Adaptación de la metaheurística BFO para la construcción de Mixed Coverings Arrays

Alumna: L.I.S. Zelma Jenifer Loera Pacheco

Asesores:

CIMAT, Unidad Zacatecas:

- Dr. Carlos Alberto Lara Álvarez

UdeG, Centro Universitario de los Valles:

- Dr. Himer Avila George

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS A.C.
UNIDAD ZACATECAS

Esta investigación se realizó gracias a los siguientes apoyos del Consejo Nacional de Ciencia y Tecnología: CATEDRAS-3163, Proyecto de estímulo a la innovación 221940.

Versión 1.0, noviembre 2017.

Índice general

Índice de figuras

Índice de tablas

Resumen

Para verificar completamente el funcionamiento de un sistema se deben probar las combinaciones de todos sus componentes; sin embargo, esto puede ser muy costoso. Una alternativa es probar sólo algunas combinaciones; por ejemplo, con un *Covering Array* (CA) o un *Mixed Covering Array* (MCA). Los CA o MCA garantizan que todas las posibles t combinaciones en k componentes se hayan probado. Encontrar CA o MCA con un conjunto mínimo de pruebas es un problema NP-completo. Esta investigación propone un nuevo método para construir CA y MCA basado en la metaheurística de optimización llamada *Bacterial Foraging Optimization* (BFO) que explota el comportamiento de búsqueda de alimento de las bacterias. La idea se basa en que una bacteria –prueba– debe cubrir espacios –interacciones– no cubiertos por el conjunto de pruebas, para cubrir esos espacios las bacterias usan movimientos guiados por una función de repulsión. Aunque en la literatura se han propuesto técnicas guiadas por el comportamiento de enjambre –v.g., PSO– con funciones de búsqueda diferentes –v.g., distancia de Hamming. El método propuesto usa una función de repulsión que promueve que cada prueba sea diferente a otras pruebas. Los resultados experimentales al probar CA y MCA para casos reales y sintéticos con una fuerza t=2 hasta t=6 muestran que la técnica propuesta obtiene resultados óptimos en un tiempo menor (reducción de 40 por ciento en promedio).

Palabras clave: diseño combinatorio, pruebas de software, covering arrays, metaheurísticas, optimización, BFO, distancia de Hamming.

Agradecimientos

A través de estas líneas quiero expresar mi más sincero agradecimiento a todas las personas que con su soporte científico y humano han colaborado en la realización de este trabajo de investigación.

Agradezco en primer lugar al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado durante la realización de este proyecto.

Agradezco al Centro de Investigación en Matemáticas A.C., intitución de enorme calidad, que me brindó todo el apoyo durante mi estancia como alumna.

Quiero agradecerle a mis asesores de tesis, el Dr. Carlos Alberto Lara Álvarez y el Dr. Himer Avila George, sus conocimientos invaluables que me brindaron para llevar a cabo esta investigación, y sobretodo su gran paciencia para esperar a que este trabajo pudiera llegar a su fin.

Agradezco a los miembros del jurado, el Dr. Luis Julián Domínguez Pérez y al Maestro José Guadalupe Hernández Reveles, por las valiosas contribuciones que hicieron al trabajo final y por el tiempo que dedicaron para revisarlo.

Agradezco a mi familia. A ellos, los seres universalmente más queridos, sinceramente gracias por todo su cariño, paciencia, comprensión y apoyo desde siempre, inculcándome que todo lo que sueñe es posible realizarlo.

Dedicatoria

A mi madre y abuela María Jesús Flores Zazueta.

Por ser el pilar fundamental en todo lo que soy, por la gran fortaleza que siempre mostró, por sus consejos, sus valores y por toda la motivación, pero más que nada, por su amor incondicional. ¡Gracias por todo!, ¡Te quiero mucho y siempre te voy a extrañar!

A mi papá Manuel Loera Ayala.

Por los ejemplos de perseverancia y constancia que lo caracterizan y que me ha infundado siempre, por el valor mostrado para salir adelante y por todo su amor. ¡Te quiero mucho, papá!

A mi mamá Guadalupe Pacheco Flores.

Por todos sus consejos, por el amor que siempre me ha brindado, por cultivar e inculcar el sabio don de la responsabilidad. ¡Te quiero mucho, mamá!

Este trabajo de investigación ha sido posible gracias a ellos.

1. Introducción

En este capítulo se presenta un panorama general de la calidad del software y la importancia de la fase de pruebas en el desarrollo de sistemas de software. Se describen las estrategias básicas para crear casos de prueba. Después se explican los conceptos principales en el diseño de pruebas de interacción, tales como *Covering Arrays* (CA) y *Mixed Covering Arrays* (MCA). Finalmente, se presenta el problema de investigación, la hipótesis, los objetivos, la metodología utilizada y las contribuciones logradas.

1.1 Importancia de las pruebas en la calidad de software

En la actualidad el desarrollo de software debe adecuarse a las situaciones del mundo globalizado, necesidad de reducción de costos y agilidad en la presentación de productos. Además, es sumamente importante tomar en cuenta la calidad del software para competir con mayores posibilidades de éxito.

A continuación se presentan algunas definiciones acerca de la calidad del software:

- "El conjunto de propiedades inherentes a una cosa que permite caracterizarla y valorarla con respecto a las restantes de su especie". Diccionario de Oxford.
- "Es la concordancia con los requisitos funcionales y de rendimiento establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado de forma profesional". (Pressman, 2002)
- "Grado al que el software posee una combinación deseada de factores". (Radatz et al, 1990)
- "Es la totalidad de rasgos y atributos de un producto de software que le apoyan en su capacidad de satisfacer sus necesidades explícitas o implícitas". (for Standardization/International Electrotechnical Commission et al, 2001)

Para medir la calidad de software existen diferentes clasificaciones o agrupaciones de atributos de calidad definidos en ciertos estándares como el ISO/IEC 9126 (*International Standard Organization*, 2001) o modelos de calidad como los definidos por Pressman (2002) o Boehm et al (1976).

Los atributos de calidad definidos en el estándar ISO/IEC 9126 son:

ISO/IEC 9126					
Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Facilidad de mantenimiento	Portabilidad
-Pertinencia -Precisión -Interoperabilidad -Seguridad -Conformidad funcional	-Madurez -Tolerancia a fallas -Facilidad de recuperación -Conformidad de confiabilidad	-Entendimiento -Facilidad de aprendizaje -Operabilidad -Atractivo -Conformidad de uso	-Comportamiento en tiempo -Uso de recursos -Conformidad de eficiencia	-Facilidad de análisis -Facilidad de cambio -Estabilidad -Facilidad de ser probado -Conformidad de mantenimiento	-Adaptabilidad -Facilidad de instalación -Co-existencia -Facilidad de ser reemplazado -Conformidad de portabilidad

Tabla 1.1: Atributos de calidad considerados por el estándar ISO/IEC 9126.

Funcionalidad: Conjunto de atributos que se relacionan con la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son aquellas que satisfacen las necesidades implícitas o explícitas.

Confiabilidad: Conjunto de atributos que se basan en la capacidad de un sistema para mantener su nivel de servicio bajo condiciones establecidas por periodos específicos de tiempo.

Usabilidad: Conjunto de atributos relacionados con el esfuerzo necesario para su uso, y en la valoración individual de tal uso, por un establecido o implicado conjunto de usuarios.

Eficiencia: Conjunto de atributos relacionados con la relación entre el nivel de desempeño del software y la cantidad de recursos utilizados bajo condiciones establecidas.

Facilidad de mantenimiento: Conjunto de atributos que tienen que ver con el esfuerzo necesario para realizar modificaciones específicas.

Portabilidad: Conjunto de atributos relacionados con la capacidad del software para ser transferido de un entorno a otro.

De los atributos de calidad descritos anteriormente, este trabajo de investigación se centra en la confiabilidad del software, específicamente en la tolerancia a fallas donde se pueden localizar las pruebas de software, como se muestra en la Figura 1.1.

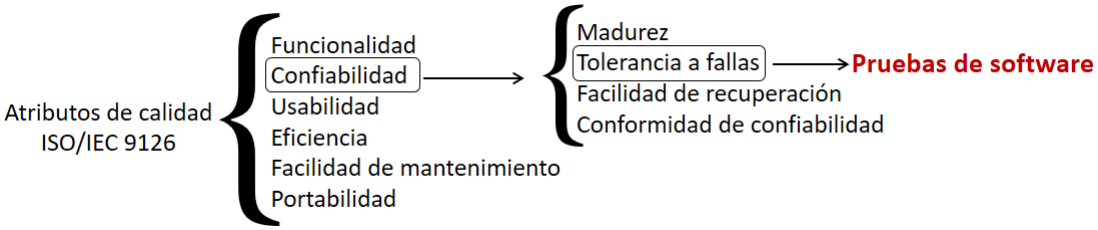

Figura 1.1: Pruebas de software en el contexto de atributos de calidad.

Una vez que el software se encuentra funcionando, según especificaciones, la confiabilidad consiste en medir la capacidad de un sistema para mantener su nivel de servicio bajo condiciones definidas por periodos específicos de tiempo (Barbacci et al, 1995). La confiabilidad considera aspectos como madurez, tolerancia a fallas, facilidad de recuperación y conformidad de confiabilidad.

La tolerancia a fallas describe la capacidad del sistema para seguir operando, en un nivel especificado, en caso de que alguna falla se presente. La mayoría de los sistemas presentan fallas en operación, por eso sería deseable conocer qué tan frecuentemente o bajo qué condiciones suceden ese tipo de eventos y repararlos; la mayor información de las fallas del software se recaba de la etapa de pruebas de software.

Por lo general, el ciclo de vida del software se compone de las etapas: análisis, diseño, programación, pruebas y mantenimiento; siendo la etapa de pruebas una de las actividades más desafiantes y costosas en el desarrollo de software (Hartman and Raskin, 2004). En la etapa de pruebas se verifica el funcionamiento de todos los componentes de un sistema de software, con la intención de encontrar los posibles fallas, lo que ayuda a mejorar la calidad de los productos desarrollados. El encargado de pruebas de software tiene la resposabilidad de diseñar los casos de prueba que localicen las fallas del sistema utilizando de manera eficiente los recursos disponibles.

Existen dos estrategias básicas que pueden ser usadas para diseñar casos de prueba, Burnstein (2003) las clasifica como pruebas de caja blanca y pruebas de caja negra.

1.1.1 Pruebas de caja blanca

También conocidas como pruebas estructurales o pruebas de caja de cristal.

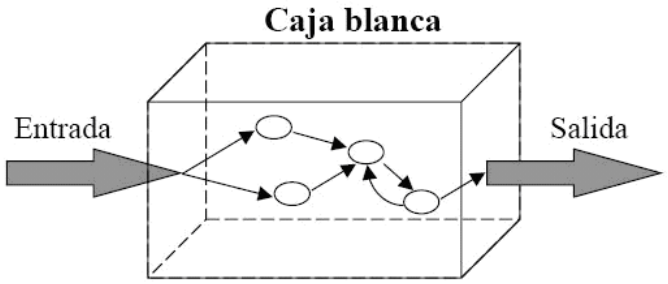

Figura 1.2: Pruebas de caja blanca.

Como se muestra en la Figura 1.2, realizan un seguimiento del código fuente mientras ejecutan cada caso de prueba, de manera que se determinan de manera concreta las instrucciones en las que existen errores. Estas pruebas normalmente dejan huecos en las especificaciones porque se aislan de su contexto y omiten comportamientos que solo ocurren al interactuar con otro elemento.

