivalentes. Si $ITERS > 1$ se repite el proceso de acuerdo a su valor.

- Cuando $(DEL > 0) \wedge (ADD = 0)$. $\forall \mathbb{D} \in \{\binom{M}{DEL}\}$ se prueba eliminar DEL renglones y se selecciona la opción que reduzca al máximo las subsecuencias faltantes del resultado.

- Cuando $(DEL > 0) \wedge (ADD > 0)$. Para cada caso de eliminación de renglones se prueban todas las maneras de agregar los renglones. Para cada conjunto de renglones definidos por $\mathbb{D} \in \{\binom{M}{DEL}\}$, se define $\mathbf{L} = \mathbf{L} \oplus \mathbf{U}$, donde $\mathbf{U}$ representa las subsecuencias únicas cubiertas por los renglones definidos en $\mathbb{D}$, una vez definida la nueva lista $\mathbf{L}$, se realiza el procedimiento del caso $(DEL = 0) \wedge (ADD > 0)$ para generar las permutaciones que se agregarán al SCA', al finalizar la simulación se habrá generado un $SCA'(M\text{-}DEL{+}ADD; t, k)$ por cada $\mathbb{D} \in \{\binom{M}{DEL}\}$. La ejecución termina exportando el SCA' que presente el menor número de subsecuencias faltantes.

En el **Algoritmo 7** se presenta el procedimiento implementado por *moveRowSCA*. Se han definido dos casos generales para la ejecución de este algoritmo, si se realiza eliminación de renglones, entonces se realiza la comparación de $(\binom{M}{DEL} * ITERS)$ SCAs y se exporta el mejor, por otro lado, si solo se realiza adición de renglones, entonces se realiza la comparación de $(ITERS)$ SCAs durante la ejecución y se exporta el que contenga mayor cobertura.

Algoritmo 7 moveRowSCA($\mathbf{A}, DEL, ADD$)

Entrada: $\mathbf{A}$: $SCA(M; t, k)$ base al que se agregan y/o eliminan renglones.
Entrada: DEL : Total de renglones por eliminar.
Entrada: ADD : Total de renglones por agregar.
Salida: $\mathbf{SCA}$: Mejor SCA' generado con la adición y eliminación de renglones.

1: ———————— Variables y Estructuras de ejecución ————————
2: $ITERS \leftarrow$ Parámetro de ejecución opcional con valor de 1 por omisión.
3: $\mathbb{D} \leftarrow$ Conjunto de renglones por eliminar.
4: $\mathbf{A}' \leftarrow$ SCA modificado por eliminación y/o adición de renglones.
5: $\mathbf{P} \leftarrow$ Matriz de cobertura correspondiente al SCA base.
6: $\mathbf{L} \leftarrow$ Lista de subsecuencias faltantes.
7: $i \in [0, ..., \binom{k}{t}\text{-}1]$
8: $j \in [0, ..., t!\text{-}1]$
9: ———————— Procedimiento ————————
10: $\forall\ iter\ \in\ [0, ..., ITERS\text{-}1]\{$
11: **si** $(DEL > 0)\{$
12: $\forall\ \mathbb{D} \in \{\binom{M}{DEL}\}\{$
13: $A' :=$ **eliminar_renglon**$(\mathbf{A}, \mathbf{P}, \mathbf{L}, \mathbb{D}, DEL, ADD)$;
14: **si** $(\nexists\ \mathbf{SCA})\ \mathbf{SCA} := \mathbf{A}'$;
15: **si no. si** $(\mathbf{A}'_{.faltantes} < \mathbf{SCA}_{.faltantes})\ \mathbf{SCA} := \mathbf{A}'$;
16: $\}$
17: $\}$
18: **si** $(DEL = 0) \wedge (ADD > 0)\{$
19: $faltantes := \forall\ i, j\ ||(\mathbf{P}_{i,j} = 0)||$;
20: $\forall\ i, j\ \mathbf{P}'_{i,j} := \mathbf{P}_{i,j}$;
21: $\forall\ r\ \in\ [0, ..., faltantes\text{-}1]\ \forall\ c\ \in\ [0, ..., t\text{-}1]\ \mathbf{L}'_{r,c} := \mathbf{L}_{r,c}$;
22: $A' :=$ **agregar_renglon**$(\mathbf{A}, \mathbf{P}', \mathbf{L}', ADD)$;
23: **si** $(\nexists\ \mathbf{SCA})\ \mathbf{SCA} := \mathbf{A}'$;
24: **si no. si** $(\mathbf{A}'_{.faltantes} < \mathbf{SCA}_{.faltantes})\ \mathbf{SCA} := \mathbf{A}'$;
25: $\}$
26: $\}$
27: **devolver** $\mathbf{SCA}$;

La función de *eliminar_renglon* presentada en el **Algoritmo 8** recibe las estructuras del SCA base $(\mathbf{A}, \mathbf{P}, \mathbf{L})$, el vector $\mathbb{D}$ que indica los renglones que deben ser eliminados, DEL que indica el número de renglones a eliminar y ADD el número de renglones por agregar. Para llevar a cabo la simulación de eliminación, primero se respaldan las estructuras del SCA base, después se decrementa el contador de las subsecuencias cubiertas por las permutaciones eliminadas, se define el SCA resultante $\mathbf{A}^{-1}$ como el SCA base menos los renglones eliminados. Si $ADD > 0$ entonces se procede a agregar ADD

permutaciones al SCA generado, y al finalizar se retorna el SCA generado de la eliminación y adición de renglones.

Algoritmo 8 eliminar_renglon($\mathbf{A}, \mathbf{P}, \mathbf{L}, \mathbb{D}, DEL, ADD$)

Entrada: $\mathbf{A}$: $SCA(M; t, k)$ inicial al que se le aplica la eliminación y/o adición de renglones.
Entrada: $\mathbf{P}$: Matriz de cobertura del SCA inicial.
Entrada: $\mathbf{L}$: Matriz de subsecuencias faltantes del SCA inicial.
Entrada: $\mathbb{D}$: Conjunto de renglones a eliminar.
Entrada: DEL : Total de renglones a eliminar.
Entrada: ADD : Total de renglones por agregar.
Salida: $\mathbf{A}'$: $SCA(M\text{-}DEL + ADD; t, k)$ generado.

1: ———————— Variables y Estructuras de ejecución ————————
2: $\mathbb{R} \leftarrow$ Almacena la permutación que se está eliminando.
3: $\mathbb{C} \leftarrow$ Indica el conjunto de columnas de $\mathbb{R}$ a evaluar.
4: $\mathbb{S} \leftarrow$ Indica la subsecuencia que se ha eliminado.
5: $\mathbb{V} \leftarrow$ Indica el conjunto de variables que conforman la subsecuencia eliminada.
6: $\mathbf{P}' \leftarrow$ Respaldo de la matriz de cobertura inicial.
7: $\mathbf{L}' \leftarrow$ Respaldo de la matriz de subsecuencias faltantes inicial.
8: ———————— Procedimiento ————————————————
9: $\forall\ i\ \in\ [0, ..., \binom{k}{t}\text{-}1]\ \forall\ j\ \in\ [0, ..., t!\text{-}1]\ \mathbf{P}'_{i,j} := \mathbf{P}_{i,j}$;
10: $\forall\ \ \in\ [0, ..., \mathbf{A}._{faltantes}\text{-}1]\ \forall\ j\ \in\ [0, ..., t\text{-}1]\ \mathbf{L}'_{i,j} := \mathbf{L}_{i,j}$;
11: $\forall\ r\ \in\ [0, ..., DEL]\{$
12: $\quad \mathbb{R} := \mathbf{A}_{\mathbb{D}_r}$;
13: $\quad \forall\ \mathbb{C} \in \{\binom{k}{t}\}\{$
14: $\quad\quad \forall\ i\ \in\ [0, ..., t\text{-}1]\ \mathbb{S}_i := \mathbb{R}_{\mathbb{C}_i}$;
15: $\quad\quad \mathbb{V} :=$ **ordenar**$(\mathbb{S})$;
16: $\quad\quad evn :=$ **DirectGTP**$(\mathbb{V}, t)$;
17: $\quad\quad sec :=$ **subsecuencia_a_entero**$(\mathbb{R}, \mathbb{C})$;
18: $\quad\quad \mathbf{P}'_{evn,sec} := \mathbf{P}'_{evn,sec}\text{-}1$;
19: $\quad\quad$ **si** $(\mathbf{P}'_{evn,sec} = 0)\{$
20: $\quad\quad\quad \mathbf{L}' := \mathbf{L}' \oplus \mathbb{S}$;
21: $\quad\quad \}$
22: $\quad \}$
23: $\}$
24: $\mathbf{A}' := \mathbf{A} \ominus \{\mathbf{A}_r \mid r \in \mathbb{D}\}$;
25: **si** $(ADD > 0)$ $\mathbf{A}' :=$ **agregar_renglon**$(\mathbf{A}', \mathbf{P}', \mathbf{L}', ADD)$;
26: **devolver** $\mathbf{A}'$;

El **Algoritmo 9** Se utiliza para ordenar un conjunto de valores $\mathbb{X}$ desordenado (sin valores repetidos), y devuelve un conjunto de valores $\mathbb{X}'$ ordenados de menor a mayor.

Algoritmo 9 ordenar($\mathbb{X}$)

Entrada: $\mathbb{X}$: Conjunto de valores desordenado.
Salida: $\mathbb{X}'$: Conjunto de valores ordenado de menor a mayor.

1: ——————— Variables y Estructuras de ejecución ———————
2: $\mathbb{E} \leftarrow$ Indica los valores de k que se encuentran en el vector $\mathbb{X}$.
3: ——————————— Procedimiento ———————————
4: $\forall\ i\ \in\ [0, ..., k\text{-}1]\ \mathbb{E}_i := \text{-}1;$
5: $\forall\ i\ \in\ [0, ..., t\text{-}1]\ \mathbb{E}_{\mathbb{X}_i} := 1;$
6: $idx := 0;$
7: $\forall\ i\ \in\ [0, ..., k\text{-}1]\{$
8: **si** $(\mathbb{E}_i = 1)\{$
9: $\mathbb{X}'_{idx} := i;$
10: $idx := idx + 1;$
11: $\}$
12: $\}$
13: **devolver** $\mathbb{X}'$;

En el **Algoritmo 10** se evalúa la subsecuencia cubierta por la permutación $\mathbb{R}$ con el conjunto de columnas $\mathbb{C}$. Se evalúa la subsecuencia $\mathbb{R}_{\mathbb{C}_{0,..,t\text{-}1}}$ y se define $\mathbb{S}$ como la subsecuencia cubierta, una vez definida la subsecuencia $\mathbb{S}$, se define la notación inversa de $\mathbb{S}$ como $\mathbb{S}^{\text{-}1}$, se define la notación factorial de $\mathbb{S}$ y se almacena en el vector $\mathbb{F}$, finalmente, se utiliza la función **Ruffini**$(\mathbb{F}, t)$ para convertir la notación factorial en un valor entero positivo que indique el contador de la subsecuencia evaluada en el vector de cobertura $\mathbb{P}$.

Algoritmo 10 subsecuencia_a_entero($\mathbb{R}, \mathbb{C}$)

Entrada: $\mathbb{R}$: Renglón del SCA que cubre la subsecuencia a evaluar.
Entrada: $\mathbb{C}$: Conjunto de columnas del SCA donde se encuentra la subsecuencia.
Salida: Valor entero positivo que representa la subsecuencia evaluada.

1: ———————— Variables y Estructuras de ejecución ————————————
2: $\mathbb{S} \leftarrow$ Secuencia cubierta.
3: $\mathbb{S}^{-1} \leftarrow$ Notación inversa de $\mathbb{S}$.
4: $\mathbb{F} \leftarrow$ Notación factorial de $\mathbb{S}$.
5: $\mathbb{E} \leftarrow$ Vector que indica el orden de cada variable de $\mathbb{R}$ para el conjunto de columnas $\mathbb{C}$.
6: ———————— Procedimiento ————————————————————
7: $\forall\ i\ \in\ [0, ..., k\text{-}1]\ \mathbb{E}_i := \text{-}1;$
8: $\forall\ i\ \in\ [0, ..., t\text{-}1]\ \mathbb{E}_{\mathbb{R}_{\mathbb{C}_i}} := 1;$
9: $idx := 0;$
10: $\forall\ i\ \in\ [0, ..., k\text{-}1]\{$
11: **si** $(\mathbb{E}_i = 1)\{$
12: $\mathbb{E}_i := idx;$
13: $idx := idx + 1;$
14: $\}$
15: $\}$
16: $\forall\ i\ \in\ [0, ..., t\text{-}1]\ \mathbb{S}_i := \mathbb{E}_{\mathbb{R}_{\mathbb{C}_i}};$
17: $\forall\ i\ \in\ [0, ..., t\text{-}1]\ \mathbb{S}^{-1}_{\mathbb{S}_i} = i;$
18: $\forall\ i\ \in\ [0, ..., t\text{-}2]\{$
19: **si** $(\mathbb{S}_i = i)\ \mathbb{F}_i := 0;$
20: **si** $(\mathbb{S}_i \neq i)\{$
21: $\mathbb{F}_i := \mathbb{S}^{\text{-}1}_i\text{-}\mathbb{S}^{\text{-}1}_{\mathbb{S}_i};$
22: $temp := \mathbb{S}_i;$
23: $\mathbb{S}_i := \mathbb{S}_{i+\mathbb{F}_i};$
24: $\mathbb{S}_{i+\mathbb{F}_i} := temp;$
25: $temp := \mathbb{S}^{\text{-}1}_i;$
26: $\mathbb{S}^{\text{-}1}_i := \mathbb{S}^{\text{-}1}_{\mathbb{S}i+\mathbb{F}_i};$
27: $\mathbb{S}^{\text{-}1}_{\mathbb{S}i+\mathbb{F}_i} := temp;$
28: $\}$
29: $\}$
30: **devolver Ruffini**$(\mathbb{F}, t);$

El procedimiento de adición de permutaciones se presenta en el **Algoritmo 11**, se reciben las estructuras del SCA $(\mathbf{A}, \mathbf{P}, \mathbf{L})$ al que se le van a agregar las nuevas permutaciones, además de recibir el valor de ADD para conocer el número de renglones por agregar; se define $\mathbb{X}$ como un ordenamiento aleatorio para recorrer la lista de subsecuencias faltantes; se inicializan los GADs como

vacíos para proceder a la adición de subsecuencias faltantes; para cada subsecuencia faltante de la lista $\mathbf{L}$, se genera un vector $\mathbb{Y}$ como un ordenamiento aleatorio de la prioridad de cada GAD para la adición de subsecuencias, si la subsecuencia faltante no se puede agregar a un GAD, entonces se intenta agregar al próximo GAD, y si un GAD logra almacenar una subsecuencia faltante, entonces se toma otra subsecuencia faltante y se intenta agregar a los GADs, este proceso se realiza hasta haber evaluado cada subsecuencia faltante. Una vez definidos los GADs, se procede a convertir cada GAD en una permutación por medio de un ordenamiento topológico, se agregan las subsecuencias cubiertas a la matriz de cobertura y se agregan las nuevas permutaciones al SCA.