1.1.2 Pruebas de caja negra

También conocidas como pruebas funcionales o basadas en la especificación. En estas pruebas, se proporcionan las entradas del software a probar, se realiza la ejecución y se determina si las salidas producidas son equivalentes a las especificadas.

Figura 1.3: Pruebas de caja negra.

Las pruebas de caja negra no buscan reemplazar a las pruebas de caja blanca, sino que pretenden ser un enfoque complementario para encontrar errores diferentes a los de la primera prueba mencionada. Se considera que las pruebas de caja negra permiten encontrar errores como:

- Funciones incorrectas o ausentes.
- Errores de interfaz.

- Errores en estructuras de datos.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Una buena estrategia para probar un sistema implica la generación de todo el conjunto de casos que participan en su funcionamiento. Normalmente el número total de casos de pruebas de software de caja negra es extremadamente grande y no es factible realizarlos, la búsqueda de las combinaciones óptimas o mínimas da lugar al diseño de experimentos.

1.2 Diseño experimentos

La experimentación o pruebas juega un papel fundamental en la mayoría de las investigaciones científicas y de la industria, éstas obtienen información acerca del producto o servicio con el objetivo de optimizarlo y mejorar su calidad.

El diseño de experimentos tiene sus orígenes en los trabajos de Fisher (1937), desarrollados en la Estación Agrícola Experimental de Rothamsted en las cercanías de Londres, Inglaterra. De 1919 a 1925 estudió y analizó experimentos relativos al trigo que se habían realizado desde 1843. Como resultado de sus investigaciones estadísticas de éstos y otros experimentos, Fisher desarrolló el análisis de varianza y unificó sus ideas básicas sobre el diseño de experimentos.

Montgomery et al (2004) define el diseño de experimentos como una prueba o conjunto de pruebas en las que se hacen cambios deliberados en los factores de entrada de un proceso o sistema para observar e identificar las razones de los cambios que pudieran observarse en la respuesta de la salida. Los niveles son el número de alternativas o ajustes para cada factor. Los factores son las variables de interés para las cuales se quiere estudiar el impacto que tienen las mismas en la respuesta.

1.2.1 Diseños de experimentos clásicos

- **Tradicional:** Consiste en variar un factor a la vez, primero se selecciona un punto de partida de cada factor, es decir, se definen unas condiciones iniciales para realizar el experimento. Luego, se va modificando el valor de un solo factor, manteniendo los demás fijos. Posteriormente se modifica el valor del siguiente factor y sucesivamente. El inconveniente de este método es que

no se pueden observar las interacciones entre factores como se muestra en la Figura 1.4.

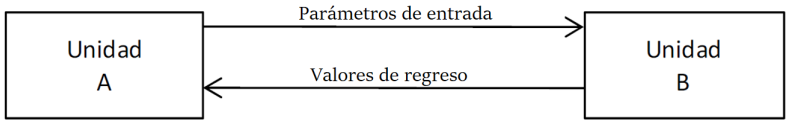

Figura 1.4: Interacción entre A y B

Se dice que existe una interacción entre las unidades A y B, cuando se cambian los parámetros de entrada de A y da por resultado diferentes valores de regreso de B. Por lo tanto, este método sólo proporciona información limitada ya que solo se puede ver un cambio a la vez.

- **Factorial completo:** Este diseño consiste en realizar todas las combinaciones posibles de varios factores, siguiendo la expresión de la Figura 1.5.

$$n = \prod_{i=1}^{k} m_i$$

Figura 1.5: Diseño factorial

Los diseños factoriales producen experimentos más eficientes que el método tradicional, pues cada observación proporciona información sobre todos los factores, y es factible ver las respuestas de un factor en diferentes combinaciones. La respuesta a cualquier factor observado en diferentes condiciones indica si los factores actúan en las unidades experimentales de manera independiente. Los diseños factoriales 2^k son diseños en los que se trabaja con k factores, todos ellos con dos niveles (se suelen denotar + y -). Estos diseños son útiles para realizar estudios preliminares con muchos factores para identificar los más importantes y sus interacciones, en la Tabla 1.2 se muestra un ejemplo de este tipo.

Si el número de factores es grande, el número de observaciones que necesita el diseño factorial es muy grande ($n = 2^k$), una alternativa es utilizar fracciones factoriales.

- **Factorial fraccionado:** Las fracciones factoriales usan sólo la mitad, la cuarta parte o incluso una fracción menor de las 2^k combinaciones de factores y se usan por una o varias de las

Combinación	X1	X2	X3
1	-	-	-
2	+	-	-
3	-	+	-
4	+	+	-
5	-	-	+
6	+	-	+
7	-	+	+
8	+	+	+

Tabla 1.2: Diseño factorial completo 2^k.

siguientes razones: el número de experimentos necesario excede a los recursos, sólo se requiere información sobre los efectos principales y las interacciones de bajo orden, se necesitan estudios exploratorios para muchos factores, o se hace la suposición de que sólo unos cuantos efectos son importantes. En la Tabla 1.3 se puede ver un diseño factorial fraccional 2^{k-p} para $k = 3$ y $p = 1$, donde se observa un diseño con sólo 4 combinaciones.

Combinación	X1	X2	X3
1	-	-	+
2	+	-	-
3	-	+	-
4	+	+	+

Tabla 1.3: Diseño factorial fraccionado 2^{k-p}.

1.2.2 Diseños de experimentos combinatorios

- ***Orthogonal Arrays* (OA):** Sean N, k, v, t enteros positivos con $t <= k$. Un orthogonal array, $\mathrm{OA}_\lambda(N, k, v, t)$ de índice λ, alfabeto v, fuerza t, es un arreglo A de tamaño N x k, donde cada elemento $\mathrm{A}_{i,j}$ toma como valor un símbolo del conjunto $S = \{1, 2..., v\}$ tal que cada N x t subarreglo contiene todas las posibles combinaciones de los v^t símbolos exactamente λ veces. Por ejemplo, en la Tabla 1.4 se muestra un OA de fuerza (grado de interacción) 2. Si se toma cualquier par de columnas, por ejemplo las últimas 2, cada una de las 4 posibles combinaciones 0,0, 0,1, 1,0, 1,1 aparecerán allí el mismo número de veces (de hecho aparecen exactamente 3 veces), estás son las características que se deben cumplir para crear OA. También hay 11

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

Tabla 1.4: Orthogonal array de fuerza 2.

columnas, lo cual significa que se pueden variar los niveles de hasta 11 variables diferentes, hay 12 renglones, lo que significa que hay 12 maneras de preparar un té o 12 diferentes casos de prueba, según sea su aplicación.

- ***Covering Arrays* (CA):** Son objetos combinatorios que a diferencia de los OA, no necesitan estar balanceados, es decir en un CA que satisface una cobertura de fuerza t, cada posible combinación de valores de variables (t-*ada*) debe estar presente al menos una vez pero no necesariamente el mismo número de veces. Por ejemplo, en la Tabla 1.5 se muestra un ejemplo de un CA de fuerza 3. Si se toma cualquier terna de columnas, cada una de las 8 combinaciones 0,0,0, 0,0,1, 0,1,0, 0,1,1, 1,0,0, 1,0,1, 1,1,0, 1,1,1 debe aparecer al menos una vez.

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

Tabla 1.5: Ejemplo de un Covering array de fuerza 3.

Un *covering array* es un objeto combinatorio, denotado por CA (N; k, v, t), que se puede describir como una matriz con $N \times k$ elementos, de manera que cada $N \times t$ subarray contiene todas las combinaciones posibles de v en t símbolos al menos una vez.

- N: número de experimentos o pruebas.
- k: Es el número de factores o componentes del sistema a probar.
- v: Es el número de símbolos por cada factor, denominado como alfabeto.
- t: Es el grado de interacción entre los factores, denominado como fuerza.

- ***Mixed Covering Arrays* (MCA):** Un *mixed covering array* a diferencia de los CA de alfabeto uniforme, permite como su nombre indica, tener factores con diferentes niveles. Lo cual permite aplicarlos a muchos campos de la ingeniería donde no es necesario que los niveles de los factores sean iguales, por ejemplo el diseño de pruebas de software y hardware. Para representar estos objetos combinatorios se usa la siguiente notación:

$$MCA(N; k, \Pi_{i=1}^{k} v_i, t)$$

En la Tabla 1.6 se presenta un MCA con 12 experimentos, 4 factores, el primero de ellos de nivel 4, los siguientes 2 de nivel 3 y el último factor de nivel 2, por último la fuerza es de 2.

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

Tabla 1.6: $MCA(12; 4, 4^1 3^2 2^1, 2)$.

El uso de los CA y MCA para construir casos de pruebas de software reduce el número de pruebas, el costo, y el tiempo empleado en dicho proceso. Dada la naturaleza de los sistemas de software, se tiene un conjunto de componentes que se pueden probar con distinto número de valores de datos. Para explicar de una mejor forma, se muestra el siguiente ejemplo:

Etiqueta	Sistema Operativo	Sensor	Servidor Web	Navegador Web
0	Windows Phone	Acelerómetro	Apache	Mozilla Firefox
1	iOS	Giroscopio	Tomcat	Google Chrome
2	Android	—	—	—

Tabla 1.7: Ejemplo de un sistema con diferente número de valores para c/u de sus parámetros.

Supóngase un sistema de software que muestra, en un navegador web, los datos de dos de los sensores de un dispositivo móvil. Este sistema cuenta con 4 parámetros:

- Sistema Operativo, con tres (0, 1, 2) posibles configuraciones.
- Sensor, Servidor Web y Navegador Web con dos (0, 1) posibles configuraciones.

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

Tabla 1.8: Conjunto de casos de prueba que cubre todos los pares de parámetros.

La Tabla 1.8 es un ejemplo de un *mixed covering array* $MCA(6;4,3^1,2^3,2)$, donde $N = 6$, $k = 4$, $t = 2$, la cardinalidad del primer parámetro es 3 y del resto de parámetros 2. A diferencia de los OA y los CA, en los MCA cada parámetro tiene una cardinalidad distinta.

Caso de prueba	Sistema Operativo	Sensor	Servidor Web	Navegador Web
1	Windows Phone	Acelerómetro	Apache	Mozilla Firefox
2	Windows Phone	Giroscopio	Tomcat	Google Chrome
3	iOS	Acelerómetro	Apache	Google Chrome
4	iOS	Giroscopio	Tomcat	Mozilla Firefox
5	Android	Acelerómetro	Tomcat	Mozilla Firefox
6	Android	Giroscopio	Apache	Google Chrome

Tabla 1.9: Interpretación del conjunto de casos de prueba de la Tabla 1.8 .

Si se sustituye cada etiqueta (0, 1 ó 2) por la configuración correspondiente, el conjunto de casos de prueba de que se obtendría es el que se muestra en la Tabla 1.9.

A continuación se presentan otros usos de CA y MCA, según Colbourn (2004):

- **Pruebas de hardware:** En un circuito, las señales de entrada interactúan a través de operaciones aritméticas y lógicas para producir un vector de salida deseado. Es deseable que el funcionamiento de tales circuitos sea correcto; sin embargo, pueden existir errores en éstas operaciones. El problema de pruebas de hardware se puede abordar utilizando CA y MCA.
- **Ensayo de materiales:** Los materiales se pueden combinar para producir mejores propiedades tales como resistencia, flexibilidad y punto de fusión. Sin embargo, ciertas combinaciones pueden ser tóxicas o explosivas, por lo cual se deben evitar. Los CA y MCA pueden ayudar en el diseño de estos experimentos.
- **Las interacciones que regulan la expresión de genes:** Las hormonas tienen un impacto en la expresión de genes particulares, y pueden interactuar unos con otros. Aunque se pueden examinar las posibles combinaciones de las hormonas, la interacción entre un pequeño número de hormonas las que resultan de interés. Nuevamente, los CA y MCA son una herramienta de modelado muy adecuado para tal problema.

1.3 Definición del problema

En la generación de casos de prueba para sistemas de software, $\mathscr{I}$ es un conjunto de n parámetros de entrada, donde cada $n \in \mathscr{I}$ tiene $k(I)$ valores de datos, y $\mathscr{O}$ es una colección de subconjuntos de $\mathscr{I}$ donde las interacciones de los parámetros en cada subconjunto $O \in \mathscr{O}$ se cree que afectan el resultado del sistema. Un caso de prueba para ($\mathscr{I}$, $\mathscr{O}$, k) es una t-ada $(t_1, t_2, ..., t_n)$ que especifica el valor de cada parámetro de entrada en $\mathscr{I}$. La meta es generar el conjunto más pequeño de casos de prueba que cubra todas las combinaciones de cada $O \in \mathscr{O}$, llamaremos óptimo a dicho conjunto de pruebas, en la literatura, la generación de un conjunto óptimo de casos de prueba se conoce como un problema del tipo NP-completo.

1.4 Hipótesis

Es posible diseñar e implementar un algoritmo de optimización combinatoria para la construcción de CA y MCA de fuerza $t <= 6$, que sea capaz de mejorar la calidad de solución y el rendimiento del tiempo de ejecución comparado con los mejores casos reportados en la literatura.

1.5 Objetivos

Objetivo general:

Diseñar e implementar un algoritmo para construir CA y MCA que permita mejorar la calidad de los resultados reportados en la literatura.

Objetivos específicos:

- Identificar metaheurísticas no usadas anteriormente para la construcción de CA y MCA.
- Implementar un algoritmo basado en la metaheurística BFO para construir CA y MCA.
- Comparar los resultados obtenidos por el método propuesto con los mejores resultados propuestos en la literatura.

1.6 Alcances y limitaciones

Se adaptó el algoritmo BFO para la construcción de CA y MCA con los siguientes parámetros:

- factores = $4 < k < 50$
- vocabulario = $2 < v < 6$
- fuerza = $2 < t < 6$

1.7 Metodología

La metodología para realizar este trabajo de investigación se ha dividido en tres etapas. La primera se centra en el estudio de los principales conceptos relacionados con el problema de investigación y la forma como éste se puede abordar; como resultado, se tiene el diseño del algoritmo para la construcción de CA y MCA. La segunda etapa está enfocada en la sintonización de los principales parámetros del algoritmo; durante ésta, se realizan los cambios necesarios, adicionando o excluyendo

características, para lograr obtener un buen rendimiento del algoritmo. Finalmente, la última etapa se centra en la evaluación de la versión final del algoritmo y el análisis de los resultados obtenidos.

Primera etapa:

Seleccionar la técnica de un algoritmo para optimizar la construcción de MCA. Diseñar e implementar una primera versión del algoritmo con diferentes alternativas de configuración en sus principales parámetros.

Segunda etapa:

Realizar un conjunto de pruebas para evaluar la mejor sintonización de los principales parámetros del algoritmo, utilizando un conjunto de instancias desde fuerza = 2 hasta fuerza = 6. Elegir la mejor combinación de parámetros con la que el algoritmo tendrá un buen rendimiento para utilizarlo en la fase de experimentación.

Tercera etapa:

Diseñar y ejecutar el experimento usando la mejor combinación de parámetros del algoritmo, resultado de la sintonización que se realizó en el paso anterior, para realizar una evaluación comparativa por otros métodos que construyen MCA. Evaluar el rendimiento del algoritmo en la construcción de un benchmark de instancias que abarquen fuerzas desde 2 hasta 6, el cual contenga los mejores casos reportados en la literatura.

1.8 Contribuciones

Las contribuciones de este trabajo de investigación son las siguientes:

- Una implementación de la metaheurística BFO para la construcción de CA y MCA de fuerza $2 <= t <= 6$, capaz de mejorar la calidad de solución de los mejores casos reportados en diferentes trabajos relacionados.
- Análisis significativo de las configuraciones de parámetros del algoritmo propuesto.

1.9 Organización de la tesis

Capítulo 1: Este capítulo presenta un panorama general de los sistemas de software, enfatizando la relevancia de la fase de pruebas dentro de la calidad de éstos y la importancia de realizarlas de manera eficiente. Asimismo, se describe de manera general el diseño de experimentos puntualizando las ventajas del uso de diseño combinatorio dentro de las pruebas de software, basado en los resultados de estudios empíricos sobre las fallas registrados en diferentes sistemas de software.

Capítulo 2: Consiste en describir el estado del arte, se revisan diferentes estrategias que han sido implementadas para la construcción de CA y MCA, entre las que destacan las métodos exactos, algebraicos, voraces y basados en metaheurísticas. Además de lo anterior, se presentan diferentes algoritmos que construyen CA y MCA; algunos de ellos enfocados en inicialización y otros en el proceso post-optimización de CA y MCA. Algunos de los mejores resultados se han construido por el algoritmo *Simulated Annealing*.

Capítulo 3: Describe la propuesta para la construcción de CA y MCA de fuerza $t = 2$ hasta $t = 6$. La propuesta se enfoca en el diseño y la implementación de un algoritmo que utiliza una adaptación de la metaheurística llamada BFO, el cual se explica a detalle en este capítulo. Es importante mencionar que el algoritmo implementado cuenta con dos funciones importantes llamadas OneCol y OneRow, estas funciones tienen diferentes opciones para la configuración de cada una, esto atribuye al algoritmo de solución la posibilidad de ser ejecutado bajo una diversa gama de configuraciones.

Capítulo 4: Presenta la sintonización del algoritmo propuesto para encontrar la mejor configuración de parámetros, que encuentre la solución óptima en el mejor tiempo de ejecución. Posteriormente se lleva a cabo la experimentación para casos reales y sintéticos, la experimentación se valida con un análisis comparativo de los resultados obtenidos.

Capítulo 5: Presenta el análisis y evaluación de los resultados obtenidos por el algoritmo propuesto, las conclusiones y el trabajo futuro.

2. Estado del arte

Dado que los CA y MCA son de gran importancia en las pruebas de interacción, se ha desarrollado una amplia gama de trabajos de investigación relacionados con su construcción. En este capítulo se presenta un panorama de los principales métodos del estado del arte.

2.1 Métodos para construir CA y MCA

A continuación se proporciona una descripción de algunos métodos para construir CA y MCA, entre los cuales figuran los métodos exactos, los métodos algebraicos, los métodos voraces y los métodos basados en metaheurísticas (Torres-Jimenez and Izquierdo-Marquez, 2013), después se proporciona una breve descripción de las investigaciones basadas en metaheurísticas.

2.1.1 Métodos exactos

Dado los valores k, t y v, los métodos exactos construyen un CA (N; t; k; v) con un número de filas N óptimo. Los métodos exactos tienen técnicas para acelerar el proceso de búsqueda; pero aún así sólo son prácticos para la construcción de CA pequeños (Torres-Jimenez and Izquierdo-Marquez, 2013).

Meagher (2002), presenta una propuesta para construir CA no isomórficos utilizando un algoritmo de retroceso que construye cada fila del CA recursivamente e independientemente. El algoritmo construye todos los CA posibles de un tamaño específico, pero sólo funciona con el alfabeto binario.

Yan and Zhang (2006), proponen EXACT (*EXhaustive seArch of Combinatorial Test* suites, por sus siglas en inglés) que permite la generación de MCA, reduciendo el espacio de búsqueda para evitar explorar MCA isomorfos. Utiliza técnicas de retroceso para volver al proceso cuando la construcción de MCA se estanca. En esta propuesta, el tiempo de cálculo aumenta significativamente cuando el número de filas N del MCA es cercano al óptimo. Además, no encuentra la solución para

ciertas configuraciones.

Bracho-Rios et al (2009), presentan un algoritmo para la creación de CA con una técnica de retroceso, que crea el CA con un orden lexicográfico de filas y columnas para evitar la búsqueda de arreglos simétricos. El algoritmo propuesto es capaz de igualar las mejores soluciones encontradas por EXACT para problemas pequeños de CA y toma menos tiempo. El algoritmo está limitado al alfabeto binario y a las fuerzas 3 <= t <= 5.

2.1.2 Métodos algebraicos

Los algoritmos basados en métodos algebraicos construyen CA en tiempo polinomial usando reglas predefinidas y son aplicables para algunos casos ofreciendo construcciones eficientes en términos de tiempo; sin embargo, en la actualidad, es difícil generar resultados precisos en una amplia gama de valores de entrada (Turban, 2006), (Zhang et al, 2014).

Por ejemplo, Katona (1973) y Kleitman and Spencer (1973) proponen dos algoritmos que crean CA óptimos en el caso de fuerza $t = 2$ y el alfabeto $v = 2$. Después, Chateauneuf et al (1999) proponen un algoritmo capaz de crear CA de fuerza $t = 3$ usando acciones de grupo sobre el alfabeto, y luego Meagher and Stevens (2005) adaptaron este método para crear CA de fuerza $t = 2$ utilizando vectores de arranque que son girados y traducidos para obtener el CA óptimo. Hartman (2005) propone un método de cuadrar el número de columnas de un CA utilizando una matriz ortogonal como una herramienta intermedia. Colbourn et al (2006) proponen un método para multiplicar dos CA de la fuerza $t = 2$, dando lugar a un nuevo CA con el producto de las columnas originales y la suma de las filas originales. Por último, Colbourn (2010) propone la creación de CA basado en matrices ciclotómicas.

2.1.3 Métodos voraces

Los métodos voraces son algoritmos de tipo deterministas; es decir, que dada una entrada particular siempre producirán la misma salida. Estos métodos han probado ser rápidos y precisos para una gran variedad de entradas; sin embargo, no siempre garantizan CA óptimos en comparación con otros métodos (Turban, 2006).

Los dos principales enfoques de los métodos voraces para construir CA son:

- ***one-row-at-time***

  El sistema AETG propuesto por Cohen et al (1997) intenta construir CA pequeños usando un algoritmo voraz. Selecciona una única prueba a la vez, repitiendo esto hasta que todos los pares estén cubiertos en al menos una de las pruebas seleccionadas. Dado que el objetivo es minimizar el número de pruebas, AETG se concentra en la selección de cada prueba para maximizar el número de pares cubiertos. Sus principales contribuciones son:

  1. Muestra un límite logarítmico en el número de pruebas necesarias en función de k.
  2. Describe una heurística (voraz) para la selección de pruebas.

  Los métodos voraces incluyen el Algoritmo de Densidad Determinística (DDA, por sus siglas en inglés) (Bryce and Colbourn, 2007) que crea CA de fuerza $t = 2$ agregando una fila a la vez.

- ***one-column-at-time***

  Lei and Tai (1998), introdujeron un nuevo método llamado *In-Parameter-Order* (IPO). Este método comienza con la generación de un conjunto de pruebas con dos parámetros y agrega iterativamente nuevos parámetros (es decir, comienza con un CA de k columnas y obtiene un CA de $k+1$ columnas). El proceso de extensión tiene dos etapas: crecimiento horizontal y crecimiento vertical. Supongamos que el nuevo parámetro tiene q símbolos. Para $1 <= j <= q$ la fila j del nuevo parámetro obtiene el valor j; y los elementos restantes del nuevo parámetro obtienen el valor que cubre la mayor cantidad de pares no cubiertos. Los pares no cubiertos en el crecimiento horizontal se almacenan en una lista y se cubrirán más adelante en la etapa de crecimiento vertical.