Algoritmo 11 agregar_renglon($\mathbf{A}$, $\mathbf{P}$, $\mathbf{L}$,ADD)

Entrada: $\mathbf{A}$: $SCA(M; t, k)$ inicial al que se le aplica la eliminación y/o adición de renglones.
Entrada: $\mathbf{P}$: Matriz de cobertura del SCA inicial.
Entrada: $\mathbf{L}$: Matriz de subsecuencias faltantes del SCA inicial.
Entrada: ADD : Total de renglones por agregar.
Salida: $\mathbf{A}$: SCA inicial más ADD renglones más.

1: ——————— Variables y Estructuras de ejecución ———————
2: $\mathbb{C} \leftarrow$ Indica el conjunto de columnas de $\mathbb{R}$ a evaluar.
3: $\mathbb{R} \leftarrow$ Almacena una permutación después del ordenamiento topológico.
4: $\mathbb{S} \leftarrow$ Subsecuencia faltante perteneciente a la matriz $\mathbf{L}$.
5: $\mathbb{V} \leftarrow$ Indica el conjunto de variables que conforman la subsecuencia eliminada.
6: $\mathbb{X} :=$ Ordenamiento aleatorio del conjunto de valores $\{0, 1, ..., \mathbf{A}._{faltantes}\text{-}1\}$;
7: $\mathbb{Y} \leftarrow$ Ordenamiento aleatorio de prioridad de los GADs.
8: $\mathbf{G} \leftarrow$ Conjunto de grafos acíclicos dirigidos.
9: ——————— Procedimiento ———————
10: $\forall\ i\ \in\ [0, ..., ADD\text{-}1]\ \mathbf{G}_i := \emptyset$;
11: $\forall\ i\ \in\ [0, ..., \mathbf{A}._{faltantes}\text{-}1]\{$
12: $\quad \mathbb{Y} :=$ Ordenamiento aleatorio del conjunto de valores $\{0, 1, ..., ADD\text{-}1\}$;
13: $\quad \mathbb{S} := \mathbf{L}_{\mathbb{X}_i}$;
14: $\quad \forall\ j\ \in\ [0, ..., ADD\text{-}1]\{$
15: $\quad\quad$ **si** $((\mathbf{G}_{\mathbb{Y}_j} \cup \mathbb{S}) = \text{GAD})\{$
16: $\quad\quad\quad \mathbf{G}_{\mathbb{Y}_j} := (\mathbf{G}_{\mathbb{Y}_j} \cup \mathbb{S})$;
17: $\quad\quad\quad$ **termina ciclo** j;
18: $\quad\quad \}$
19: $\quad \}$
20: $\}$
21: $\forall\ i\ \in\ [0, ..., ADD\text{-}1]\{$
22: $\quad \mathbb{R} :=$ Orden topológico de $\mathbf{G}_i$;
23: $\quad \mathbf{A} := \mathbf{A} \oplus \mathbb{R}$;
24: $\quad \forall\ \mathbb{C} \in \{\binom{k}{t}\}\{$
25: $\quad\quad \forall\ i\ \in\ [0, ..., t\text{-}1]\ \mathbb{S}_i := \mathbb{R}_{\mathbb{C}_i}$;
26: $\quad\quad \mathbb{V} :=$ **ordenar**$(\mathbb{S})$;
27: $\quad\quad evn :=$ **DirectGTP**$(\mathbb{V}, t)$;
28: $\quad\quad sec :=$ **subsecuencia_a_entero**$(\mathbb{R}, \mathbb{C})$;
29: $\quad\quad \mathbf{P}_{evn,sec} := \mathbf{P}_{evn,sec}+1$;
30: $\quad \}$
31: $\}$
32: **devolver** $\mathbf{A}$;

4.2 moveColSCA: Algoritmo de manipulación de columnas en un SCA

El algoritmo *moveColSCA* como su nombre lo indica, tiene un funcionamiento similar a *moveRowSCA* con la diferencia de que en lugar de manipular renglones, se generan SCAs en base a la eliminación y/o adición de columnas, es necesario recalcar que la eliminación y/o adición de columnas, se realiza en la notación inversa del SCA base, es decir, *moveColSCA* elimina las variables del SCA base con las cuales, se generan el mayor número de subsecuencias faltantes. sea $\mathbf{A}$ el SCA base al que se agregan y/o eliminan columnas, $\mathbf{P}$ la matriz de cobertura del SCA base, DEL el total de columnas a eliminar, ADD el total de columnas por agregar, $ITERS$ el número de veces que se realiza el algoritmo de ajuste para generar el $SCA(M; t, k\text{-}DEL + ADD)$ con el menor número de subsecuencias faltantes, y $rowMoves$ define el total de posiciones en las que se puede posicionar la nueva variable. *moveColSCA* se divide en dos partes, la primera se encarga de eliminar las DEL columnas de $\mathbf{A}$ con las cuales se genere el $SCA(M; t, k\text{-}DEL)$ con el menor número de subsecuencias faltantes; la segunda parte del algoritmo es la encargada de agregar las ADD columnas al $SCA(M; t, k\text{-}DEL)$ para generar el $SCA(M; t, k\text{-}DEL + ADD)$ con el menor número de subsecuencias faltantes. En el **Algoritmo 12** se muestra el desarrollo de *moveColSCA* donde primero se eliminan las columnas que participan en el mayor número de subsecuencias faltantes y después se realiza la adición de columnas que maximicen la cobertura del SCA que se está generando.

Algoritmo 12 moveColSCA($\mathbf{A}, DEL, ADD$)

Entrada: $\mathbf{A}$: $SCA(M; t, k)$ base para eliminación y/o adición de columnas.
Entrada: DEL : Total de columnas a eliminar.
Entrada: ADD : Total de columnas por agregar.
Salida: $\mathbf{A}$: $SCA(M; t, k\text{-}DEL + ADD)$ generado.

```
1: ———————— Variables y Estructuras de ejecución ————————————
2: ITERS ← Parámetro de ejecución opcional con valor de 1 por omisión.
3: A^-1 ← Notación inversa de A.
4: P ← Matriz de cobertura de A.
5: ———————————— Procedimiento ————————————————————
6: si (DEL > 0){
7:    A := eliminar_columna(A, A^-1, DEL);
8: }
9: si (ADD > 0){
10:   A := agregar_columna(A, A^-1, P, ADD, ITERS);
11: }
12: devolver A;
```

4.2.1 Procedimiento de eliminación de columnas

Para llevar a cabo el proceso de eliminación de columnas, primero se declara una lista ligada $\mathbf{L}$ donde se almacenan los subconjuntos $\phi = \{\mathbb{V} \mid \mathbb{V} \in \{\binom{k}{t}\} \wedge \forall \mathbb{S} \in \{\mathbb{V}!\} \ ||(\mathbb{S} \not\subset \mathbf{A}_r)|| > 0\}$ para $0 \leq r < M$. $\mathbf{L}$ se define como una matriz de $||\{\phi\}|| \times (3t+1)$, donde para $\forall\ \mathbb{V} \in \phi$ la columna 0 almacena el total de subsecuencias de $\mathbb{V}$ que no han sido cubiertas por $\mathbf{A}$, el conjunto de columnas $\{1, ..., t\}$ almacenan el conjunto de variables $\mathbb{V}$, el conjunto de columnas $\{t+1, ..., 2t\}$ indica el renglón anterior de la lista ligada $\mathbf{L}$ en donde aparece cada variable de $\mathbb{V}$, finalmente el conjunto de columnas $\{2t+1, ..., 3t\}$ indica el próximo renglón en el que aparece cada variable del conjunto $\mathbb{V}$.

Una vez definida la lista ligada $\mathbf{L}$, se declara $\mathbf{E}$ como una matriz de tamaño $k \times 3$ que almacena información de cada variable, donde $\mathbf{E}_{i,0}$ indica el número de subsecuencias faltantes en las que participa la variable i, $\mathbf{E}_{i,1}$ indica el renglón de la lista $\mathbf{L}$ donde se encuentra la primera aparición de la variable i, y $\mathbf{E}_{i,2}$ indica el renglón de la lista $\mathbf{L}$ donde se encuentra la última aparición de la variable i, para $0 \leq i < k$.

Ejemplo 4.3.1.1. Sea $\{(2,1,3),\ (3,4,5),\ (2,1,6),\ (0,2,1),\ (0,1,2),\ (2,0,1),\ (4,3,5)\}$ el conjunto de subsecuencias faltantes, en la Tabla 4.1 se presenta la lista ligada resultante de las

subsecuencias faltantes, se puede observar que la columna **0** indica las subsecuencias que no han sido cubiertas, en las columnas **[1,...,t]** se encuentra el conjunto de variables que conforman las subsecuencias faltantes, en las columnas **[t+1,...,2t]** se encuentran los apuntadores a los renglones previos en los que aparece cada variable, y en las columnas **[2t,...,3t]** se encuentran los apuntadores a los renglones posteriores en los que aparece cada variable, se ha definido ∅ como indicador de que no existe ya sea el renglón anterior o el renglón posterior donde aparece cada variable del conjunto evaluado.

Tabla 4.1: Lista ligada resultante del conjunto de subsecuencias con baja cobertura {(2,1,3), (3,4,5), (2,1,6), (0,2,1), (0,1,2), (2,0,1), (4,3,5)} para $t = 3$ y $k = 7$.

0	**[1,...,t]**	**[t+1,...,2t]**	**[2t+1,...,3t]**
3	0 1 2	∅ ∅ ∅	∅ 1 1
1	1 2 3	0 0 ∅	2 2 3
1	1 2 6	1 1 ∅	∅ ∅ ∅
2	3 4 5	1 ∅ ∅	∅ ∅ ∅

En la Tabla 4.2 se presenta la matriz $\mathbf{E}$ resultante del conjunto de subsecuencias faltantes, podemos observar que el número de subsecuencias donde aparece cada variable, es la suma de las subsecuencias faltantes en cada conjunto donde aparece la variable evaluada. Para este ejemplo podemos resaltar que las variables 1 y 2 están involucradas en el mayor número de subsecuencias faltantes, por lo que son las primeras candidatas en ser eliminadas.

Tabla 4.2: Matriz auxiliar $\mathbf{E}$ resultante del conjunto de subsecuencias faltantes {(2,1,3), (3,4,5), (2,1,6), (0,2,1), (0,1,2), (2,0,1), (4,3,5)} para $t = 3$ y $k = 7$.

k	**faltantes donde aparece**	**primera aparición**	**última aparición**
0	3	0	0
1	5	0	2
2	5	0	2
3	3	1	3
4	2	3	3
5	2	3	3
6	1	2	2

El proceso de eliminación de una variable $0 \leq x < k$ se basa en la lista $\mathbf{L}$, eliminando los renglones en los que x pertenece al conjunto de variables con subsecuencias faltantes. Cada vez que se elimina un renglón de $\mathbf{L}$, se realiza una actualización de los apuntadores de la anterior y posterior aparición de la variable $y \in \mathbb{V}$ donde $y \neq x$. Finalmente, se actualiza $\mathbf{E}_y$, decrementando el contador de subsecuencias faltantes, la primera aparición y última aparición de y en la lista $\mathbf{L}$, después de la eliminación de la variable de x.

En el **Algoritmo 13** se presenta el procedimiento de eliminación de renglones de la lista $\mathbf{L}$ con subsecuencias faltantes para los conjuntos de variables $\{\mathbb{V} \mid x \in \mathbb{V}\}$.

Mientras r (índice del renglón de la lista $\mathbf{L}$ a eliminar) sea distinto de $\varnothing$, para cada variable $y \in \mathbb{V}$ que sea distinta de x, se decrementan $\mathbf{L}_{r,0}$ subsecuencias al contador $\mathbf{E}_{y,0}$, posteriormente, se ajustan los apuntadores de los renglones de la lista $\mathbf{L}$ (y los apuntadores de $\mathbf{E}$ de ser necesario) para cada variable y, donde los renglones que apuntaban a r, ahora apuntarán a los renglones que apuntaba el renglón r para cada variable y, de esta manera se elimina un renglón de la lista $\mathbf{L}$. Al finalizar, se definen los valores $\mathbf{E}_x$ como $\varnothing$, para indicar que la variable x ya no se encuentra en la lista $\mathbf{L}$ y será reemplazada en la matriz SCA.