  Este algoritmo IPO se generaliza a continuación para las fuerzas mayores a 2 en IPOG (Lei et al, 2007), que luego se mejora en IPOG-F (Forbes et al, 2008). También Calvagna and Gargantini (2012), extendió las ideas de IPO para construir MCA de cualquier fuerza.

2.1.4 Métodos basados en metaheurísticas

El problema de encontrar un conjunto mínimo de pruebas con un CA o un MCA pertenece a la clase NP-duros, es decir, no existe ningún algoritmo que permita determinar la solución óptima al problema en tiempo polinomial. Una estrategia es utilizar metaheurísticas para aproximar una

solución óptima en tiempos factibles.

A continuación se presentan los métodos basados en metaheurísticas que a la fecha han logrado generar CA y MCA de tamaño óptimo a un costo significativo en tiempo de ejecución (Turban, 2006), (Kacker et al, 2013).

• ***Tabu Search***, se ha aplicado con éxito a una variedad de problemas de optimización combinatoria (Glover and Laguna, 1997), (Gendreau, 2002)

• ***Simulated Annealing***, ha producido los resultados más precisos, y ha encontrado nuevas y mejores soluciones para parámetros de entrada con valores múltiples (Gendreau and Potvin, 2010).

• ***Genetic Algorithms***, basan su modelo en la selección de los individuos más aptos, generalmente utilizan la selección, el cruce, la mutación y el reemplazo como operadores. (Luke, 2013).

• ***Ant Colonies Optimization***, simula computacionalmente la comunicación indirecta que las hormigas realizan para establecer la ruta más corta entre su lugar de partida y una fuente de alimento. (Dorigo and Stützle, 2004).

• ***Particle Swarm Optimization***, se inspira en el comportamiento o movimientos de ciertos organismos en la naturaleza como enjambres de abejas, bandadas de aves o cardúmenes de peces. Intentan, después de explorar en varias áreas, localizar aquellas regiones del espacio donde la comida está más concentrada. En última instancia, todo el enjambre orientará la búsqueda en este nueva dirección (Kennedy and Eberhart, 1995).

• ***Harmony Search***, basa su operación en el proceso de improvisación musical que tiene lugar cuando un músico busca producir una armonía agradable. Los componentes del algoritmo son: uso de la memoria armónica, ajuste de tono y aleatoriedad. (Geem et al, 2001).

2.2 Metaheurísticas sobresalientes

2.2.1 *Tabu Search*

Tabu Search (TS, por sus siglas en inglés) es un enfoque para la optimización de búsqueda local que se enfrenta a diferentes problemas de optimización combinatoria. Esta estrategia propuesta por Glover and Laguna (1997), busca evitar caer en ciclos y mínimos locales, prohibiendo o penalizando movimientos que tienen la solución en la siguiente iteración, para señalar en el espacio de solución

visitado previamente, de ahí el término tabú . La idea básica de TS es realizar una búsqueda local evitando caer en un mínimo local mediante la elección de movimientos que no mejoran la solución, con la suposición de que una estrategia mal elegida puede dar más información que una buena aleatoria. Para evitar volver a soluciones pasadas y quedar atrapados dentro de ciclos, se utiliza una memoria temporal, llamada lista tabú, que almacena el historial de búsqueda reciente.

Nurmela (2004) usó TS para encontrar CA óptimos. El algoritmo comienza con una matriz M generada aleatoriamente de tamaño N x k, donde las filas corresponden al alfabeto CA. El coste de la matriz se define por el número de combinaciones que faltan. Una combinación que falta entonces se selecciona al azar. Se verifica que las filas requieren un solo elemento para ser cambiado para que la fila cubra la combinación seleccionada. Estos cambios son los movimientos del vecino actual. El coste se calcula de acuerdo con cada movimiento del vecino y se selecciona el movimiento que genera el menor coste, siempre y cuando no sea tabú (es decir, en la lista tabú). Si hay más de un movimiento con el mismo coste, uno de los movimientos se selecciona al azar. El proceso se repite hasta que el coste de la matriz M es cero o se alcanza el número máximo de movimientos. Los resultados mostraron que la implementación mejoró algunas de las mejores soluciones previamente conocidas. Sin embargo, un inconveniente importante de este algoritmo es que consume mucho más tiempo de cálculo que otros algoritmos. Además, en ciertos casos, terminó tomando varios meses para resolver las diferentes instancias de prueba.

Walker and Colbourn (2009) utilizan el algoritmo de búsqueda tabú para generar CA a partir de la permutación de vectores y objetos matemáticos, conocido como *Covering Perfect Hash Families* (CPHF). Esta representación de una CA es capaz de encontrar eficientemente matrices más pequeñas para mayores fuerzas t. También habilitó búsquedas para t mayores o iguales a 5.

Gonzalez-Hernandez et al (2010) presentan un enfoque basado en la búsqueda de tabú conocido como TSA para la construcción de *mixed covering arrays* (MCA) de fuerza variable. Las principales características de este enfoque se encuentran en los siguientes aspectos: en primer lugar, el algoritmo selecciona a partir de un conjunto de funciones de vecindad predefinidas, donde a cada una se le asigna una probabilidad de ser seleccionado para crear un nuevo vecino; en segundo lugar, tiene un cálculo eficiente de la función objetivo para determinar la mejor probabilidad de selección para cada

función vecinal; y en tercer lugar, tiene una nueva función de inicialización. Como el rendimiento de TS depende de los valores de las probabilidades asignadas, se realizó un proceso de sintonización en las configuraciones de estas probabilidades. La configuración utilizada por TSA permitió generar MCA de menor tamaño y en menos tiempo. TSA mejoró el tamaño de los MCA en comparación con IPOG-F, y encontró la solución óptima en 15 casos de los 18 que componen el conjunto completo. Estas instancias van desde un alfabeto v de 2 a 11, el número de columnas k de 2 a 20, y una fuerza t de 2 a 6.

Hernández (2013) presenta un algoritmo de optimización combinatoria para la construcción de MCA de fuerza variable denominado *Mixed Tabu Search* (MiTS), que utiliza la metaheurística de la estrategia de la búsqueda tabú y su característica principal es la mezcla de diferentes funciones de vecindario, cada uno con una cierta probabilidad de ser seleccionado. Otras características importantes incluidas en el diseño de MiTS son el tamaño de la lista de tabu utilizada y la función de inicialización para crear la solución inicial.

Para cada uno de estos aspectos, se proponen tres alternativas diferentes: aleatoria, utilizando la distancia de Hamming, y a través de un subconjunto de t columnas. Los resultados experimentales indicaron que el MiTS fue capaz de establecer 91 nuevas dimensiones óptimas y correspondieron a 36 de los mejores casos reportados en la literatura, de los cuales 31 ya eran óptimos, por lo que no eran susceptibles de mejora.

Zamli et al (2016) propone una nueva estrategia híbrida de generación de pruebas, basada en la fuerza de interacción t, llamada HHH (*High Level Hyper-Heuristic*). Esta estrategia adopta *Tabu Search* como su metaheurística de alto nivel y aprovecha la fortaleza de cuatro metaheurísticas de bajo nivel: *Teaching Learning based Optimization, Global Neighborhood Algorithm, Particle Swarm Optimization, y Cuckoo Search Algorithm*. HHH es capaz de capitalizar las fortalezas y limitar las deficiencias de cada algoritmo individual de una manera colectiva y sinérgica. A diferencia de las hiperheurísticas existentes, HHH se basa en tres operadores definidos que están basados en la mejora, la intensificación y la diversificación, para seleccionar de forma adaptativa la metaheurística más adecuada en un momento determinado. Los resultados son prometedores ya que HHH logra superar las estrategias t-way existentes. HHH es una hiperheurística capaz de generar una solución "buena,

rápida y barata".

2.2.2 *Simulated Annealing*

Simulated Annealing (SA) es una técnica de optimización estocástica de propósito general basada en los pasos para el recocido de metales, es decir, simula los pasos empleados en la industria para obtener materiales que son más resistentes y poseen mejores cualidades (Kirkpatrick et al, 1983). El método comienza con un proceso de calentamiento que básicamente consiste en fundir el material a una temperatura alta hasta que alcanza su estado líquido. En este punto, los átomos aumentan significativamente su movilidad dentro de la estructura del material. Un proceso de enfriamiento comienza entonces cuando la temperatura se baja gradualmente en etapas, hasta que los átomos se ajustan correctamente antes de perder completamente su movilidad y lograr así el equilibrio térmico. Cuando el proceso está completo, es posible lograr una estructura altamente regular y estable. Los estudios han demostrado que con caídas bruscas de la temperatura o no esperando bastante tiempo en cada etapa, la estructura resultante no es altamente estable.

Cohen et al (2008) hace ciertas modificaciones a la propuesta original, incluyendo un refinamiento al algoritmo *Simulated Annealing* que permitió encontrar casos de prueba más rápido e incluye construcciones algebraicas que hicieron posible construir casos de prueba mucho más pequeños. También encontró nuevas dimensiones para algunas CA de fuerza $t = 3$. Este enfoque híbrido se denomina recocido aumentado.

Torres-Jimenez and Rodriguez-Tello (2010) presentan una nueva implementación del algoritmo SA para construir CA de vocabulario $v = 2$ de fuerza $t = 2$ hasta $t = 5$, las cuales integran tres características importantes que determinan su desempeño. En primer lugar, incorpora una heurística eficiente para generar soluciones iniciales de buena calidad; segundo, el diseño de una función de vecindad compuesta que permite a la búsqueda reducir rápidamente el costo total de las soluciones candidatas, evitando al mismo tiempo caer en mínimos locales; en tercer lugar, un programa de enfriamiento que permite al algoritmo converger mucho más rápido al generar soluciones de calidad. El algoritmo se comparó con el Algoritmo de Densidad Determinística (DDA), *Tabu Search*, y los algoritmos IPOG-F (Forbes et al, 2008). Los resultados mostraron que SA encontró nuevos límites superiores y emparejó otras soluciones previamente conocidas de las referencias seleccionadas.

Rodriguez-Cristerna and Torres-Jimenez (2012) presentaron un enfoque híbrido denominado SA-VNS para la construcción de MCA (CA con diferentes posibles valores en k columnas) basado en *Simulated Annealing* y una función de búsqueda de la variable vecindad (VNS).

Los CA y MCA son estructuras combinatorias que tienen aplicaciones en campos como pruebas de software y detección de troyanos de hardware. Torres-Jimenez et al (2017) proponen un algoritmo de recocido simulado de dos etapas para construir MCA, se ejemplifica a través de la construcción de MCA de fuerza *t=3* donde obtiene 579 nuevos límites superiores. Para mostrar la generalidad la propuesta, definen un nuevo punto de referencia compuesto de 25 instancias de MCA tomadas de la literatura, todas las instancias fueron mejoradas.

2.2.3 *Genetic Algorithms*

Los algoritmos genéticos fueron introducidos por Holland en Kacker et al (2013) para entender el proceso adaptativo de los sistemas naturales. Se inspiran en el proceso biológico de la evolución de la especie y utiliza la analogía de la supervivencia de los individuos más aptos (Yu and Gen, 2010). Comienza con una población inicial, que evoluciona a través de generaciones representadas por iteraciones. En cada iteración, los individuos son evaluados por una función de fitness y solo sobreviven los mejores individuos de la población. Las generaciones siguientes se generan aplicando sucesivamente operadores genéticos tales como selección, crossover, mutación y reemplazo. Un algoritmo genético (GA) es un método adaptativo utilizado para resolver problemas de búsqueda y optimización. Stardom (2001) presenta los resultados de la comparación de los algoritmos de optimización SA, TS y GA para la construcción de CA. Los tres enfoques fueron capaces de encontrar nuevas dimensiones. Sin embargo, los algoritmos genéticos no son eficaces en la búsqueda de CA de calidad. Se tardan más tiempo, no sólo en los movimientos, sino también para encontrar un CA óptimo.

Bryce and Colbourn (2007) presentan un algoritmo genético para la construcción de CA. En este algoritmo, la población es un conjunto de matrices con tuplas que faltan; la aptitud de las matrices es el número de tuplas que faltan, por lo que las matrices más aptas son aquellas que tienen menos tuplas que faltan. Para seleccionar a los padres de los nuevos individuos, el conjunto S de los individuos en la población actual se divide en dos grupos de tamaño $jSj = 2$; En cada grupo los individuos

son ordenados aleatoriamente y los i-ésimo miembros de cada grupo se combinan con el operador crossover para producir jSj nuevos individuos. El operador de crossover consiste en seleccionar un conjunto E de coordenadas $(i; j)$ de uno de los padres y al copiar estas coordenadas en un hijo, las coordenadas restantes para completar al hijo se toman del segundo padre. El conjunto E que se toma de uno de los padres puede ser el primer i filas completas, las primeras j columnas completas o un bloque conformado por las celdas en las primeras filas y en las primeras columnas. En cada caso se satisface la condición $i < N$ y $j < k$, donde N y k son las dimensiones de las matrices de la población, para asegurar que los individuos producidos por el operador crossover tengan coordenadas de ambos padres. Después de eso, el operador de mutación se aplica a los nuevos individuos jSj; este operador consiste en cambiar el contenido de una célula seleccionada al azar por otro elemento del alfabeto. A continuación se calcula la media de la aptitud de los padres jSj y de los hijos jSj; Las matrices T con una aptitud inferior a la media pasan a la siguiente generación, más matrices jSj que pertenecen a T seleccionadas aleatoriamente de las matrices con aptitud mayor o igual que la media.

Rodriguez-Tello and Torres-Jimenez (2009) presentan un algoritmo memético (algoritmo genético que incluye el conocimiento del problema para encontrar mejores soluciones a través de un optimizador local) para encontrar soluciones óptimas para la construcción de CA de vocabulario binario y de fuerza $t = 3$, que incorporaron características importantes tales como incluir una heurística eficiente para generar una población inicial de buena calidad y un operador de búsqueda local basado en la afinación del algoritmo SA. Los resultados computacionales, comparados con otros en la literatura entre los que se encuentran IPOG-F y TS, mostraron que el algoritmo propuesto mejoró en nueve casos las soluciones más conocidas y coincidió con los otros resultados.

Sabharwal et al (2016) presenta G-PWiseGen, una propuesta general de una herramienta de código abierto existente llamada PWiseGen, que se utiliza para generar casos de prueba de fuerza 2. El principal inconveniente de PWiseGen es que se requiere conocer de antemano el tamaño N de la prueba Como entrada. G-PWiseGen genera CA para la fuerza t menor o igual a 2, y al incorporar un algoritmo de búsqueda binaria es posible establecer límites inferiores y superiores para el tamaño de la CA, eliminando la necesidad de conocer N. G-PWiseGen se comparó con otras fuentes abiertas Herramientas para generar CA, como ACTS, Jenny, TVG y CASA. Los resultados mostraron que

G-PWiseGen toma mucho más tiempo para generar CAs con respecto a las otras proposiciones, debido a la complejidad en las operaciones de cruce y mutación cuando se aumenta la resistencia t. Sin embargo, el tamaño de las CA generadas compensan el tiempo empleado en su construcción.

2.2.4 *Ant Colony Optimization*

Un caso de prueba en este algoritmo se representa como una ruta desde un punto de inicio hasta un punto final de destino. Cuando una hormiga alcanza el nodo objetivo, se deposita una cantidad de feromonas en cada trayectoria de las que ha visitado, proporcional a la calidad de la solución. Cuando una hormiga se requiere para elegir entre diferentes caminos, elige el camino con la mayoría de las feromonas

El algoritmo Ant Colony (ACA), propuesto en 1999 por Dorigo Dorigo et al (1996) se basa en el comportamiento estructurado de las colonias de hormigas cuando se busca alimento. En este enfoque, cada ruta de un punto de partida a un punto final se asocia con una solución candidata para un problema dado. Cuando una hormiga alcanza el punto final, la cantidad de feromona depositada en cada borde (vértice) de la trayectoria seguida por la hormiga es proporcional a la calidad de la solución candidata correspondiente. Cuando una hormiga tiene que elegir entre los diferentes bordes en un punto dado (nodo), el borde con la mayor cantidad de feromona se elige con mayor probabilidad. Como resultado, las hormigas finalmente convergen hacia el camino más corto.

En 2004, Shiba presentó un algoritmo de generador de pruebas basado en algoritmos genéticos y ACA, y los comparó con otros algoritmos que incluyen SA, IPO y The Automatic Eficcient Test Generator (AETG) Cohen et al (1994) que utiliza la estrategia codiciosa. Los resultados para la resistencia $t \in (2-3)$ mostraron un buen desempeño a nivel general con respecto al tamaño de los casos de prueba (valor de N en una CA) y la cantidad de tiempo requerido para esto. Sin embargo, los resultados del algoritmo genético no siempre fueron óptimos. Los resultados obtenidos por este algoritmo fueron mejores que los generados por la GA. En 2009, Chen Chen et al (2009) adaptó el algoritmo ACA para construir un conjunto de pruebas priorizado para la interacción de pares, una CA y una MCA de fuerza $t = 2$. Especi En realidad, propuso cuatro algoritmos para generar pruebas basadas en ACA, buscando una implementación más efectiva. Aunque los resultados fueron competitivos, su desempeño no pudo ser generalizado para instancias con características diferentes.

2.2.5 *Particle Swarm Optimization*

El algoritmo de optimización de enjambre de partículas (PSO) es un método popular de optimización propuesto por Kenndy and Eberhart (1995). PSO intenta optimizar un problema que comienza con el manejo de un cierto número de soluciones candidatas. Cada solución está representada por una partícula que trabaja en un espacio de búsqueda para encontrar una mejor posición o solución al problema. La población en su conjunto se conoce como un enjambre. Como tal, cada partícula tiene una posición aleatoria y actualiza su posición iterativamente con la esperanza de encontrar mejores soluciones. Cada partícula también mantiene información esencial acerca de sus movimientos. Aplicada a la construcción de CA, una partícula normalmente representaría un caso de prueba. Cada partícula está asociada con un factor de ponderación que representa el número de interacciones cubiertas por el caso de prueba. Ahmed et al (2011) desarrollan una nueva estrategia para generar pares de datos de prueba basados en PSO, denominado Pairwise Particle Swarm (PPSTG). El estudio evaluó el desempeño de esta propuesta en términos del tamaño de las pruebas generadas contra otras estrategias y herramientas. En una primera etapa, PPSTG se comparó con los resultados publicados en la literatura para la GA, ACA, AETG y el algoritmo IPO (Lei and Tai, 1998), un codicioso algoritmo. PPSTG generó conjuntos de pruebas con resultados satisfactorios en la mayoría de los experimentos. Sin embargo, GA y ACA generaron tamaños ligeramente mejores que PPSTG, que se desempeñó mejor que AETG pero SA generó los resultados más óptimos. Ahmed and Zamli (2011) presentan el generador de ensayos de enjambres de partículas VS (VS-PSTG) para generar casos de prueba en interacciones de fuerza variable (VS). VS-PSTG adopta PSO para asegurar la reducción óptima del tamaño de las pruebas. Los resultados en comparación con otras estrategias y configuraciones se consideraron competitivos. Se realizó un estudio de caso empírico sobre un sistema de software no trivial para mostrar su aplicabilidad y determinar la eficiencia en la generación de casos de prueba, con resultados prometedores. Ahmed et al (2012) demuestran la eficiencia del Generador de Pruebas en T (PSTG), una estrategia para generar CA de fuerza variable, que maneja altas resistencias de interacción de hasta t = 6. PSTG es computacionalmente más ligero en comparación con otros Optimización debido a la simplicidad en la estructura del algoritmo PSO en el que se basa, y también supera a otras estrategias en relación con los tamaños generados para la

CA. Mahmoud and Ahmed (2015) presentan una estrategia para la construcción de CA utilizando lógica difusa para ajustar los parámetros heurísticos utilizados por el algoritmo PSO. El algoritmo se probó con diferentes propuestas incluyendo SA y *Hill Climbing*. Los resultados mostraron una mejora significativa en términos del tamaño de la CA generada. Sin embargo, el mecanismo difuso requería requisitos computacionales adicionales. En consecuencia, la estrategia es comparativamente lenta en la generación de CA de fuerza $t > 4$.

2.2.6 *Harmony Search*

Para este proceso, hay tres opciones posibles: uno, tocando una canción exactamente como se la conoce, como está en su memoria; dos, tocando algo similar a la melodía mencionada con un ligero ajuste de tono; y tres, componiendo una nueva melodía con notas seleccionadas aleatoriamente. Estas tres opciones se formalizan en (Geem et al, 2001) y corresponden a los componentes del algoritmo: uso de la memoria armónica, ajuste de tono y aleatoriedad. Existen varias variaciones de esta propuesta, incluyendo la búsqueda de armonía global que combina los conceptos de búsqueda de armonía con PSO.

El algoritmo Harmony Search (HS) propuesto por Geem et al (2001), simula el proceso de improvisación musical para encontrar un perfecto estado de armonía. Esta armonía en la música es análoga a encontrar un óptimo en un proceso de optimización. Un músico siempre tratará de producir una pieza musical con perfecta armonía. Una solución óptima en un problema de optimización debe ser siempre la mejor solución disponible para el problema bajo determinados objetivos y ciertas restricciones. Ambos procesos tratan de generar lo mejor u óptimo. Alsewari and Zamli (2012) proponen y evalúan una estrategia denominada *Strategy of Pairwise Harmony Search* (PHSS) para generar datos de prueba en parejas. PHSS se evalúa en dos partes. En la primera parte, se toma una configuración del sistema con 10 parámetros de entrada de valores v, donde v varió de 3 a 10. Se tomó otra configuración del sistema con p parámetros de entrada de 2 valores, donde p osciló de 3 a 15. El objetivo fue examinar cómo se comportó PHSS en relación con la variación de v y p. Los resultados mostraron que su desempeño no fue afectado por el creciente número de v y p. Además, en la mayoría de los casos, el algoritmo generó tamaños de conjuntos de prueba más pequeños que otras estrategias, como PPSTG, IPOG, TConfig, Jenny y TVG, entre otros. En una segunda parte,

se generaron otras conformaciones del sistema para comparar el desempeño del PHSS con otras estrategias representativas de la literatura, como SA, GA y ACA. Los resultados de PHSS fueron competitivos.