Algoritmo 13 eliminar($x, \mathbf{L}, \mathbf{E}$)

Entrada: x : Variable que se elimina del SCA (columna que se elimina de la matriz inversa).
Entrada: $\mathbf{L}$: Lista ligada de los conjuntos de variables con subsecuencias faltantes.
Entrada: $\mathbf{E}$: Información de faltantes generadas, primera y última aparición de cada variable.

```
1: ———————— Variables y Estructuras de ejecución ————————
2: ∅ ← Valor no asignado.
3: ℝ ← Renglón de la lista ligada L que se va a eliminar.
4: ———————————— Procedimiento ————————————
5: r := E_{x,1};
6: hacer{
7:    ∀ c ∈ [0, ..., k-1] ℝ_c := L_{r,c};
8:    ∀ i ∈ [1, ..., t]{
9:       si (x = ℝ_i) r := ℝ_{i+(2t)};
10:      si no{
11:         E_{ℝ_i,0} := E_{ℝ_i,0}-ℝ_0;
12:         si (ℝ_{i+(t)} ≠ -1){
13:            ∀ c ∈ [1, ..., t]{
14:               si (L_{ℝ_{i+(t)},c} = ℝ_i) termina ciclo c;
15:            }
16:            L_{ℝ_{i+(t)},c+(2t)} := ℝ_{i+(2t)};
17:         }si no{
18:            si (ℝ_{i+(2t)} ≠ -1){
19:               E_{ℝ_i,1} := ℝ_{i+(2t)};
20:            }si no{
21:               E_{ℝ_i,1} := -1;
22:               E_{ℝ_i,2} := -1;
23:            }
24:         }
25:         si (ℝ_{i+(2t)} ≠ -1){
26:            ∀ c ∈ [1, ..., t]{
27:               si (L_{ℝ_{i+(2t)},c} = ℝ_i) termina ciclo c;
28:            }
29:            L_{ℝ_{i+(2t)},c+(t)} := ℝ_{i+(t)};
30:         }
31:      }
32:   }
33: }mientras (r ≠ ∅);
34: ∀ i ∈ [0, ..., t-1] E_{x,i} := ∅;
```

En el **Algoritmo 14** se presenta el procedimiento para eliminar las variables que intervienen en el mayor número de subsecuencias faltantes, generando un $\mathbf{A}'$ con DEL columnas menos. Se define la

variable a eliminar c como $max(\mathbf{E}_{i,0})$ donde $0 \leq i < k$, se agrega la variable a eliminar al conjunto $\mathbb{D}$, se actualiza la lista ligada $\mathbf{L}$ y la matriz $\mathbf{E}$ solo si aun quedan subsconjuntos con subsecuencias faltantes en $\mathbf{L}$. Finalmente, se eliminan las variables definidas en $\mathbb{D}$ y se hace un reetiquetado de las variables restantes del SCA.

Algoritmo 14 eliminar_columna($\mathbf{A}, \mathbf{A}^{-1}, DEL$)

Entrada: $\mathbf{A}$: Matriz $SCA(M; t, k)$ inicial.
Entrada: $\mathbf{A}^{-1}$: Matriz inversa de $\mathbf{A}$.
Salida: $\mathbf{A}'$: Matriz $SCA(M; t, k\text{-}DEL)$ generada.

```
1: ———————— Variables y Estructuras de ejecución ————————
2: 𝔻 ← Conjunto de columnas a eliminar del SCA dado.
3: L ← Lista ligada de conjuntos de variables con subsecuencias faltantes.
4: E ← Información de faltantes generadas, primera y última aparición de cada variable.
5: ———————————— Procedimiento ————————————
6: ∀ i ∈ [0, ..., DEL-1]{
7:    c := 0;
8:    ∀ j ∈ [0, ..., k-1]{
9:       si (E_{j,0} > E_{c,0}){
10:         c := j;
11:      }si no ((E_{j,0} = E_{c,0}) ∧ (aleatorio mód 2 = 0)){
12:         c := j;
13:      }
14:   }
15:   𝔻_i := c;
16:   si (E_{c,0} > 0) eliminar(c, L, E);
17:   si no E_{c,0} := ∅;
18: }
19: A' := reetiquetar(A, A^{-1}, 𝔻);
20: devolver A';
```

En el **Algoritmo 15** se presenta el procedimiento para eliminar las columnas del SCA definidas por $\mathbb{D}$, desde la línea 1 hasta la línea 13 se define el reetiquetado de cada variable, en la línea 16 se define $\mathbf{A}'_r$ como la permutación de tamaño $k\text{-}DEL$ de $\mathbf{A}_r$, en la línea 18 se realiza el reetiquetado de las variables $c \notin \mathbb{D}$ y en la línea 19 se define la permutación inversa de cada renglón del SCA con $k\text{-}DEL$ columnas.

Algoritmo 15 reetiquetar($\mathbf{A}, \mathbf{A}^{-1}, \mathbb{D}, DEL$)

Entrada: $\mathbf{A}$: Matriz $SCA(M; t, k\text{-}DEL)$ inicial.
Entrada: $\mathbf{A}^{-1}$: Matriz inversa de $\mathbf{A}$.
Entrada: $\mathbb{D}$: Conjunto de columnas por eliminar.
Entrada: DEL : Total de columnas a eliminar.
Salida: $\mathbf{A}'$: Matriz $SCA(M; t, k\text{-}DEL + ADD)$.

```
1: ———————— Variables y Estructuras de ejecución ————————
2: 𝕏 ← Define el nuevo valor para cada variable.
3: 𝕀 ← Permutación de k-DEL variables.
4: ———————————— Procedimiento ————————————
5: ∀ i ∈ [0, ..., k-1]{
6:    rest := 0;
7:    ∀ j ∈ 𝔻{
8:       si (i > j){
9:          rest := rest + 1;
10:      }si no. si (i = j){
11:         rest := i-(k-|𝔻|);
12:         termina ciclo j;
13:      }
14:   }
15:   𝕏_i := i-rest;
16: }
17: ∀ r ∈ [0, ..., M-1]{
18:   A'_r := {∅};
19:   A'_r = {∀ c ∈ [0, ..., k-1]A_{r,c} | A_{r,c} ∉ 𝔻};
20:   𝕀 = A'_r;
21:   ∀ c ∈ [0, ..., k-DEL-1] A'_{r,c} := 𝕏_{𝕀_c};
22:   ∀ c ∈ [0, ..., k-DEL-1] A^{-1}_{𝕀_{c,c}} := c;
23: }
24: devolver A';
```

4.2.2 Procedimiento de agregado de columnas

Para llevar a cabo el proceso de agregado de columnas, se recibe un $SCA(M; t, k)$ que puede ser el SCA base dado a *moveColSCA* o bien, puede ser el SCA resultante del proceso de eliminación de columnas. El procedimiento general de agregar una nueva columna consiste en: "Para cada permutación del SCA base, agregar la nueva variable n en la posición que genere la mayor cobertura del conjunto $\{(\binom{k\text{-}1}{t\text{-}1} \oplus n)t!\}$ de nuevas subsecuencias por cubrir", este procedimiento general se

realiza para cada nueva variable por agregar.

Ejemplo de posicionamiento. Sea: $SCAN(3,4) = 6$ un SCA óptimo al cual se desea agregar una nueva columna para generar un $SCA(6;3,5)$, (2 1 0 3) la permutación a la que se agrega la nueva variable $n = 4$, $\mathbb{C} = \{0,1,2,3,4\}$ el conjunto de posiciones que serán comparadas, y $\{(1\ 4\ 0), (4\ 0\ 3), (2\ 1\ 4)\}$ el conjunto de subsecuencias faltantes. En la Tabla 4.3 se presenta el resultado de agregar $n = 4$ en cada posición $\mathbb{C}$, donde la posición 2 genera la mayor cobertura de las subsecuencias faltantes.

Tabla 4.3: Comparación de cobertura de las subsecuencias $\{(1\ 4\ 0), (4\ 0\ 3), (2\ 1\ 4)\}$, al agregar $n = 4$ a la permutación $(2\ 1\ 0\ 3)$ en las posiciones $\mathbb{C} = \{0,1,2,3,4\}$.

Posición	**Permutación generada**	**subsecuencias de $\{\binom{k\text{-}1}{t\text{-}1} \oplus n\}$ cubiertas**	**Subsecuencias faltantes cubiertas**
0	(**4** 2 1 0 3)	(4 2 1), (4 2 0), (4 2 3) (4 1 0), (4 1 3), (4 0 3)	(4 0 3)
1	(2 **4** 1 0 3)	(2 4 1), (2 4 0), (2 4 3) (4 1 0), (4 1 3), (4 0 3)	(4 0 3)
2	(2 1 **4** 0 3)	(2 1 4), (2 4 0), (2 4 3) (1 4 0), (1 4 3), (4 0 3)	(2 1 4), (1 4 0), (4 0 3)
3	(2 1 0 **4** 3)	(2 1 4), (2 0 4), (2 4 3) (1 0 4), (1 4 3), (0 4 3)	(2 1 4)
4	(2 1 0 3 **4**)	(2 1 4), (2 0 4), (2 3 4) (1 0 4), (1 3 4), (0 3 4)	(2 1 4)

En el **Algoritmo 16** se presenta el procedimiento de construcción de SCAs por medio de la adición de columnas a un SCA base, donde se realiza la adición de nuevas variables de una en una. Para cada variable por agregar, se genera un vector de orden $\mathbb{O}$ aleatorio que indica el orden de los renglones para la adición de la nueva variable, para el primer renglón se genera una nueva permutación agregando la nueva variable en cualquier posición aleatoria, ya que, al ser la primera permutación aportará la misma cobertura para cualquier posición. Para el resto de permutaciones, la aportación de nuevas subsecuencias será más estricta para las últimas permutaciones del orden aleatorio, por lo que, se realiza una comparación de permutaciones, y se selecciona la que aporte la mayor cobertura de las nuevas subsecuencias. Al terminar de agregar la nueva variable en cada permutación, si $ITERS > 1$ entonces se realiza el proceso de ajuste, el cual es generar un nuevo

orden aleatorio $\mathbb{O}$, y para cada permutación evaluar un nuevo conjunto de posiciones $\mathbb{C}$ para tratar de incrementar la cobertura.

Algoritmo 16 agregar_columna($\mathbf{A}, \mathbf{A}^{-1}, \mathbf{P}, ADD, ITERS$)

Entrada: $\mathbf{A}$: Matriz $SCA(M; t, k\text{-}DEL)$ a la cual se agregarán las columnas.
Entrada: $\mathbf{A}^{-1}$: Matriz inversa de $\mathbf{A}$.
Entrada: $\mathbf{P}$: Matriz de cobertura de $\mathbf{A}$.
Entrada: ADD : Total de columnas por agregar.
Entrada: $ITERS$: Total de ciclos de comparación de permutaciones.
Salida: $\mathbf{A}$: Matriz $SCA(M; t, t\text{-}DEL + ADD)$ generada.

```
1: ———————— Variables y Estructuras de ejecución ————————————
2: 𝕆 ← Orden aleatorio de las permutaciones para posicionamiento de nuevas columnas.
3: ———————————— Procedimiento ——————————————————————
4: ∀ i ∈ [0, ..., ADD-1]{
5:    𝕆 ← ordenamiento aleatorio ({0, ..., M-1});
6:    pos := aleatorio mód (k + i);
7:    A_{𝕆_0} := genera_permutacion((k + i), pos, A_{𝕆_0});
8:    ∀ r ∈ [1, ..., M-1]{
9:       A_{𝕆_r} := mejor_permutacion(A_{𝕆_r});
10:   }
11:   si (ITERS > 1){
12:      𝕆 ← ordenamiento aleatorio ({0, ..., M-1});
13:      ∀ r ∈ [0, ..., M-1]{
14:         A_{𝕆_r} := mejor_permutacion((k + i), A, 𝕆, r);
15:      }
16:   }
17: }
18: devolver A;
```

El proceso de construcción de una permutación $\mathbb{X}$ que contenga la nueva variable n en la posición p, utiliza una permutación base $\mathbb{B}$ donde $(n \in \mathbb{B}) \vee (n \notin \mathbb{B})$. En el **Algoritmo 17** se define $\mathbb{X}$ como el conjunto de elementos de $\mathbb{B}$ que se encuentran antes de la posición p, concatenado con n y a su vez, concatenado con el conjunto de elementos de $\mathbb{B}$ restantes.

Algoritmo 17 genera_permutacion($n, p, \mathbb{B}$)

Entrada: n : Nueva variable por agregar.
Entrada: p : Posición donde se almacenará n.
Entrada: $\mathbb{B}$: Permutación base.
Salida: $\mathbb{X}$: Permutación con la variable n en la posición pos.
1: $\mathbb{X} := \{\forall\ i\ \mathbb{B}_i \neq n \mid i \in [0, ..., p\text{-}1]\} \oplus n \oplus \{\forall\ i\ \mathbb{B}_i \neq n \mid i \in [p, ..., |\mathbb{B}|\text{-}1]\}$;
2: **devolver** $\mathbb{X}$;

Ahora bien, para llevar a cabo el proceso de selección de la mejor permutación del conjunto de nuevas permutaciones con la variable n, primero se define la permutación base $\mathbb{B}$ como la mejor opción para evitar seleccionar una permutación peor al finalizar la comparación. La comparación se realiza en base al número de subsecuencias faltantes generado con cada permutación, simulando el intercambio de la permutación $\mathbb{B}$ por la permutación candidata $\mathbb{R}$. En el **Algoritmo 18** se presenta el desarrollo del algoritmo de selección de la mejor permutación para el renglón evaluado, donde se define $\mathbb{C}$ como un conjunto de posiciones aleatorias, $pos \in \mathbb{C}$ como la posición en donde se ubicará n en la nueva permutación $\mathbb{X}$, y $\mathbb{R}$ como permutación que genere máxima cobertura del conjunto de permutaciones evaluado.