Alsewari and Zamli (2012) diseñan, implementan y evalúan un algoritmo basado en HS para la fuerza variable llamado Harmony Search Strategy (HSS), que consta de dos algoritmos principales. El primero es un algoritmo generador de interacciones que genera parámetros, tuplas y valores de interacción basados en una intensidad especificada, así como en una lista de restricciones. El segundo es un algoritmo para generar casos de prueba en los que se especifican el tamaño de la memoria de armonía, la tasa de consideración de la memoria de armonía, la tasa de ajuste del tono y los criterios de parada. Los resultados del algoritmo se evaluaron en dos partes: en una parte inicial, se evaluó el desempeño de HSS en comparación con otras estrategias de fuerza variable, incluyendo VS-PSTG, ACS, SA e IPOG. Dependiendo de la resistencia, los parámetros y los valores definidos, HSS generó los resultados más óptimos para muy altas resistencias, ya que es capaz de manejar resistencias de interacción de hasta t = 15. HSS superó a otros algoritmos como ACS y SA que generan casos de prueba con la interacción La fuerza t menor o igual a 3 y la VS-PSTG que genera casos de prueba con una resistencia a la interacción t menor o igual a 6. Sin embargo, SA generó los resultados más óptimos con valores bajos de interacción t menor o igual a 3. HSS, VS-PSTG y ACS obtenidos iguales o cercanos a los de SA. En una segunda parte, el HSS fue comparado con otras estrategias que manejan restricciones, como.

Sin embargo, estas dos últimas propuestas a pesar de ser capaces de manejar restricciones, no tienen éxito generando pruebas de fuerza variable. Por lo tanto, no fueron considerados en los experimentos de fuerza variable. PICT reportó los peores resultados. Finalmente, los resultados obtenidos por el HSS fueron competitivos con los producidos por SA_SAT, ya que en la mayoría de los casos fue capaz de igualar los resultados, y solo en algunas congestiones logró superarlos. Bao et al (2015) presenta el algoritmo *Improvement Harmony Search* (IHS), una propuesta que busca mejorar la velocidad de convergencia del algoritmo HS estándar. IHS utiliza un algoritmo Greedy para generar un conjunto óptimo de soluciones iniciales para inicializar la memoria de armonía. Para evitar que el algoritmo caiga en óptima local, los valores de HMCR y PAR se ajustaron

dinámicamente. Los resultados de los experimentos mostraron que el tamaño de los casos de prueba generados por IHS es menor que los generados por HS. Se demostró además que cuando la fuerza t es pequeña, el tiempo de ejecución de IHS es muy similar al de HS. Sin embargo, a medida que t aumenta, el tiempo de ejecución de IHS disminuye significativamente en comparación con HS. Al comparar IHS con otros algoritmos inteligentes como GA, ACA y SA, se pudo ver que en la mayoría de los experimentos los tamaños de los casos de prueba IHS fueron los más óptimos.

2.3 Conclusiones del estado del arte

Un buen método para construir CA y/o MCA es relevante si encuentra un equilibrio apropiado entre el tiempo para construirlo y la calidad de la matriz resultado, esta última medida en el número de filas (valor N).

Los métodos algebraicos a menudo proporcionan buenos resultados para la construcción de CA, sin embargo solo son aplicables a casos muy específicos y generalmente pequeños. Si se llegan a utilizar para encontrar grandes CA, se requiere un largo tiempo de ejecución y más recursos computacionales. Los métodos voraces son más flexibles que los métodos algebraicos; sin embargo, rara vez obtienen CA óptimos. Los avances recientes en métodos exactos, específicamente en el uso de coeficientes binomiales y ramificación, establecen una estrategia eficiente para construir CA de fuerza $t = 2$ e indican un camino de investigación prometedor en el uso de coeficientes trinomiales para obtener CA de fuerza $t = 3$.

Los métodos basados en metaheurísticas son los más utilizados recientemente y generan mejores resultados en la construcción de CA y MCA, sin embargo, requieren mucho tiempo computacional.

El método metaheurístico más exitoso reportado hasta la fecha para la construcción de CA y MCA es *Simulated Annealing*. Las principales razones son su capacidad para escapar del óptimo local gracias a la aceptación condicional de movimientos que no necesariamente mejoran la solución actual y su capacidad para evolucionar rápidamente a mejores regiones del espacio de búsqueda que permite una mezcla de diferentes funciones del vecindario.

En este sentido, se considera que en los próximos años, las siguientes investigaciones de métodos para la construcción de CA y MCA estarán basados en metaheurísticas como el *Simulated Annealing* o en metaherísticas aún no aplicadas a esta área como el *Bacterial Foraging Optimization* (BFO) que se explica en el capítulo 3.

3. Algoritmo propuesto: Bacterial Foraging Optimization

La explosión combinatoria es el problema donde la cantidad de combinaciones que se deben examinar crece exponencialmente, tan rápido que incluso las computadoras más rápidas requerirán una cantidad intolerable de tiempo para examinarlas. Esto se debe a que en la mayoría de los problemas realistas que nos interesan, tienen una cantidad de combinaciones muy grande. El número de combinaciones crece exponencialmente a medida que N aumenta. Para mejorar la velocidad se necesita computadoras mucho más rápidas o algoritmos inteligentes que permitan ser altamente selectivo en qué secuencias examinar. Dado que la construcción de CA y MCA óptimos es (en general) un problema NP-Completo, una alternativa es el uso de metaheuísticas para calcularlos de manera eficiente. En esta investigación se propone la metaheurística BFO para construir CA y MCA de fuerzas 2 a 6, que tiene como principal característica el uso de la distancia de Hamming para cubrir todas las combinaciones faltantes.

3.1 Panorama de las metaheurísticas

Encontrar soluciones óptimas resulta difícil para muchos problemas de optimización de mayor importancia industrial y científica (Talbi, 2009). En un problema de optimización, además de las condiciones que deben cumplir las soluciones factibles del problema, se busca la que es óptima según algún criterio de comparación entre ellas. En la práctica, se suele estar satisfecho con "buenas" soluciones, que se obtienen por medio de algoritmos heurísticos o metaheurísticas. A diferencia de los algoritmos de optimización exactos, las heurísticas y metaheurísticas no garantizan la optimalidad de las soluciones obtenidas.

El término metaheurística se introdujo por Glover (1986). La palabra heurística tiene su origen en la antigua palabra griega *heuriskein*, lo que significa el arte de descubrir nuevas estrategias (reglas)

para resolver problemas. El sufijo *meta*, también una palabra griega, significa "metodología de nivel superior". Las metaheurísticas representan una familia de técnicas de optimización aproximadas que proporcionan soluciones "aceptables" en un tiempo razonable para la solución de problemas difíciles y complejos en ciencia e ingeniería.

Las metaheurísticas se pueden describir como estrategias aplicadas a procesos de búsqueda donde todas las situaciones intermedias en el proceso de resolución del problema se interpretan como elementos de un espacio de búsqueda, que se van modificando a medida que se aplican las distintas operaciones diseñadas para llegar a la resolución definitiva. Tales procesos se conocen como búsquedas monótonas, algoritmos escaladores o búsquedas locales. Esta última denominación obedece a que la mejora se obtiene con base en el análisis de soluciones similares a la que realiza la búsqueda, denominadas soluciones vecinas.

3.1.1 Conceptos básicos

- **Heurístico:** Se califica de heurístico a un procedimiento para el que se tiene un alto grado de confianza en que encuentra soluciones de alta calidad con un costo computacional razonable, aunque no se garantice su optimalidad.
- **Metaheurísticas:** Las metaheurísticas guían los procedimientos que usan transformaciones o movimientos para recorrer el espacio de soluciones alternativas y explotar las estructuras de entorno asociadas.
- **Búsqueda local:** Basa su estrategia en el estudio de soluciones del vecindario o entorno de la solución que realiza el recorrido.
- **Función de vecindad:** Una función de vecindad N es un mapeo $N : S \rightarrow 2^S$ que asigna a cada solución s de S un conjunto de soluciones $N(s) \subset S$.
- **Óptimo local:** Relativo a una función de vecindad N dada, una solución $s \in S$ es un óptimo local si tiene una mejor calidad que todos sus vecinos; esto es, en caso de un problema de minimización, $f(s) <= f(s')\forall s' \in N(s)$.
- **Algoritmo voraz:** Un algoritmo voraz siempre hace la mejor elección en cada iteración, i.e., hace una elección óptima localmente esperando que ésta conduzca a una solución global óptima.

Las metaheurísticas aplican iterativamente procedimientos de generación y reemplazo:

- En la fase de generación, un conjunto de soluciones candidatas se generan a partir de la solución actual s. Este conjunto $C(s)$ se obtiene generalmente por transformaciones locales de la solución.
- En la fase de reemplazo, una solución $s' \in C(s)$ se selecciona para ser la nueva solución, su elección se basa generalmente por los resultados obtenidos por la función de evaluación $C(s')$ (Hernández, 2013).

Este proceso itera hasta que un criterio de terminación se cumple (véase Algoritmo 1).

Algorithm 1 Bosquejo general de un algoritmo metaheurístico

Input: A initial solution s_0
1: $i = 0$
2: **for** termination criteria = true **do**
3: /* Generate candidate solutions of s_i */
4: Generate $(C(s_i))$
5: /* Select a solution of $C(s)$ and replace the current solution s_i */
6: $s_i + 1$ =Select $(C(s_i))$
7: $i = i + 1$
8: **end for**
Output: The best solution s_*

3.2 Revisión de BFO

La metaheurística BFO (*Bacterial Foraging Optimization*, por sus siglas en inglés) como lo indica Brownlee (2015), pertenece al campo de optimización de las bacterias y optimización de enjambre. En este trabajo de investigación se adapta e implementa BFO debido al estudio del Estado del Arte donde se verifica que no se encuentra implementada para resolver el problema de la construcción de CA y/o MCA.

BFO se inspira en el comportamiento de la quimiotaxis del grupo de bacterias tales como E.coli y M.xanthus; las bacterias al percibir gradientes (nutrientes) químicos en el medio ambiente, se mueven acercándose o alejándose de señales específicas. Permite reducir el coste de las bacterias mediante la proximidad a otras bacterias; se mueven a lo largo de la superficie de costo manipulado una a la vez. Las bacterias son tratadas como agentes en un entorno, dentro del cual se mueven de

manera caótica y otras veces de manera dirigida. La bacteria tiende a percibir la dirección de la comida y trata de escalar hacia los gradientes del entorno (bajos o altos) tomando en cuenta efectos combinados de señales de atracción y repulsión (función de interacción bacteria-bacteria). Esta estrategia del algoritmo de manera colectiva y estocástica logra un tipo de optimización al proveer a las bacterias de un comportamiento (movimientos) que les permite buscar valores altos o bajos (pudiendo enjambrar en ellos), evadir valores neutrales o en su caso encontrar la forma de salir de ellos (Passino, 2002).

La interacción bacteria-bacteria permite reducir el costo de la bacteria y además aporta un comportamiento de enjambre; es decir, las bacterias se perciben unas a otras y se organizan en grupos para dirigirse y abundar en altas concentraciones de nutrientes. A continuación se presenta la ecuación para la función de interacción, g:

$$g(cell_k) = \sum_{i=1}^{S} \left[-d_{attr} \times exp\left(-w_{attr} \times \sum_{m=1}^{P} \left(cell_m^k - other_m^i \right)^2 \right) \right] + \sum_{i=1}^{S} \left[h_{repel} \times exp\left(-w_{repel} \times \sum_{m=1}^{P} \left(cell_m^k - other_m^i \right)^2 \right) \right] \tag{3.1}$$

Donde $cell_k$ es una bacteria dada, d_{attr} y w_{attr} son coeficientes de atracción, h_{repel} y w_{repel} son coeficientes de repulsión, S es el número de bacterias en la población y P es el número de dimensiones en un vector de posiciones de bacterias dado.

3.2.1 Pseudocódigo

El pseudocódigo del Algoritmo 2 representa el movimiento clásico de las bacterias en BFO; es decir, el comportamiento quimiotáctico de nadado y giro de las bacterias. Este implica los parámetros restantes del algoritmo: *Population* son las posiciones de las bacterias, $Cells_{\text{num}}$ es la cantidad de bacterias en la población, N_c es el número de Quimiotaxis, N_s es el número de pasos de natación para una célula dada y $Step_{\text{size}}$ es un vector de dirección con el mismo número de dimensiones que el espacio de búsqueda del problema.

Algorithm 2 Pseudocodigo para la funcion de quimiotaxis y nadado de BFO

Input: $Population, Problem_{\text{size}}, Cells_{\text{num}}, N_s, Step_{\text{size}}, d_{\text{attract}}, w_{\text{attract}}, h_{\text{repellant}}, w_{\text{repellant}}$

1: **for** $(Cell \in Population)$ **do**
2: $Cell_{\text{fitness}} \leftarrow Cost(Cell) + interaction(Cell, Population, d_{\text{attract}}, w_{\text{attract}}, h_{\text{repellant}}, w_{\text{repellant}})$
3: $Cell_{\text{health}} \leftarrow Cell_{\text{fitness}}$
4: $Cell' \leftarrow \emptyset$
5: **for** $i \leftarrow 1, N_s$ **do**
6: $RandomStepDirection \leftarrow CreateStep(Problem_{\text{size}})$
7: $Cell'_{\text{fitness}} \leftarrow Cost(Cell') + interaction(Cell', Population, d_{\text{attract}}, w_{\text{attract}}, h_{\text{repellant}}, w_{\text{repellant}})$
8: **if** $(Cell'_{\text{fitness}} > Cell_{\text{fitness}})$ **then**
9: $i \leftarrow N_s$
10: **else**
11: $Cell \leftarrow Cell'$
12: $Cell_{\text{health}} \leftarrow Cell_{\text{health}} + Cell'_{\text{fitness}}$
13: **end if**
14: **end for**
15: **end for**

Para la presentación computacional del problema, una matriz posee cierta cantidad de bacterias. Cada bacteria representa un lazo de búsqueda de una posible solución al problema, se ubican inicialmente al azar en todo el espacio y su posición representa una solución al problema.

Estos individuos pueden moverse en pasos del mismo tamaño o cada uno puede tener un tamaño de paso predefinido por el programador. El tamaño del paso definirá qué tan rápido se mueve la bacteria; las bacterias de paso más grande se desplazarán más rápido hacia la solución, pero pueden presentar sobrepicos más grandes que las de paso más corto. Cada bacteria varía su posición en búsqueda de distintas soluciones, avanzando hacia las direcciones donde el gradiente de nutrientes es positivo, es decir, donde la función de costo se reduce.

Transcurrido determinado número de ciclos, el algoritmo puede dejar morir las bacterias que se encuentran en posiciones con menor alimento y reproducir las bacterias ubicadas en sitios con mayor cantidad de nutrientes. Por lo tanto, la población tenderá a aumentar en aquellos sitios de alta concentración de alimentos y a desaparecer en aquellos sitios donde la concentración de alimentos es escasa. En otros términos, las bacterias que ocupen posiciones que tengan una función de costo elevada o que representen una peor solución al problema, serán eliminadas y reubicadas en las posiciones donde la bacteria representa una mejor solución.

Este acto se llama evento de reproducción y representa el número de pasos quemotácticos necesarios para que se produzca un evento de este tipo. Mientras menos cantidad de ciclos transcurran para producirse un evento de reproducción, con mayor rapidez serán reubicadas las bacterias en las posiciones con mejor función de costo.

Es posible también que las condiciones del medio ambiente donde vive la población de bacterias cambien gradualmente o incluso repentinamente debido a una influencia externa. Puede ocurrir por tanto, que aparezcan nuevos sitios con alimento, o que los lugares actuales donde hay alimento desaparezcan. Debido a estos cambios de condiciones, el algoritmo reubica con una probabilidad a las bacterias luego de cierta cantidad de generaciones o eventos de reproducción, asegurando que las bacterias no se queden en una solución local, sino que siempre estén en búsqueda de nuevas soluciones. Este hecho se llama evento de eliminación y dispersión.

En sí, el algoritmo determinará la dirección del paso de cada una de las bacterias de la población total en búsqueda de gradientes de nutrientes que permitan mejorar la función de costo.

3.2.2 Función objetivo

Para que un método de optimización o estimación funcione, es necesaria una función de interacción que se da por la sumatoria del amontonamiento de bacterias y una función de costo que refleje cuán cercana o cuán lejana se encuentra la búsqueda del valor óptimo.

3.3 Algoritmo propuesto: BFO para la construcción de CA y MCA

3.3.1 Representación y estrategia de solución

Sea s una solución potencial en el espacio de búsqueda S, el cual se deriva a partir de la instancia especificada por un MCA($N;t,k,v_1,v_2...v_k$) con N filas, k columnas y fuerza t. Sea V el vector de cardinalidades $v_1,v_2...v_k$ que indica la cardinalidad de cada columna. Una solución s es representada como una matriz M de tamaño $N \times k$, en la cual el elemento $m_{i,j}$ denota el valor asignado a cada conjunto de prueba $r_i(1 <= i <= N)$ del parámetro $p_j(1 <= j <= k)$. La cardinalidad del conjunto de todas posibles soluciones está dada por la siguiente ecuación:

$$|S| = \binom{\alpha}{N}$$

donde α, se calcula como:

$\alpha = \Pi_{j=1}^{k} v_j$

En la metaheurística BFO clásica cada bacteria representa una solución completa, pero en este trabajo de investigación se define que cada bacteria se representa por una fila i de la matriz M, es decir, cada bacteria representa una parte de la solución. Se selecciona esta representación porque optimiza los recursos computacionales para construir una solución CA ó MCA.

3.3.2 Función objetivo

La función objetivo permite que los vacíos se cubran de manera óptima, está compuesta por la función de costo y la función de interacción que se describen a continuación:

Función de costo

La función de costo de BFO propone la distancia de Hamming para la repulsión entre dos baterias, es decir, entre cada par de renglones se suman las veces que difieren sus elementos, su función es la siguiente:

$$Hamming = \sum_{i=1}^{k} d(r_1, r_2)$$

La distancia de Hamming cuenta con estas dos propiedades:

$d(a,b) = d(b,a)$

$d(a,b) = 0$ si y sólo si $a = b$

La distancia de Hamming se calcula entre una fila contenida previamente en la matriz M y una fila candidata, este cálculo se realiza para definir cuál es la fila candidata que mayor combinaciones

faltantes cubre, es decir, se selecciona la fila candidata que obtenga mayor distancia de Hamming.

La motivación de usar distancia de Hamming para calcular la repulsión de bacterias se basa en la observación de cuando dos filas cuya distancia de Hamming sea 0 significa que estas filas cubren exactamente los mismos espacios en el CA ó MCA. Por ejemplo, en la Tabla 3.1, la distancia de Hamming es $= 0+0+0+0+0+0+0+0+0+0$, o sea 0, cuando se usan dos filas que son completamente iguales.

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Tabla 3.1: Ejemplo de distancia de Hamming para dos filas iguales.

Ahora, en el caso cuando la distancia de Hamming entre dos filas es máxima, entonces se dice que las filas cubren distintas zonas del CA ó MCA. Por ejemplo, en la Tabla 3.2 la distancia de Hamming es $= 1+1+1+1+1+1+1+1+1+1$, o sea 10, cuando se usan dos filas completamente distintas:

0	2	1	1	1	0	2	0	1	1
1	1	0	0	2	1	1	1	0	2
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Tabla 3.2: Ejemplo de distancia de Hamming para dos filas diferentes.

Función de interacción:

Se refiere al amontonamiento de bacterias, es decir, es la sumatoria de todas las bacterias que cubren el mismo espacio. De forma análoga a la ecuación 3.1, se usa:

$$g(\text{cell}_k) = \sum_{i=1}^{S} h_{\text{repel}} \exp\left(-w_{\text{repel}} \cdot D_{ik}\right)$$

donde D_{ik}, es el número de t-adas iguales entre las celdas cell_i y cell_k, y también se puede calcular como:

$$D_{ik} = \begin{cases} \binom{k-\text{Hamm}_{ik}}{t} & \text{si } k - \text{Hamm}_{ik} \geq t \\ 0 & \text{en caso contrario} \end{cases} \tag{3.2}$$

Así, la función de repulsión total se calcula como:

$$g = \sum_{k=1}^{S} g(\text{cell}_k)$$

3.3.3 Funciones de vecindad

Una función de vecindad es un conjunto de movimientos que permite la generación de nuevas soluciones s' a partir de una solución actual s. La solución s' es llamado un vecino de s. Siempre que un movimiento llevado a cabo por una función de vecindad sea aleatorio, el conjunto de vecinos derivados de s son llamados la vecindad y denotados por $N(s)$. Cuando más de un vecino es creado, se usa una función objetivo o de evaluación que calcula su costo, para decidir cual vecino será seleccionado como la nueva solución s'.

Para mejorar el rendimiento de los algoritmos que resuelven los problemas de optimización, algunos trabajos previos han implementado el uso de diferentes funciones de vecindad (Avanthay et al, 2003), (Mladenović and Hansen, 1997), (Rodriguez-Tello et al, 2008). Basados en esta premisa, el diseño de BFO incorpora dos funciones de vecindad con la posibilidad de una mezcla entre ellas a través de una probabilidad de uso.

Función de vecindad: OneRow

El principal movimiento de la función OneRow selecciona una única prueba a la vez, es decir, una fila a la vez. Este proceso se repite hasta que se cubre el mayor número de interacciones y se agregan a la matriz M, una vez que todas las interacciones están cubiertas, se dice que el CA o MCA está construido.

Es un método ambicioso y de ninguna manera por sí solo garantiza construir un CA y/o MCA de tamaño mínimo. Sin embargo, sí garantiza que en cada etapa, al menos $|P|/L$ pares nuevos estén cubiertos donde L es el producto de los dos tamaños más grandes de vocabulario v.

Algorithm 3 Función de vecindad: OneRow

Input: $Population, Problem_{\text{size}}$

```
1:  minArray ← ∞
2:  for (i ← N) do
3:      MCA ← [][]
4:      while (t-tuplas > 0) do
5:          for (i ← N) do
6:              rowTest ← []
7:              Best ← 0
8:              while (factors = free) do
9:                  rank all free factors according to a factor selection criterion
10:                 among factors tied for best, select a subset T using a first factor tie-break
11:                 among factors in T, select a single factor f using a second factor tie-break
12:                 all possible values for f in rowTest using a level selection criterion
13:                 among all best values for f, select a subset V using a first level tie-break
14:                 among values in V, select value v using a second level tie-break
15:                 fix factor f to value v in rowTest
16:             end while
17:             if ( thenR covers alpha> Best t -tuples uncovered in C),
18:                 set Best = alpha , B = rowTest
19:             end if
20:         end for
21:         add row B to C
22:     end while
23:     if ( thenC has Size < MinArray rows)
24:         set MinArray = Size and Best Array = C
25:     end if
26: end for
27: report Best Array
```

Esto, a su vez, asegura que el tamaño del conjunto de pruebas construido está limitado por una función logarítmica del número k de factores.