Algoritmo 18 mejor_permutacion($n, \mathbf{A}, \mathbb{O}, r$)

Entrada: n : Nueva variable por agregar.
Entrada: $\mathbf{A}$: Matriz $SCA(M; t, k\text{-}DEL)$ inicial.
Entrada: $\mathbb{O}$: Ordenamiento aleatorio.
Entrada: r : Renglón base para la generación de la mejor permutación con variable n.
Salida: $\mathbb{R}$: Mejor permutación generada.

```
1: ———————— Variables y Estructuras de ejecución ————————
2: rowMoves := (k-DEL/4);
3: ℂ ← Conjunto de posiciones a comparar.
4: 𝔹 ← Permutación base para construir una nueva permutación.
5: 𝕏 ← Permutación generada.
6: ———————————— Procedimiento ————————————
7: ∀ i ∈ [0, ..., rowMoves-1] ℂ_i := (aleatorio mód (n + 1)) ∉ ℂ;
8: ℝ := 𝔹 := A_{𝕆_r};
9: ∀ pos ∈ ℂ{
10:    𝕏 := generar_permutacion(n, pos, 𝔹);
11:    si ((A|𝔹 = 𝕏).faltantes < (A|𝔹 = ℝ).faltantes) ℝ := 𝕏;
12: }
13: devolver ℝ;
```

4.3 saSCA: Algoritmo de recocido simulado para el ajuste de cobertura de SCAs

saSCA es un algoritmo de recocido simulado de la rama de metaheurísticas basado en el proceso de recocido de un metal. Este tipo de algoritmos simulan que un metal en estado líquido es enfriado lentamente hasta lograr una estructura cristalina altamente estable [29].

El algoritmo saSCA recibe un cuasi-SCA y puede producir un SCA o un cuasi-SCA con menos subsecuencias faltantes que el cuasi-SCA dado de entrada. El algoritmo se encarga de realizar perturbaciones a las permutaciones del SCA, generando SCAs que pueden presentar una mayor o menor cobertura que el SCA actual. Las perturbaciones se realizan con tres métodos distintos: *XCHG* genera la perturbación por medio del intercambio de variables; *ROT* genera la perturbación por medio de la rotación de variables; y *PMISS* genera la perturbación por medio de la adición de subsecuencias faltantes mediante el uso de grafos acíclicos dirigidos. Si el SCA generado presenta una

mayor cobertura o es aceptado por el criterio de aceptación de la distribución de Boltzmann, entonces se reemplaza el SCA actual por el SCA generado por la perturbación, por otro lado, si el SCA no es aceptado entonces se procede a generar otra perturbación del SCA actual. Es importante resaltar que, a diferencia del algoritmo de recocido simulado implementado por Daniel Osvaldo Ramírez Acuña [18], el recocido simulado desarrollado en este proyecto de tesis hace uso de dos métodos de perturbación adicionales ($XCHG$ y ROT), estos métodos permiten generar perturbaciones que ayudan a explorar el campo de soluciones del SCA actual saliendo de óptimos locales, ya que pueden generar soluciones con una menor cobertura que el SCA actual, pero que al descender la temperatura pueden llegar a converger en muy buenas soluciones; por otro lado, el método $PMISS$ siempre tratará de mejorar la solución actual por medio de la adición de subsecuencias faltantes.

4.3.1 Función principal del recocido simulado saSCA

Sea $\mathbf{A} = SCA(M; t, k)$ un cuasi-SCA, C la temperatura inicial, CF la temperatura final, α el factor de congelado, $congelado\_maximo$ el máximo número de variación en la temperatura sin actualizar el mejor cuasi-SCA global, $XCHG$ el porcentaje de perturbación por intercambio de variables, ROT el porcentaje de perturbación por rotación de variables, y $PMISS$ el porcentaje de perturbación por adición de subsecuencias faltantes por medio de GADs.

En el **Algoritmo 19** se presenta la función principal del recocido simulado. Sea $\mathbf{A}$ el cuasi-SCA actual, $\mathbf{A}'$ el cuasi-SCA generado en base a una perturbación, $\mathbf{SCA}$ el SCA global que contiene la máxima cobertura generada durante la ejecución, $aceptados\_maximo$ define el máximo número de $\mathbf{A}'$ aceptados para una temperatura e $intentos\_maximo$ el máximo número de perturbaciones a realizar por temperatura. Se construye un $\mathbf{A}'$ siempre y cuando, el número de subsecuencias faltantes de $\mathbf{A}$ sea mayor que 0, C mayor a CF y no se ha producido un estado de congelamiento. De la línea 10 a la línea 20 se evalúa si $\mathbf{A}'$ es aceptado (ya sea por mejora o por criterio de aceptación de la distribución de Boltzmann), de ser aceptado se reemplaza $\mathbf{A}$ por $\mathbf{A}'$ y si $\mathbf{A}'$ es mejor que el mejor global $\mathbf{SCA}$, entonces también lo sustituye y se exporta el archivo SCA que presenta mayor cobertura. Cada vez que se mejora el mejor SCA global, se incrementa el contador de mejoras

$mejorados$, si $mejorados$ es mayor que 0, entonces se reinicia el contador $congelado = 0$ y se incrementa la temperatura C, de lo contrario, si no hubo mejora entonces se incrementa el contador $congelado$ y se disminuye la temperatura C. El programa termina al generar un SCA con máxima cobertura (sin subsecuencias faltantes) o cuando se produce un estado de congelamiento (no hubo mejora al decrementar C, $congelado\_maximo$ veces).

Algoritmo 19 saSCA($\mathbf{A}$)

Entrada: $\mathbf{A}$: cuasi-SCA con subsecuencias faltantes.
Salida: $\mathbf{SCA}$: SCA con menos subsecuencias faltantes.

1: ———————— Variables y Estructuras de ejecución ————————
2: $\mathbf{A}' \leftarrow$ SCA perturbado.
3: ———————————— Procedimiento ————————————
4: $aceptados\_maximo := (Mk)^2$;
5: $intentos\_maximo := aceptados\_maximo * 10$;
6: **mientras** $((\mathbf{A}._{faltantes} > 0) \wedge (congelado < congelado\_maximo) \wedge (C \geq CF))\{$
7: $\quad aceptados := 0$;
8: $\quad mejorados := 0$;
9: $\quad intentos := 0$;
10: $\quad$ **mientras** $((intentos < intentos\_maximo) \wedge (aceptados < aceptados\_maximo) \wedge (\mathbf{SCA}._{faltantes} > 0))\{$
11: $\quad\quad intentos := intentos + 1$;
12: $\quad\quad \mathbf{A}' :=$ **perturbar**$(\mathbf{A}, \mathbf{SCA})$;
13: $\quad\quad r :=$ valor aleatorio uniforme $(0, 1)$;
14: $\quad\quad \delta := \mathbf{A}'._{faltantes}$-$\mathbf{A}._{faltantes}$;
15: $\quad\quad$ **si** $((r \leq$ **exp**$(-\frac{\delta}{C})) \vee (\mathbf{A}'._{faltantes} < \mathbf{A}._{faltantes}))\{$
16: $\quad\quad\quad aceptados := aceptados + 1$;
17: $\quad\quad\quad \mathbf{A} := \mathbf{A}'$;
18: $\quad\quad \}$
19: $\quad\quad$ **si** $(\mathbf{A}._{faltantes} < \mathbf{SCA}._{faltantes})\{$
20: $\quad\quad\quad mejorados := mejorados + 1$;
21: $\quad\quad\quad \mathbf{SCA} := \mathbf{A}$;
22: $\quad\quad\quad$ **exporta** $SCA(M; t, k) = \mathbf{SCA}$;
23: $\quad\quad \}$
24: $\quad \}$
25: $\quad$ **si** $(mejorados > 0)\{$
26: $\quad\quad congelado := 0$;
27: $\quad\quad C = \frac{C}{\alpha}$;
28: $\quad \}$**si no**$\{$
29: $\quad\quad congelado := congelado + 1$;
30: $\quad\quad C := (C * \alpha)$;
31: $\quad \}$
32: $\}$

El proceso de perturbación se basa en reemplazar una permutación del SCA actual $\mathbf{A}$ por medio de uno de los tres métodos de perturbación. La selección del método de perturbación a utilizar se realiza de forma aleatoria, se selecciona un valor aleatorio del rango [0,...,99], y se utiliza el método

cuyo porcentaje cubra el valor aleatorio. Por ejemplo: si $PMISS = 70$, $XCHG = 20$ y $ROT = 10$ entonces $PMISS$ cubre el rango [0,...,69], $XCHG$ el rango [70,...,89] y ROT el rango [90,...,99]. En el **algoritmo 20** se presenta el procedimiento de selección del método de perturbación.

Algoritmo 20 perturbar($\mathbf{A}, \mathbf{SCA}$)

Entrada: $\mathbf{A}$: SCA actual al que se realiza la perturbación.
Entrada: $\mathbf{SCA}$: Mejor SCA global generado durante la ejecución.
Salida: $\mathbf{A}'$: SCA perturbado.

1: ———————— Variables y Estructuras de ejecución ————————
2: $usar\_metodo :=$ **aleatorio** $\mod 100$;
3: ———————— Procedimiento ————————
4: **si** $(usar\_metodo < PMISS)$ $\mathbf{A}' := PMISS(A, SCA)$;
5: **si no. si** $(usar\_metodo < (PMISS + XCHG))$ $\mathbf{A}' := XCHG(A, SCA)$;
6: **si no. si** $(usar\_metodo < (PMISS + XCHG) + ROT)$ $\mathbf{A}' := ROT(A, SCA)$;
7: **devolver** $\mathbf{A}'$;

4.3.2 Método de perturbación XCHG

En el **Algoritmo 21** se presenta el desarrollo del método de perturbación por intercambio de dos variables. Sea $\mathbf{A}$ el SCA actual y $\mathbf{SCA}$ el mejor SCA global, seleccionar un renglón aleatorio $r \in \{M\}$. Generar $maxCH$ permutaciones por medio del intercambio de dos variables de la permutación base $\mathbf{A}_r$ y devolver la que genere menos subsecuencias faltantes. Para generar una permutación por intercambio, se seleccionan dos columnas ($col1$ y $col2$) de $\mathbf{A}_r$ de forma aleatoria, y se intercambian los valores $\mathbf{A}_{r,col1}$ y $\mathbf{A}_{r,col2}$ para generar la permutación $\mathbb{X}$. El método de perturbación termina cuando se mejora la mejor solución global o al terminar de comparar todas las $maxCH$ permutaciones. Se define $\mathbb{R} = \mathbb{X}$ como la mejor permutación en base al menor número de subsecuencias generado por reemplazar $\mathbf{A}_r$ por $\mathbb{X}$.

Algoritmo 21 XCHG($\mathbf{A}, \mathbf{SCA}$)

Entrada: $\mathbf{A}$: SCA actual al que se realiza la perturbación.
Entrada: $\mathbf{SCA}$: Mejor SCA global generado durante la ejecución.
Salida: $\mathbf{A}'$: Mejor SCA generado por perturbación.

```
1: ———————— Variables y Estructuras de ejecución ————————————
2: maxCH := 100;
3: 𝕏 ← Permutación perturbada actual.
4: ℝ ← Mejor perturbación generada.
5: ———————————— Procedimiento ————————————————————————
6: ℝ := ∅;
7: r := aleatorio mód M;
8: ∀ i ∈ [0, ..., maxCH-1]{
9:    col1 := aleatorio mód k;
10:   col2 := (aleatorio mód k) ≠ col1;
11:   𝕏 := A_r;
12:   temp := 𝕏_col1;
13:   𝕏_col1 := 𝕏_col2;
14:   𝕏_col2 := temp;
15:   si (ℝ = ∅) ℝ := 𝕏;
16:   si no. si ((A|A_r = 𝕏).faltantes < (A|A_r = ℝ).faltantes){
17:      ℝ := 𝕏;
18:   }
19:   si ((A|A_r = ℝ).faltantes < (SCA).faltantes) termina ciclo i;
20: }
21: A' := A;
22: A'_r := ℝ;
23: devolver A';
```

4.3.3 Método de perturbación ROT

En el **Algoritmo 22** se presenta el método de perturbación por rotación de elementos, similar al método $XCHG$. Sea $\mathbf{A}$ el SCA actual y $\mathbf{SCA}$ el mejor SCA global generado, seleccionar un renglón aleatorio $r \in \{M\}$, generar $maxRT$ permutaciones por rotación de variables de la permutación base $\mathbf{A}_r$ y devolver el SCA que genere menos subsecuencias faltantes al reemplazar $\mathbf{A}_r$ por la mejor perturbación generada. Para realizar la rotación, se seleccionan dos columnas aleatorias $col1 < col2$ y se hace un corrimiento de los elementos de las columnas $[col1, ..., col2\text{-}1]$ hacia la izquierda y se define el valor de la columna $col2$ por el valor de la columna $col1$.

Algoritmo 22 ROT($\mathbf{A}, \mathbf{SCA}$)

Entrada: $\mathbf{A}$: SCA actual al que se realiza la perturbación.
Entrada: $\mathbf{SCA}$: Mejor SCA global generado durante la ejecución.
Salida: $\mathbf{A}'$: Mejor SCA generado por perturbación.

1: ———————— Variables y Estructuras de ejecución ————————
2: $maxRT := 100$;
3: $\mathbb{X} \leftarrow$ Permutación perturbada actual.
4: $\mathbb{R} \leftarrow$ Mejor perturbación generada.
5: ———————— Procedimiento ————————
6: $\mathbb{R} := \emptyset$;
7: $r :=$ **aleatorio** mód M;
8: $\forall\ i\ \in\ [0, ..., maxRT\text{-}1]\{$
9: $\quad \mathbb{X} := \mathbf{A}_r$;
10: $\quad col1 :=$ **aleatorio** mód k;
11: $\quad col2 :=$ (**aleatorio** mód k) $\neq col1$;
12: $\quad$ **si** $(col1 > col2)\{$
13: $\quad\quad temp := col1$;
14: $\quad\quad col1 := col2$;
15: $\quad\quad col2 := temp$;
16: $\quad \}$
17: $\quad temp := \mathbb{X}_{col1}$;
18: $\quad \forall j\ \in\ [col1, ..., col2\text{-}1]\ \mathbb{X}_j := \mathbb{X}_{j+1}$;
19: $\quad \mathbb{X}_{col2} := temp$;
20: $\quad$ **si** $(\mathbb{R} = \emptyset)\ \mathbb{R} := \mathbb{X}$;
21: $\quad$ **si no. si**$((\mathbf{A}|\mathbf{A}_r = \mathbb{X}).{}_{faltantes} < (\mathbf{A}|\mathbf{A}_r = \mathbb{R}).{}_{faltantes})\{$
22: $\quad\quad \mathbb{R} := \mathbb{X}$;
23: $\quad \}$
24: $\quad$ **si** $((\mathbf{A}|\mathbf{A}_r = \mathbb{R}).{}_{faltantes} < (\mathbf{SCA}).{}_{faltantes}$ **termina ciclo** i;
25: $\}$
26: $\mathbf{A}' := \mathbf{A}$;
27: $\mathbf{A}'_r := \mathbb{R}$;
28: **devolver** $\mathbf{A}'$;

4.3.4 Método de perturbación PMISS

En el **Algoritmo 23** se presenta el método de perturbación por adición de subsecuencias faltantes mediante GADs. Primero se define un orden aleatorio $\mathbb{O}$ que indica el orden de perturbación de cada permutación del SCA actual para la comparación de cobertura. Se define $\mathbb{S}$ como la subsecuencia faltante que se desea cubrir en el SCA perturbado. Para cada permutación del SCA actual, se define

un grafo acíclico dirigido $\mathbf{G}$ inicializado con la subsecuencia $\mathbb{S}$, y se define $\mathbf{U}$ como el conjunto de subsecuencias únicas de la permutación base. Para cada subsecuencia única $\mathbb{Z}$, se intenta agregar la subsecuencia única al GAD $\mathbf{G}$. En la línea 12 se define $\mathbb{X}$ como una permutación utilizando un ordenamiento topológico del grafo $\mathbf{G}$. El resto del método es similar a los métodos anteriores, comparando la cobertura de reemplazar la permutación base por la permutación perturbada, y almacenando la mejor permutación de perturbación en $\mathbb{R}$.

Algoritmo 23 PMISS($\mathbf{A}, \mathbf{SCA}$)

Entrada: $\mathbf{A}$: SCA actual al que se realiza la perturbación.
Entrada: $\mathbf{SCA}$: Mejor SCA global generado durante la ejecución.
Salida: $\mathbf{A}'$: Mejor SCA generado por perturbación.

1: ———————— Variables y Estructuras de ejecución ————————
2: $\mathbb{O}$:= ordenamiento aleatorio del conjunto $[0, ..., M\text{-}1]$;
3: $\mathbb{S} \leftarrow$ Subsecuencia faltante a cubrir.
4: $\mathbb{X} \leftarrow$ Permutación candidata generada por la perturbación.
5: $\mathbb{R} \leftarrow$ Mejor permutación generada por PMISS.
6: $\mathbb{Z} \leftarrow$ Subsecuencia única que se pierde al eliminar el renglón base.
7: $\mathbf{P} \leftarrow$ Matriz de cobertura de la matriz $\mathbf{A}$.
8: $\mathbf{U} \leftarrow$ Conjunto de subsecuencias únicas del renglón a reemplazar.
9: $\mathbf{G} \leftarrow$ Grafo acíclico dirigido.
10: ———————— Procedimiento ————————————————
11: $\mathbb{R} := \emptyset$;
12: $\mathbb{S}$:= Subsecuencia faltante aleatoria.
13: $\forall\ r\ \in\ [0, ..., M\text{-}1]\{$
14: $\quad \mathbf{G} := \mathbb{S}$;
15: $\quad \mathbf{U} :=$ **subsecuencias_unicas**$(\mathbf{A}_{\mathbb{O}_r}, \mathbf{P})$;
16: $\quad \forall\ \mathbb{Z} \in\ \mathbf{U}\{$
17: $\quad\quad$ **si** $(\mathbf{G} \cup \mathbb{Z} = \text{GAD})\ \mathbf{G} := \mathbf{G} \cup \mathbb{Z}$;
18: $\quad \}$
19: $\quad \mathbb{X}$:= permutación por ordenamiento topológico de $\mathbf{G}$;
20: $\quad$ **si** $(\mathbb{R} = \emptyset)\ \mathbb{R} := \mathbb{X}$;
21: $\quad$ **si no. si** $((\mathbf{A}|\mathbf{A}_{\mathbb{O}_r} = \mathbb{X}).{}_{faltantes} < (\mathbf{A}|\mathbf{A}_{\mathbb{O}_r} = \mathbb{R}).{}_{faltantes})\{$
22: $\quad\quad \mathbb{R} := \mathbb{X}$;
23: $\quad \}$
24: $\quad$ **si** $((\mathbf{A}|\mathbf{A}_{\mathbb{O}_r} = \mathbb{R}).{}_{faltantes} < (\mathbf{SCA}).{}_{faltantes})$ **termina ciclo** i;
25: $\}$
26: $\mathbf{A}' := \mathbf{A}$;
27: $\mathbf{A}'_r := \mathbb{R}$;
28: **devolver** $\mathbf{A}'$;

En el **Algoritmo 24** se presenta el procedimiento de construcción de la lista $\mathbf{U}$ de subsecuencias únicas dada una permutación $\mathbb{R}$. Para cada subconjunto de columnas $\mathbb{C}$ se define la subsecuencia $\mathbb{S}$ cubierta por $\mathbb{R}$, y se evalúa su cobertura en la matriz $\mathbf{P}$, si el contador de cobertura de la subsecuencia es igual a 1, entonces al eliminar la permutación $\mathbb{R}$ se producirá una subsecuencia faltante. Si al eliminar una subsecuencia $\mathbb{S}$ se produce un incremento en las subsecuencias faltantes, entonces se agrega la subsecuencia $\mathbb{S}$ a la lista $\mathbf{U}$ de subsecuencias únicas.

Algoritmo 24 subsecuenciasUnicas($\mathbb{R}, \mathbf{P}$)

Entrada: $\mathbb{R}$: Permutación base que será reemplazada.
Entrada: $\mathbf{P}$: Matriz de cobertura del SCA base.
Salida: $\mathbf{U}$: Conjunto de subsecuencias únicas de $\mathbb{R}$.

1: ———————— Variables y Estructuras de ejecución ————————
2: $\mathbb{C} \leftarrow$ Subconjunto de columnas de $\mathbb{R}$ que definen una subsecuencia cubierta.
3: $\mathbb{V} \leftarrow$ Conjunto de variables que conforman la subsecuencia cubierta.
4: $\mathbb{S} \leftarrow$ Subsecuencia cubierta.
5: ———————— Procedimiento ————————————
6: $\mathbf{U} := \{\emptyset\}$;
7: $\forall\, \mathbb{C} \in \{\binom{k}{t}\}\{$
8: $\quad \forall\, i \ \in \ [0, ..., t\text{-}1]\ \mathbb{S}_i := \mathbb{R}_{\mathbb{C}_i}$;
9: $\quad \mathbb{V} :=$ **ordenar**$(\mathbb{S})$;
10: $\quad evn :=$ **DirectGTP**$(\mathbb{V}, t)$;
11: $\quad sec :=$ **subsecuencia_a_entero**$(\mathbb{R}, \mathbb{C})$;
12: $\quad$ **si** $(\mathbf{P}_{evn,sec} = 1)\{$
13: $\quad\quad \mathbf{U} := \mathbf{U} \oplus \mathbb{S}$;
14: $\quad\}$
15: $\}$
16: **devolver** $\mathbf{U}$;

4.4 testSCA: Algoritmo de evaluación de cobertura de SCAs

testSCA es un algoritmo desarrollado con el objetivo de validar y evaluar la cobertura un SCA, sea $\mathbf{A}$ el SCA a evaluar y $\mathbf{A}^{-1}$ la representación inversa del SCA a evaluar. Para cada conjunto $\mathbb{V}$ de t variables, se evalúan las subsecuencias cubiertas por cada una de las M permutaciones del SCA.

El vector $\mathbb{P}$ de tamaño $t!$, realiza el conteo de subsecuencias cubiertas para cada conjunto $\mathbb{V}$. Al finalizar de evaluar un conjunto $\mathbb{V}$, se realiza un repaso del vector de cobertura $\mathbb{P}$ para contabilizar el número de subsecuencias con cobertura 0, se realiza la exportación de las subsecuencias faltantes y se procede a evaluar el siguiente conjunto de t variables. A continuación se presenta un ejemplo de evaluación de cobertura con *testSCA*, en la Tabla 4.4 se realiza la evaluación de $SCA(8;3,10)$ para el conjunto de variables $\mathbb{V} = \{3,5,7\}$, $\mathbf{A}_{\mathbb{V}}^{-1}$ indica las columnas del SCA donde se encuentran las variables del conjunto $\mathbb{V}$, $\mathbb{C}$ indica el conjunto de columnas ordenado del SCA donde se encuentra la subsecuencia a evaluar, $\mathbf{A}_{\mathbb{C}}$ indica la subsecuencia cubierta, $\mathbb{X}$ indica la subsecuencia de $\mathbb{V}$ evaluada y α indica el valor de $\mathbb{X}$ convertido en valor entero.

Tabla 4.4: Ejemplo de evaluación de cobertura de las permutaciones de $SCA(8;3,10)$ para el conjunto de variables $\mathbb{V} = \{3,5,7\}$ en *testSCA*. $\mathbf{A}$ Representa la permutación del SCA, $\mathbf{A}^{-1}$ representa la notación inversa de la permutación $\mathbf{A}$, $\mathbb{V}$ indica el conjunto de variables a evaluar, $\mathbf{A}_{\mathbb{V}}^{-1}$ indica las columnas del SCA donde se encuentran las variables del conjunto $\mathbb{V}$, $\mathbb{C}$ indica el conjunto de columnas ordenado del SCA donde se encuentra la subsecuencia a evaluar, $\mathbf{A}_{\mathbb{C}}$ indica la subsecuencia cubierta, $\mathbb{X}$ indica la subsecuencia de $\mathbb{V}$ evaluada y α indica el valor de $\mathbb{X}$ convertido en valor entero.

$\mathbf{A}$	$\mathbf{A}^{-1}$	$\mathbb{V}$	$\mathbf{A}_{\mathbb{V}}^{-1}$	$\mathbb{C}$	$\mathbf{A}_{\mathbb{C}}$	$\mathbb{X}$	α
0 2 3 1 4 5 6 7	0 3 1 2 4 5 6 7	3 5 7	2 5 7	2 5 7	3 5 7	0 1 2	0
2 0 7 6 5 4 1 3	1 6 0 7 5 4 3 2	3 5 7	7 4 2	2 4 7	7 5 3	2 1 0	4
7 3 2 5 1 6 4 0	7 4 2 1 6 3 5 0	3 5 7	1 3 0	0 1 3	7 3 5	2 0 1	3
4 1 6 0 7 3 2 5	3 1 6 5 0 7 2 4	3 5 7	5 7 4	4 5 7	7 3 5	2 0 1	3
5 6 1 7 0 2 3 4	4 2 5 6 7 0 1 3	3 5 7	6 0 3	0 3 6	5 7 3	1 2 0	5
1 5 4 3 2 0 7 6	5 0 4 3 2 1 7 6	3 5 7	3 1 6	1 3 6	5 3 7	1 0 2	2
3 7 0 4 6 1 5 2	2 5 7 0 3 6 4 1	3 5 7	0 6 1	0 1 6	3 7 5	0 2 1	1
6 4 5 2 3 7 0 1	6 7 3 4 1 2 0 5	3 5 7	4 2 5	2 4 5	5 3 7	1 0 2	2

Al incrementar el contador de cada una de las subsecuencias cubiertas, obtenemos un vector de cobertura $\mathbb{P} = \{1,1,2,2,1,1\}$ con el cual, se verifica que $\mathbf{A}$ cubre las $t!$ subsecuencias del conjunto de variables $\mathbb{V} = \{3,5,7\}$ por lo menos una vez. En el **Algoritmo 25** se presenta el desarrollo de *testSCA* para la evaluación de cobertura de un SCA.

Se recibe el SCA/cuasi-SCA $\mathbf{A}$ y se devuelve el total de subsecuencias faltantes (que no fueron cubiertas ni una sola vez). Se define la matriz inversa del SCA/cuasi-SCA en la matriz $\mathbf{A}^{-1}$; se define $faltantes$ como 0; para todo conjunto $\mathbb{V}$ de t variables, se inicializa el vector de cobertura $\mathbb{P}$ con sus

Algoritmo 25 testSCA($\mathbf{A}$)

Entrada: $\mathbf{A}$: SCA a evaluar.
Salida: $faltantes$: Total de subsecuencias faltantes.

```
1: ———————— Variables y Estructuras de ejecución ————————————————
2: A^-1 ← Notación inversa de A.
3: 𝕍 ← Conjunto de t variables a evaluar subsecuencias cubiertas.
4: ℙ ← Vector de conteo de subsecuencias cubiertas.
5: ℂ ← Conjunto de columnas de A donde se encuentra la subsecuencia a contar.
6: ———————————— Procedimiento ————————————————————————————
7: faltantes := 0;
8: i ∈ [0, ..., t!-1];
9: j ∈ [0, ..., t-1];
10: ∀ 𝕍 ∈ {C(k,t)}{
11:    ∀ i ℙ_i := 0;
12:    ∀ r ∈ [0, ..., M-1]{
13:       ∀ j ℂ_j := A^{-1}_{𝕍_j};
14:       ℂ ← ordenar(ℂ);
15:       α ← subsecuencia_a_entero(A_r, ℂ);
16:       ℙ_α := ℙ_α + 1;
17:    }
18:    faltantes := faltantes + ||(ℙ_i = 0)||;
19:    si (||(ℙ_i = 0)|| > 0){
20:       exporta(DirectGTP(𝕍, t) ⊕ ||(ℙ_i = 0)|| ⊕ {∀ i | (ℙ_i = 0)});
21:    }
22: }
23: devolver faltantes;
```

contadores en 0, y para cada permutación r se evalúa la subsecuencia cubierta, definiendo $\mathbb{C}$ como el conjunto de columnas que define la subsecuencia a evaluar, se realiza el ordenamiento de los valores de $\mathbb{C}$ de menor a mayor; se define α como el número entero positivo que representa la subsecuencia cubierta; se incrementa el contador α en el vector de cobertura $\mathbb{P}$; se suman las subsecuencias faltantes encontradas al evaluar todas las permutaciones con el conjunto de columnas $\mathbb{V}$; se exporta las subsecuencias faltantes en forma de renglón, donde los primeros t valores corresponden al conjunto de variables evaluado, el valor $t+1$ indica el número de subsecuencias faltantes con dicho conjunto de variables, y el resto de valores indican las subsecuencias faltantes con un valor entero positivo.

4.5 Resumen

En este capítulo se presentaron los algoritmos de manipulación de SCAs que permiten el agregado/borrado de renglones/columnas. De manera adicional se presentó el algoritmo de recocido simulado usado para reducir el número de subsecuencias faltantes en un cuasi-SCA. Al final, se presentó el algoritmo de verificación de cobertura de SCAs/cuasi-SCAs. Estos cuatro algoritmos fueron usados en el proceso de construcción de los SCAs reportados en esta tesis. La orquestación de cómo se usan estos algoritmos es descrita en la metodología presentada en el siguiente capítulo, en el siguiente capítulo también son presentados los resultados computacionales obtenidos al aplicar la metodología presentada.

5

Metodología de construcción de SCAs y SCAs construidos

En este capítulo se presenta la metodología para orquestar la operación de los algoritmos de manipulación y el algoritmo de recocido simulado presentados en el capítulo previo. Finalmente, se presentan los SCAs de fuerza 3 construidos en esta tesis utilizando desde 4 hasta 200 columnas.

5.1 Metodología para la construcción de SCAs

La metodología de construcción de SCAs que se llevó a cabo en este proyecto de tesis, se basa en la máxima adición de variables con el menor número de permutaciones mediante los algoritmos de manipulación de SCAs y un algoritmo de recocido simulado. Los elementos que componen la metodología de construcción se presentan a continuación:

- **Sequence Covering Array inicial:** El SCA inicial se construye utilizando el algoritmo de Levenshtein [37] para construir un $SCAN(t, t+1) = t!$ para $t = 3$.
- **Repositorio de los mejores SCAs construidos:** El repositorio se encarga de almacenar los

SCAs (que pueden variar en número de renglones o número de columnas) generados durante la ejecución de la metodología. El repositorio se encarga de actualizar los SCAs para cada determinado número de renglones y columnas.

- **Algoritmos de manipulación de SCAs:** Se conforma por los algoritmos *moveRowSCA* y *moveColSCA* para el incremento de renglones y columnas, y *saSCA* para el ajuste de cobertura.

La metodología implementada en esta tesis se presenta a continuación por medio de un diagrama de flujo en la Figura 5.1. Se inicializa la metodología con un repositorio vacío que puede contener como máximo, un $SCA(M_{max}; 3, k_{max})$. Se inicializa un SCA óptimo de fuerza 3 por medio del algoritmo de Levenshtein [37], $SCAN(t, t+1) = SCA(6; 3, 4)$. Se almacena el SCA generado en el repositorio, posteriormente se agrega una nueva columna por medio de *moveColSCA* para generar un $SCA(6; 3, 5)$, si se presentan subsecuencias faltantes, se utiliza *saSCA* para realizar el ajuste de cobertura. Si después del ajuste se presentan subsecuencias faltantes, se utiliza *moveRowSCA* para agregar una permutación más al cuasi-SCA, si después de agregar la nueva permutación aun se presentan subsecuencias faltantes, entonces se utiliza nuevamente *saSCA* para realizar un ajuste de cobertura. El ciclo de agregado de permutación y ajuste se realiza hasta que se genere un SCA con máxima cobertura. Cada vez que se genere un SCA con $faltantes = 0$, se agrega al repositorio de soluciones y se procede a agregar una nueva columna.

Gracias a la metodología de construcción implementada, se pueden construir SCAs con distinto número de columnas y con el mismo o mínimo número de permutaciones, basado en un SCA óptimo o de mínima cardinalidad.

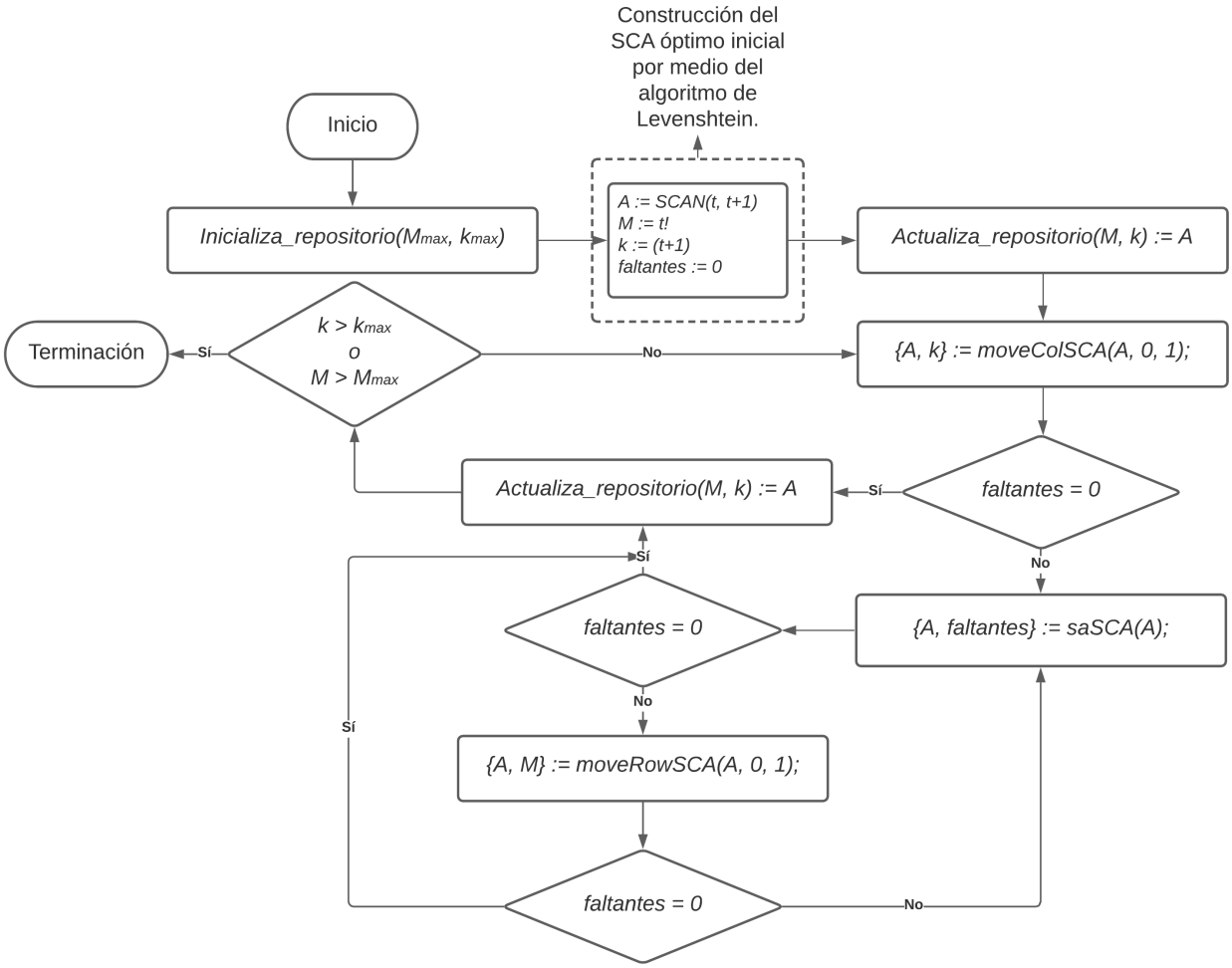


Figura 5.1: Diagrama de flujo sobre la metodología de construcción de SCAs por enfoque CAEX, utilizando como base un SCA óptimo.

5.2 Parámetros utilizados en los algoritmos para la metodología de construcción

5.2.1 Parámetros utilizados moveRowSCA

Para el incremento de permutaciones dado un SCA, se utilizaron los parámetros $DEL = 0$, $ADD = 1$ e $ITERS = \{1, ..., 5\}$, con los que se agrega una permutación que cubra el mayor

número de subsecuencias faltantes.

5.2.2 Parámetros utilizados en moveColSCA

El incremento de columnas en un SCA dado se realiza con los parámetros $DEL = 0$, $ADD = 1$, $ITERS = \{1, ..., 5\}$ con lo que se agrega una nueva variable al SCA, ubicando la nueva variable de manera que se cubra el mayor número de subsecuencias faltantes.

5.2.3 Parámetros utilizados en saSCA

Basados en trabajos relacionados como [18] que utilizan el recocido simulado, se han definido los siguientes parámetros de ejecución:

- **C** = 4000
- **CF** = 4×10^{-20}
- α = 0.9
- **maxFrozen** = 10
- **maxMoves** = $(Mk)^2$
- **maxAttMoves** = $10(Mk)^2$
- **XCHG:** 20 %
- **ROT:** 10 %
- **PMISS:** 70 %

5.3 Resultados obtenidos en la construcción de SCAs

El desarrollo de los algoritmos utilizados en la metodología de construcción de SCAs, se lleva a cabo por medio del lenguaje de programación C por su portabilidad, rendimiento óptimo, asignación dinámica de memoria, entre otras ventajas. La experimentación se llevó a cabo gracias a los servidores proporcionados por el Cinvestav, lo que nos permitió ejecutar la experimentación en menor tiempo en comparación a un equipo común.

Para efectos de la metodología de construcción mediante incremento de columnas y renglones de un SCA óptimo, los algoritmos de modificación *moveRowSCA* y *moveColSCA* se ejecutaron con los parámetros $DEL = 0$ y $ADD = 1$, por otro lado, el algoritmo de recocido simulado se implementa con un mayor porcentaje de perturbación por parte del método $PMISS$, el cual, demostró un mayor rendimiento al momento de ajustar la cobertura de un cuasi-SCA, por su forma de cubrir el mayor número de subsecuencias faltantes.

La metodología implementada en este proyecto de tesis nos permitió construir SCAs con un mínimo número de permutaciones para SCAs de fuerza $t = 3$ y $k = \{4, ..., 200\}$ con un rango de $M = \{6, ..., 24\}$ permutaciones, con lo que se reportan 118 nuevas cotas de SCAs para fuerza 3. En la literatura se conoce como *cota mínima* o *mejor cota* al mínimo número de permutaciones reportado en la literatura para construir un SCA en particular.

Actualmente las mejores cotas de SCAs de fuerza $t = 3$, se encuentran reportadas en [18] por Ramirez-Acuña y Torres-Jimenez, por lo que, las nuevas cotas construidas en este proyecto están definidas en comparación a los resultados reportados en [18]. En la Tabla 5.1 se presentan los SCAs construidos en este proyecto de tesis donde la primera columna define un número de permutaciones M, la segunda columna indica el máximo valor de k con el que se logró construir un SCA dadas M permutaciones, y en la tercera columna se reportan el total de nuevas cotas generadas para el determinado valor de M.

Tabla 5.1: Nuevas cotas construidas con la metodología definida en el proyecto de tesis para $k = \{4, ..., 200\}$ variables y fuerza $t = 3$. Se compararon los resultados obtenidos en este trabajo de tesis contra los mejores SCAs reportados al día de hoy por [18] para fuerza $t = 3$, la primera columna indica el número de permutaciones, la segunda columna el máximo valor de k con el que se logró construir un SCA, y la tercera columna reporta las nuevas cotas generadas para cada número de permutaciones.

M	**máximo valor de k**	**Nuevas cotas**
10	19	2
11	21	2
12	36	9
13	42	6
14	49	7
15	60	11
16	74	14
17	86	12
18	95	9
19	102	7
20	119	17
21	133	14
22	154	21
23	172	18
24	200	28
Total de nuevas cotas		118

A continuación se presenta la comparación de las mejores cotas construidas con distintos algoritmos reportados en la literatura de los SCAs para fuerza $t = 3$. En la Tabla 5.2 se presentan las cotas en función de SCAKs, donde se compara el máximo valor de k generado por cada algoritmo con un determinado número de permutaciones, en la columna LEV se presentan los resultados reportados por Levenshtein [37]; la columna TA presenta los resultados reportados por Tauri [51]; la columna U presenta los resultados de la selección de permutaciones aleatorias y Ur los resultados con función de inversión reportados por Colbourn *et al.* [10]; en la columna K se presentan los resultados del algoritmo avaro reportado por Richard Khun [31]; en las columnas ER, BTI y BR se presentan los resultados reportados en [19, 1, 4] donde se utiliza ASP para la construcción de SCAs; en la columna D se presentan los resultados de un algoritmo avaro, y en Dr se presentan los resultados de la función de inversión reportados en [10]; en la columna MC se presentan los resultados de los algoritmos de post-optimización reportados en [43]; en la columna TR se presentan las mejores cotas reportadas por Daniel Ramírez y Torres-Jimenez [18]; finalmente, en la columna TP se presentan los mejores resultados generados en este proyecto de tesis. La primera columna indica el número de permutaciones para construir un SCA y el resto de las columnas representa los resultados de cada

algoritmo de construcción en la literatura. En negrita se resaltan los nuevos SCAKs generados por la metodología de construcción implementada en esta tesis.

Tabla 5.2: Comparación de las mejores cotas reportadas por distintos algoritmos de construcción de SCAs en la literatura para fuerza $t = 3$ y $k = \{4, ..., 200\}$, donde la columna (TP) presenta los resultados de la metodología de este proyecto de tesis. La primera columna indica el número de permutaciones para construir un SCA y el resto de las columnas representa los resultados de cada algoritmo de construcción en la literatura: LEV[37], TA[51], U y Ur [10], K[31], ER[19], Dr y D[10], BTI[1], BR[4], MC[43], y TR[18]. Los resultados se presentan como el mayor número de variables con las que se puede construir un SCA con el número de permutaciones indicado. En negrita se resaltan los nuevos SCAKs generados en la tesis.

Permutaciones	LEV	TA	U	Ur	K	ER	Dr	D	BTI	BR	MC	TR	TP
6	4							4			4	4	4
7									5	5	5	5	5
8		6			5		4	6	8	8	7	8	8
9								7	10	10	9	10	10
10		10			6		6	8	15	16	10	17	**19**
11						10		9	17	17		19	**21**
12		20	4	4	8		9	10	21	22	15	27	**36**
13								13	22	24	20	29	**42**
14		30			11		12	15	28	29	25	39	**49**
15								17	30	30	30	42	**60**
16		70		5	14		16	20			40	53	**74**
17			5					24	40	40	50	61	**86**
18		90		6	16		21	28		50	60	71	**95**
19						20		30	50		70	79	**102**
20			6		18		28			60	80	90	**119**
21								40	60		90	107	**133**
22				7	22		30		70	70		120	**154**
23			7			30		50		80		135	**172**
24				8	26		40	60	80			155	**200**
25								70				173	
26			8	9	30		60	80				190	
27						40		90				200	
28			9	10			70						
29													
30			10	12			90						
31						50							
32			11	13	40								
33			12										
34				15	50	60							
35			13										
36			14	17		70							
37			15										
38				20	60	80							
39			16										
40			17	23	70								
41			18										
42			20	26	80								
43			21										
44			22	29	90								
45			23										
46			25	30									
47			26										
48			28										
49			30										
50				40									
52				50									
54			40										
56				60									
58			50	70									
60				80									
61			60										
62				90									
64			70										
66			80										
68			90										

En la Figura 5.2 se presenta una gráfica de la comparativa presentada en la Tabla 5.2. Se presentan los resultados para $k = \{4, ..., 200\}$ y $t = 3$, donde el eje horizontal muestra el número de permutaciones y el eje vertical el número de variables alcanzado. Cada punto representa el máximo número de variables con las que se logra construir un SCA dado un determinado número de permutaciones, las línea que conectan los puntos indican que, entre cada conjunto de soluciones existe una permutación de diferencia.

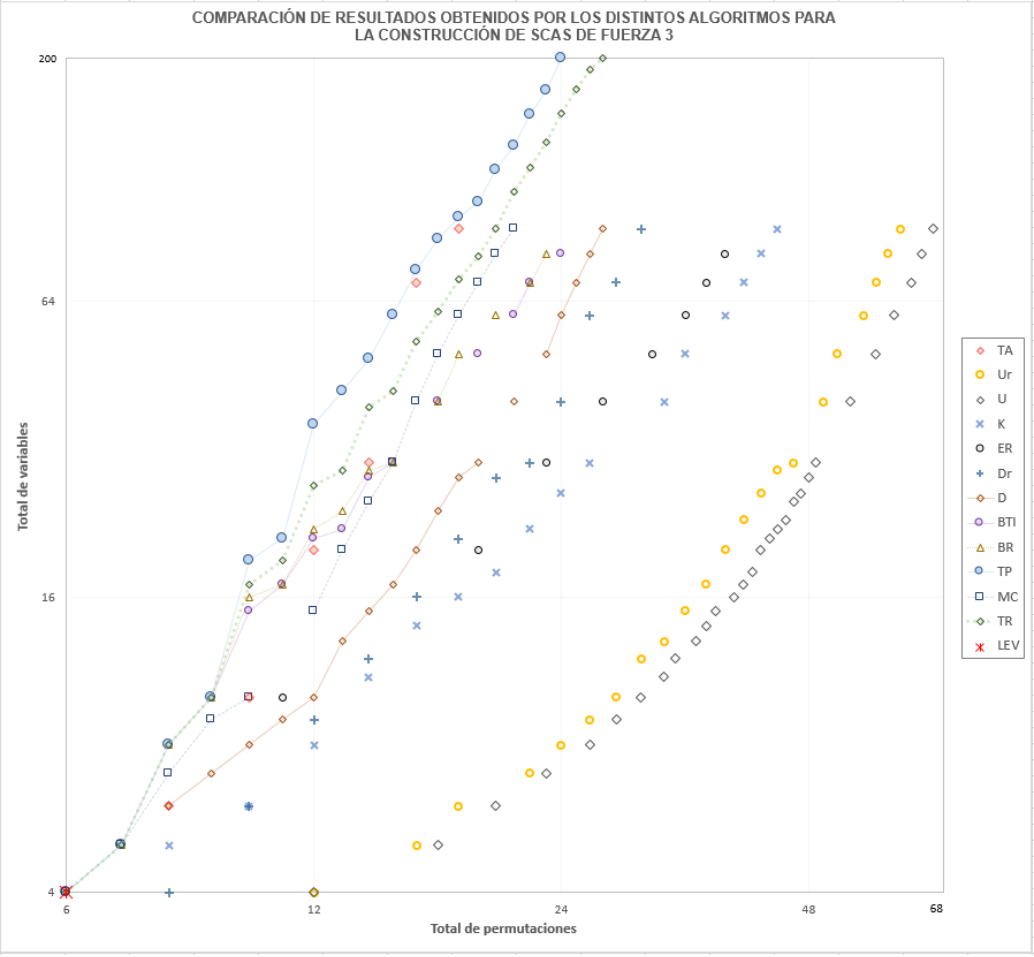


Figura 5.2: Gráfica comparativa de las mejores cotas reportadas por los distintos algoritmos de construcción de SCAs de fuerza $t = 3$ y $k = \{4, ..., 200\}$, donde adicionalmente se presentan las mejores cotas reportadas en este proyecto de tesis. El eje x indica el total de permutaciones, por otro lado, el eje y indica el total de variables; cada punto indica el máximo número de variables con las que se puede construir un SCA con el número de permutaciones dado. Cada línea entre puntos indica que entre cada conjunto de soluciones existe una permutación de diferencia.

Se puede observar que en el algoritmo TR (el cual, presenta el mayor número cotas mínimas reportadas en la literatura) es igualado en los casos pequeños, pero superado por el algoritmo TP conforme incrementa el número de permutaciones, presentando una reducción de 3 permutaciones para el caso de $k = 200$. Gracias a la representación gráfica de los resultados obtenidos, se puede observar la calidad de las soluciones generadas con la metodología de construcción implementada, donde las cotas generadas destacan frente al resto de los algoritmos, ya que, hemos logrado alcanzar el máximo número de variables con el menor número de permutaciones reportado actualmente para fuerza 3.

Finalmente, en la Tabla 5.3 se presentan los SCANs generados por los algoritmos comparados durante la experimentación para fuerza 3 y $k = \{4, ..., 200\}$, donde se compara el mínimo número de permutaciones generado para cada número de variables. En la columna LEV se presentan los resultados reportados por Levenshtein [37]; la columna TA presenta los resultados reportados por Tauri [51]; la columna U presenta los resultados de la selección de permutaciones aleatorias y Ur los resultados con función de inversión reportados por Colbourn *et al.* [10]; en la columna K se presentan los resultados del algoritmo avaro reportado por Richard Khun [31]; en las columnas ER, BTI y BR se presentan los resultados reportados en [19, 1, 4] donde se utiliza ASP para la construcción de SCAs; en la columna D se presentan los resultados de un algoritmo avaro, y en Dr se presentan los resultados de la función de inversión reportados en [10]; en la columna MC se presentan los resultados de los algoritmos de post-optimización reportados en [43]; en la columna TR se presentan las mejores cotas reportadas por Daniel Ramírez y Torres-Jimenez [18]; y en la columna TP se presentan los mejores resultados generados en este proyecto de tesis. La primera columna indica el número de variables, el resto de las columnas indica el mínimo número de permutaciones generado. En las últimas dos columnas se presentan entre paréntesis el máximo número de variables generado por TR y TP, y en negrita se resaltan las nuevas cotas generadas por la metodología de construcción implementada en la tesis.

Tabla 5.3: Comparación de los SCANs generados por los algoritmos: LEV[37], TA[51], U y Ur [10], K[31], ER[19], Dr y D[10], BTI[1], BR[4], MC[43], TR[18] y TP (resultados generados en la tesis). Entre paréntesis se indica el máximo número de variables generado y en negrita se resaltan los nuevos SCANs generados por la metodología de construcción implementada en la tesis.

k	LEV	TA	U	Ur	K	ER	Dr	D	BTI	BR	MC	TR	TP
4	6	8	12	12			8	6			6	6	6
5		8	17	16	8		10	8	7	7	7	7	7
6		8	20	18	10		10	8	8	8	8	8(8)	8(8)
7		10	23	22	12		12	9	8	8	8	8	8
8		10	26	24	12		12	10	8	8	9	8	8
9		10	28	26	14		12	11	9	9	9	9(10)	9(10)
10		10	30	28	14	11	14	12	9	9	10	9	9
11		12	32	30	14		14	12	10	10		10(17)	10(19)
12		12	33	30	16		14	13	10	10		10	10
13		12	35	32	16		16	13	10	10		10	10
14		12	36	34	16		16	14	10	10		10	10
15		12	37	34	18		16	14	10	10	12	10	10
16		12	39	36	18		16	15	11	10		10	10
17		12	40	36	20		18	15	11	11		10	10
18		12	41	38	20		18	16	12	12		11(19)	**10**
19		12	42	38	22		18	16	12	12		11	**10**
20		12	42	38	22	19	18	16	12	12	13	12(27)	**11(21)**
21		14	43	40	22		18	17	12	12		12	**11**
22		14	44	40	22		20	17	13	12		12	12**(36)**
23		14	45	40	24		20	17	14	13		12	12
24		14	46	42	24		20	17	14	13		12	12
25		14	46	42	24		20	18	14	14	14	12	12
26		14	47	42	24		20	18	14	14		12	12
27		14	48	44	26		20	18	14	14		12	12
28		14	48	44	26		20	18	14	14		13(29)	**12**
29		14	49	44	26		22	19	15	14		13	**12**
30		14	49	46	26	23	22	19	15	15	15	14(39)	**12**
40		16	54	50	32	27	24	21	17	17	16	15(42)	**13(42)**
49		16	54	50	32	27	24	21	17	17	16	16(53)	**14(49)**
50		16	58	52	34	31	26	23	19	18	17	16	**15(60)**
60		16	61	56	38	34	26	24	21	20	18	17(61)	**15**
70		16	64	58	40	36	28	25	22	22	19	18(71)	**16(74)**
79		16	64	58	40	36	28	25	22	22	19	19(79)	**17(86)**
80		18	66	60	42	38	30	26	24	23	20	20(90)	**17**
90		18	68	62	44		30	27			21	20	**18(95)**
100												21(107)	**19(102)**
110												22(120)	**20(119)**
120												22	**21(133)**
121												23(135)	**21**
130												23	**21**
140												24(155)	**22(154)**
150												24	**22**
160												25(173)	**23(172)**
170												25	**23**
180												26(190)	**24(200)**
190												26	**24**
200												27(200)	**24**

5.4 Resumen

En este capítulo se presentó la metodología para orquestar la operación de los algoritmos de manipulación y el algoritmo de recocido simulado para la construcción de SCAs de fuerza 3.

Los resultados del uso de la metodología son 18 SCAs de fuerza 3 cubriendo columnas desde 4 hasta 200. Gracias a estos SCAs, se definieron 118 nuevas cotas superiores en el valor del SCAN. La parte más relevante de los resultados es que todos los resultados previamente reportados fueron

igualados o mejorados. Es muy relevante que se logró construir un $SCA(24; 3, 200)$ y el mejor reportado previamente era $SCA(27; 3, 200)$. En el siguiente capítulo se presentan las conclusiones soportadas por los resultados obtenidos y se ilustran algunos posibles trabajos futuros.

6

Conclusiones y trabajos a futuro

En este capítulo se presentan las conclusiones derivadas de los resultados obtenidos en base a la experimentación del capítulo previo. Para finalizar, se hace mención de algunos posibles trabajos futuros.

6.1 Conclusiones

La construcción de conjuntos de pruebas que evalúen la funcionalidad de componentes de software o hardware usando el menor número de pruebas es un tema de gran relevancia a nivel mundial. La principal aportación de esta tesis radica en la construcción de mejores Sequence Covering Arrays, los cuales son muy útiles para validar el funcionamiento de componentes de software o hardware que dependen del orden en el que son introducidos los valores de las variables de entrada.

A través del uso de la metodología de construcción de SCAs planteada en esta tesis, se logró coordinar la operación de dos algoritmos de manipulación y un algoritmo metaheurístico. Los dos algoritmos de manipulación de SCAs son: *moveRowSCA* y *moveColSCA* que permiten agregar/borrar renglones/columnas de un SCA/cuasi-SCA. El algoritmo metaheurístico implementado está basado

en el paradigma de recocido simulado.

Los resultados obtenidos con la experimentación, demostraron que el uso de la metodología propuesta produce resultados altamente competitivos. Las principales conclusiones derivadas de la experimentación son:

- La metodología de construcción implementada en la tesis logra igualar y en algunos casos mejorar todos los casos reportados en la literatura para fuerza 3 y un máximo de 200 variables.
- Se reportan 118 nuevas cotas superiores de SCANs de fuerza 3 cubriendo hasta 200 variables.
- La metodología de construcción implementada en la tesis logra construir los mejores casos de fuerza 3 no reproducidos por otro algoritmo de construcción mencionado a partir de $k = 18$ variables hasta $k = 200$ variables.
- La metodología de construcción implementada en la tesis logra mejorar las cotas de SCAKs reportadas en la literatura para $M = \{10, ..., 24\}$.
- Se reporta la construcción de un SCA de fuerza 3 y 200 variables usando solo 24 permutaciones, logrando reducir tres permutaciones al mejor caso reportado en la literatura.

6.2 Trabajos a futuro

Como futuros proyectos para este tema de investigación, se encuentran:

- Construcción de SCAs de fuerza 3 con $k \geq 200$ utilizando la metodología implementada.
- Construcción de SCAs de fuerza $t = \{4, 5, 6\}$ que logren mejorar los casos reportados para dichas fuerzas en la literatura (dichos casos de prueba no se realizaron debido a que demandan una gran cantidad de tiempo de cómputo).
- Rediseño de la metodología de construcción para que se aprovechen los casos de eliminación de renglones/columnas en los algoritmos de manipulación con el objetivo de construir SCAs con máxima cobertura.

- Explotación de los elementos redundantes en un SCA para mejorar el desempeño del algoritmo de recocido simulado y los algoritmos de manipulación de SCAs.
- Paralelización de los algoritmos de manipulación de SCAs propuestos en esta tesis.
- Paralelización del algoritmo de recocido simulado para construir SCAs.

A Publicación relacionada con esta tesis en la que participé como coautor

En esta sección se presenta el abstract correspondiente al artículo *Three Representations for Set Partitions*[54] en el cual participé como coautor:

" The Set Partitioning Problem (SPP) aims to obtain non-empty disjoint subsets of objects such that their union equals the whole set of objects, and the partition meets some prespecified criteria. The ubiquity of SPP is impressive, given that it has a lot of theoretical and practical motivations. In the theoretical side, the study of the SPP is closely related to Bell numbers, Stirling numbers of the second kind, integer partitions, Eulerian numbers, Restricted Growth Strings (RGS), factoradic number system, power calculations, etc. In the practical side, SPP is intimately related to classification problems, clustering problems, reduction of dimensionality problems, and so on. In this work, three representations for instances of SPP are presented, these representations use: Restricted Growth Strings (RGS), factoradic number system, and a number system with a fixed base. Two cases for these representations will be presented: where the number of subsets is unbounded (i.e. the number of subsets can be the number of objects); and where the number of subsets is less than the number of objects. Bidirectional mappings between these three representations will be introduced, also the mapping among these three representations and the power of a base is defined. Given, that these three representations can be used to solve instances of SPP using exact, greedy, and metaheuristic algorithms, that require to do small changes to one possible solution and/or recombination of two possible solutions, definitions of mutation and recombination operators for the three representations will be shown. In order to motivate the use of the three representations for the solution of particular instances of SPP, it was decided to present their application to solve an instance of a set partition of integers problem (SPIP) using a simple genetic algorithm. "

Bibliografía

[1] Mutsunori Banbara, Naoyuki Tamura, and Katsumi Inoue. Generating Event-Sequence Test Cases by Answer Set Programming with the Incidence Matrix. In Agostino Dovier and Vítor Santos Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*, volume 17 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 86–97, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[2] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003.

[3] Gregor V. Bochmann and Alexandre Petrenko. Protocol testing: Review of methods and relevance for software testing. In *Proceedings of the 1994 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '94, page 109–124, New York, NY, USA, 1994. Association for Computing Machinery.

[4] Martin Brain, Esra Erdem, Katsumi Inoue, Johannes Oetsch, Hans Tompits, and Cemal Yilmaz. Event-sequence testing using answer-set programming.

[5] G. Brassard and P. Bratley. *Algorithmics - theory and practice*. Prentice Hall, 1988.

[6] R.C. Bryce, C.J. Colbourn, and M.B. Cohen. A framework of greedy methods for constructing interaction test suites. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 146–155, 2005.

[7] Kevin Burr and William Young. Combinatorial test techniques: Table-based automation, test generation and code coverage. In *Proceedings of the Intl. Conf. on Software Testing Analysis and Review*, pages 503–513. West, 1998.

[8] Andrea Calvagna and Angelo Gargantini. T-wise combinatorial interaction test suites construction based on coverage inheritance. *Softw. Test. Verif. Reliab.*, 22(7):507–526, November 2012.

[9] L. Carlitz. Permutations with prescribed pattern. *Mathematische Nachrichten*, 58(1-6):31–53, 1973.

[10] Yeow Meng Chee, Charles J. Colbourn, Daniel Horsley, and Junling Zhou. Sequence covering arrays. *SIAM Journal on Discrete Mathematics*, 27(4):1844–1861, December 2013.

[11] T.S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, 1978.

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

[13] D.M. Cohen, S.R. Dalal, A. Kajla, and G.C. Patton. The automatic efficient test generator (aetg) system. In *Proceedings of 1994 IEEE International Symposium on Software Reliability Engineering*, pages 303–309, 1994.

[14] Myra B. Cohen, Charles J. Colbourn, and Alan C.H. Ling. Constructing strength three covering arrays with augmented annealing. *Discrete Mathematics*, 308(13):2709–2722, 2008. Combinatorial Designs: A tribute to Jennifer Seberry on her 60th Birthday.

[15] Charles J. Colbourn, Sosina S. Martirosyan, Tran Trung, and Robert A. Walker. Roux-type constructions for covering arrays of strengths three and four. *Des. Codes Cryptography*, 41(1):33–57, October 2006.

[16] C.J. Colbourn, Yinong Chen, and Wei-Tek Tsai. Progressive ranking and composition of web services using covering arrays. In *10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 179–185, 2005.

[17] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[18] Ramirez-Acuña Daniel. Construcción de sequence covering arrays. Master's thesis, CINVESTAV, Tamaulipas, Cd. Victoria, Tamaulipas, México, Octubre 2016.

[19] Esra Erdem, Katsumi Inoue, Johannes Oetsch, Jörg Pührer, Hans Tompits, and Cemal Yilmaz. Answer-set programming as a new approach to event-sequence testing.

[20] Esra Erdem, Katsumi Inoue, Johannes Oetsch, Jörg Pührer, Hans Tompits, and Cemal Yilmaz. Event-sequence testing using answer-set programming as a new approach to event-sequence testing. *VALID 2011 - 3rd International Conference on Advances in System Testing and Validation Lifecycle*, 01 2011.

[21] Michael A. Forbes, J. Lawrence, Y. Lei, R. Kacker, and D. R. Kuhn. Refining the in-parameter-order strategy for constructing covering arrays. *Journal of Research of the National Institute of Standards and Technology*, 113:287 – 297, 2008.

[22] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187-188:52–89, 2012.

[23] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall PTR, USA, 2nd edition, 2002.

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

[25] K. V. Hanford. Automatic generation of test cases. *IBM Systems Journal*, 9(4):242–257, 1970.

[26] Alan Hartman. *Software and Hardware Testing Using Combinatorial Covering Suites*, pages 237–266. Springer US, Boston, MA, 2005.

[27] Yoshiyasu Ishigami. An extremal problem of d permutations containing every permutation of every t elements. *Discrete Mathematics*, 159(1):279–283, 1996.

[28] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

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

[30] Noritaka Kobayashi. Design and evaluation of automatic test generation strategies for functional testing of software. 2002.

[31] D. Kuhn, J.M. Higdon, James Lawrence, Raghu Kacker, and Y. Lei. Efficient methods for interoperability testing using event sequences. 25:15–18, 07 2012.

[32] D. RIchard Kuhn, James M. Higdon, James F. Lawrence, Raghu N. Kacker, and Yu Lei. Combinatorial methods for event sequence testing. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 601–609, 2012.

[33] D. Richard Kuhn and Vadim Okun. Pseudo-exhaustive testing for software. In *2006 30th Annual IEEE/NASA Software Engineering Workshop*, pages 153–158, 2006.

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

[35] Yu Lei, Raghu Kacker, D. Richard Kuhn, Vadim Okun, and James Lawrence. Ipog: A general strategy for t-way software testing. In *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, pages 549–556, 2007.

[36] Yu Lei and K.C. Tai. In-parameter-order: a test generation strategy for pairwise testing. In *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (Cat. No.98EX231)*, pages 254–261, 1998.

[37] V. I. LEVENSHTEIN. On perfect codes in deletion and insertion metric. 2(3):241–258, 1992.

[38] Vladimir Lifschitz. What is answer set programming? *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1594–1597, 2008.

[39] Oded Margalit. Better bounds for event sequencing testing. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 281–284, 2013.

[40] Rudolf Mathon and Tran Van Trung. Directed t -packings and directed t -steinersystems. *Des. Codes Cryptography*, 18(1–3):187–198, December 1999.

[41] Dr. Zbigniew Michalewicz and D. Fogel. How to solve it: Modern heuristics. In *Springer Berlin Heidelberg*, 2004.

[42] Mohamed Zabil Mohd Hazli, Zamli Kamal Z., and Othman Rozmie R. Sequence-based interaction testing implementation using bees algorithm. In *2012 IEEE Symposium on Computers Informatics (ISCI)*, pages 81–85, 2012.

[43] Patrick C. Murray and Charles J. Colbourn. Sequence covering arrays and linear extensions. In Kratochvíl Jan, Mirka Miller, and Dalibor Froncek, editors, *Combinatorial Algorithms*, pages 274–285, Cham, 2015. Springer International Publishing.

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

[45] Jeff Offutt, Shaoying Liu, Aynur Abdurazik, and Paul Ammann. Generating test data from state-based specifications. *Software Testing, Verification and Reliability*, 13(1):25–53, 2003.

[46] David L. Parnas. On the use of transition diagrams in the design of a user interface for an interactive computer system. In *Proceedings of the 1969 24th National Conference*, ACM '69, page 379–385, New York, NY, USA, 1969. Association for Computing Machinery.

[47] D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi. - the bees algorithm — a novel tool for complex optimisation problems. In D.T. Pham, E.E. Eldukhri, and A.J.

Soroka, editors, *Intelligent Production Machines and Systems*, pages 454–459. Elsevier Science Ltd, Oxford, 2006.

[48] Behçet Sarikaya. Conformance testing: Architectures and test sequences. *Computer Networks and ISDN Systems*, 17(2):111–126, 1989.

[49] G. Seroussi and N.H. Bshouty. Vector sets for exhaustive testing of logic circuits. *IEEE Transactions on Information Theory*, 34(3):513–522, 1988.

[50] J. Spencer. Minimal scrambling sets of simple orders. *Acta Mathematica Academiae Scientiarum Hungaricae*, 22(3):349–353, 1971.

[51] Jun Tarui. On the minimum number of completely 3-scrambling permutations. *Discrete Mathematics*, 308(8):1350–1354, 2008. Third European Conference on Combinatorics.

[52] Jose Torres-Jimenez, Brenda Acevedo-Juárez, and Himer Avila-George. Covering array extender. *Applied Mathematics and Computation*, 402:126122, 2021.

[53] Jose Torres-Jimenez and Idelfonso Izquierdo-Marquez. Survey of covering arrays. In *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 20–27, 2013.

[54] Jose Torres-Jimenez, Carlos Lara-Alvarez, Alfredo Cardenas-Castillo, Roberto Blanco-Rocha, and Oscar Puga-Sanchez. Three representations for set partitions. *IEEE Access*, 9:34604–34625, 2021.

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

[56] Yu-Wen Tung and W.S. Aldiwan. Automating test case generation for the new generation mission software system. In *2000 IEEE Aerospace Conference. Proceedings (Cat. No.00TH8484)*, volume 1, pages 431–437 vol.1, 2000.

[57] Gérard Viennot. Maximal chains of subwords and up-down sequences of permutations. *Journal of Combinatorial Theory, Series A*, 34(1):1–14, 1983.

[58] Ziyuan Wang and H. He. Generating variable strength covering array for combinatorial software testing with greedy strategy. *J. Softw.*, 8:3173–3181, 2013.

[59] C. Yilmaz, M.B. Cohen, and A.A. Porter. Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering*, 32(1):20–34, 2006.

[60] Xun Yuan, Myra Cohen, and Atif M. Memon. Covering array sampling of input event sequences for automated gui testing. In *ASE '07: Proceedings of the 22nd IEEE international conference on Automated software engineering*, Washington, DC, USA, 2007. IEEE Computer Society.