Para crear una instancia de este método, se deben tomar una serie de decisiones que se muestran en el algoritmo 2. Los algoritmos codiciosos para la construcción de CA o MCA tienen cuatro puntos de decisión principales para definir. Éstas incluyen:

1. el número de repeticiones de matriz de cobertura;

2. la cantidad de candidatos a casos de prueba para generar;
3. factores de heurística de ordenamiento (incluido el desempate); y
4. heurística de sección de nivel (incluido el desempate).

Función de vecindad: OneCol

Esta función selecciona aleatoriamente una columna $j(1 <= j <= k)$ de la matriz y evalúa todos los cambios de símbolos de cada una de las celdas involucradas en la columna, i.e. $\forall m_{i,j} \in M$ donde el renglón $i(1 <= i <= N)$, se realizan todos los cambios de símbolos y se calcula el valor de F para cada vecino.

Algorithm 4 Función de vecindad: OneCol

Input: $Population, Problem_{\text{size}}$

```
1:  minArray ← ∞
2:  for (i ← N) do
3:      MCA ← [][]
4:      while (t-tuplas > 0) do
5:          for (i ← N) do
6:              colTest ← []
7:              Best ← 0
8:              while (factors = free) do
9:                  rank all free factors according to a factor selection criterion
10:                 among factors tied for best, select a subset T using a first factor tie-break
11:                 among factors in T, select a single factor f using a second factor tie-break
12:                 all possible values for f in rowTest using a level selection criterion
13:                 among all best values for f, select a subset V using a first level tie-break
14:                 among values in V, select value v using a second level tie-break
15:                 fix factor f to value v in rowTest
16:             end while
17:             if ( thenR covers alpha> Best t -tuples uncovered in C),
18:                 set Best = alpha , B = rowTest
19:             end if
20:         end for
21:         add row B to C
22:     end while
23:     if ( thenC has Size < MinArray rows)
24:         set MinArray = Size and Best Array = C
25:     end if
26: end for
27: report Best Array
```

El cambio de símbolo que genere el menor valor de F es seleccionado para crear la nueva

solución s'. Debido a que existen $v_j - 1$ posibles cambios de símbolos en la celda de la j-ésima columna y N filas en la columna, el esfuerzo computacional requerido para la evaluación es:

$$O(2N[(v_j - 1)\binom{t-1}{k-1}])$$

En la Figura 3.1 se muestra un ejemplo de OneCol después de seleccionar aleatoriamente la segunda columna $j = 2$. Se presenta la solución actual s del MCA$(6;2,3,3^1 2^2)$ y los posibles cambios de símbolos generados por esta función de vecindad.

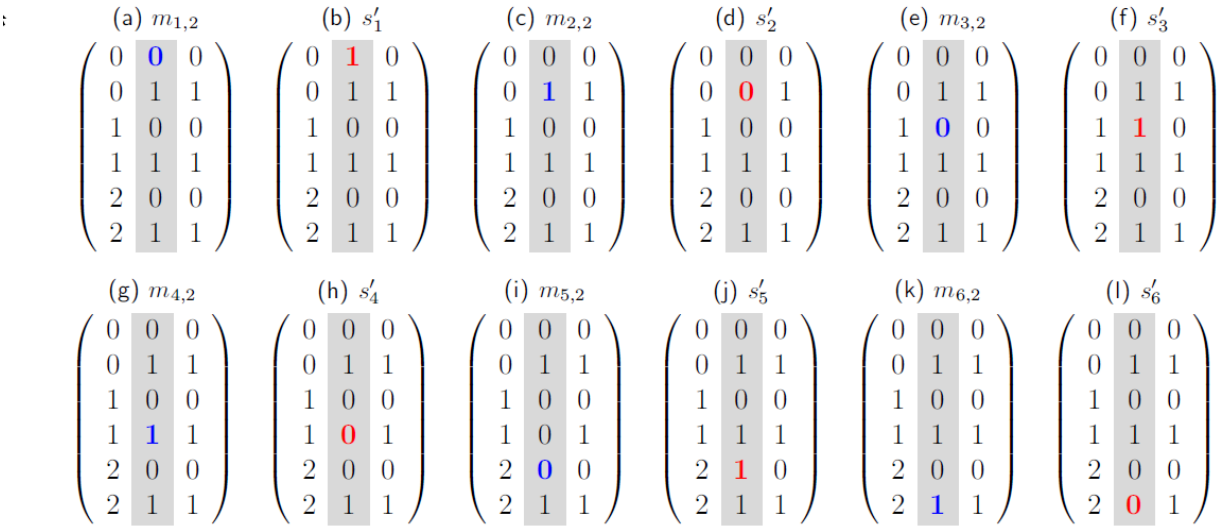

Figura 3.1: OneCol para la columna $j = 2$.

Los cambios que se realizan en este ejemplo se describen en los siguientes incisos:

a) Símbolo actual en $m_{1,2} = 0$.

b) Vecino generado por OneCol en la posición $m_{1,2} = 1$.

c) Símbolo actual en $m_{2,2} = 1$.

d) Vecino generado por OneCol en la posición $m_{2,2} = 0$.

e) Símbolo actual en $m_{3,2} = 0$.

f) Vecino generado por OneCol en la posición $m_{3,2} = 1$.

g) Símbolo actual en $m_{4,2} = 1$.

h) Vecino generado por OneCol en la posición $m_{4,2} = 0$.

i) Símbolo actual en $m_{5,2} = 0$.

j) Vecino generado por OneCol en la posición $m_{5,2} = 1$.

k) Símbolo actual en $m_{6,2} = 1$.

l) Vecino generado por OneCol en la posición $m_{6,2} = 0$.

3.3.4 Criterios de terminación

Un criterio de terminación, le indica al algoritmo el momento en que debe detener su ejecución. En BFO, dos criterios de terminación han sido incorporados:

1. $\mathscr{E}$: Número de Evaluaciones

El criterio de terminación $\mathscr{E}$ indica que el algoritmo debe detenerse cuando el número de llamadas e a la función de evaluación $C(s)$ iguala o excede al indicado, i.e. **Si** $e >= \mathscr{E}$ **entonces** se deberá detener la ejecución. Este criterio es utilizado en la etapa de sintonización de los experimentos desarrollados. Los valores que han sido propuestos para el número de evaluaciones $\mathscr{E}$ se muestran en la tabla 3.3, $v_{max} = \Pi_{j=1}^{t} v_j$ (donde v_j es la j-ésima cardinalidad del alfabeto en orden decreciente).

Codificación	Número de evaluaciones
$\mathscr{E}_1$	$N * k * v_{max^t} * 100$
$\mathscr{E}_2$	$N * k * v_{max^t} * 150$
$\mathscr{E}_3$	$N * k * v_{max^t} * 200$

Tabla 3.3: Número de evaluaciones $\mathscr{E}$ usadas como criterio de terminación en BFO.

Debido a la etapa de sintonización cada instancia es resuelta 31 veces por cada configuración de parámetros, es importante que BFO no itere indefinidamente, i.e. que termine la ejecución en un número máximo de evaluaciones para los casos donde no puede construir el CA ó MCA. Posterior a las 31 ejecuciones correspondientes a una configuración, se obtiene la tasa de éxito (hits), i.e. solo se cuantifica el número de construcciones exitosas sobre las 31 corridas, considerando que no todas las configuraciones logran construir el CA o MCA en el tiempo especificado por el valor de $\mathscr{E}$.

Una corrida se considera exitosa cuando se logra la construcción del CA ó MCA (**Si** $F = 0$).

2. $\mathscr{F}$: Número de Combinaciones Faltantes

El criterio de terminación correspondiente al número de combinaciones faltantes $\mathscr{F}$, indica que el algoritmo debe detener su ejecución cuando la función de evaluación para la mejor solución encontrada, es decir el CA o MCA optimizado, obtiene un número menor o igual de combinaciones faltantes, i.e. . Cuando el objetivo de la ejecución es construir una instancia MCA($N;t,k,v_1,v_2...v_k$) se asigna el valor $\mathscr{F} = 0$. Este criterio es utilizado en las etapas de evaluación del rendimiento de BFO dentro de los experimentos desarrollados.

3.4 Resumen del capítulo

En este capítulo se detalló la metodología de solución propuesta para la construcción de CA y MCA de fuerzas $t >= 2$ a $t <= 6$, la cual se enfoca en el diseño y desarrollo de un algoritmo que utiliza la metaheurística Bacterial Foraging Optimization (BFO) teniendo como principal característica la representación de cada prueba de software como una bacteria y la distancia de Hamming como función de interacción entre las bacterias.

Al inicio del capítulo se puntualizaron los aspectos generales de la metaheurística BFO clásica, para en lo sucesivo enfatizar las características que posee el método propuesto.

Se resaltó que el hecho de contar con diversas opciones de los parámetros OneRor y OneCol, atribuye a BFO la posibilidad de ser ejecutado bajo una diversa gama de configuraciones, entendiendo como una configuración, al valor establecido para OneRow y OneCol para realizar una ejecución.

Adicionalmente, se describió la forma de representación para la solución, las funciones de evaluación y los criterios de terminación que fueron utilizados en las etapas de los experimentos desarrollados.

4. Experimentos y resultados

En este capítulo se describe la metodología seguida para evaluar la propuesta de solución. Posteriormente, se brinda un esquema general del diseño experimental, incluyendo la especificación de las características de la plataforma de experimentación utilizada. Asimismo, se presentan los resultados obtenidos por el algoritmo BFO derivados de la etapa de sintonización y la de evaluación de su rendimiento. Finalmente, para corroborar el cumplimiento del objetivo general de esta tesis, se realiza un análisis de comparación contra las mejores reportadas en un benchmark que incluye casos reales y casos sintéticos. Los CA y MCA incluidos en este benchmark abarcan fuerzas 2 a 6.

4.1 Metodología para evaluar la propuesta de solución

1. Diseño experimental: Se deben definir los objetivos del experimento, la selección de las instancias y la identificación de los principales parámetros de la metaheurística.

2. Medición: En este paso, se establecen las métricas de comparación y, después de realizar los diferentes experimentos, se debe llevar a cabo un análisis estadístico sobre los resultados obtenidos. Los contrastes deben ser efectuados con otras estrategias del estado del arte enfocadas a la solución del problema al cual se enfoca la metaheurística implementada.

3. Reporte: En este paso, se presentan los resultados en forma comparativa, asimismo, se debe realizar un análisis de los datos basados en los objetivos definidos.

4.1.1 Diseño experimental

Dentro del diseño experimental, se deben definir claramente los objetivos, ya que las métricas que se van a reportar y el análisis estadístico dependerán del propósito del diseño de la metaheurística. Entre los criterios que pueden ser considerados, en términos de la instancia, se encuentran el tiempo de búsqueda, la calidad de solución y robustez.

Sintonización de parámetros

Una vez que los principales parámetros de la metaheurística identificados, se debe seleccionar un conjunto de valores de los parámetros para realizar una sintonización. Estos parámetros pueden tener una gran influencia sobre la eficiencia y efectividad de la búsqueda de la solución, por esta razón es importante establecer diferentes niveles de valores para cada parámetro.

Algo que debe tenerse en cuenta es que los valores óptimos para los parámetros dependen primordialmente del problema e incluso la instancia a resolver. Dado que no existe un conjunto de valores óptimo para los parámetros establecidos de forma universal para una metaheurística dada, es importante seleccionar de forma adecuada las instancias que se utilizarán durante la sintonización.

La sintonización se realiza combinando los parámetros OneRow y OneCol desde 0 hasta 100 por ciento, como se muestra en la Tabla 4.1:

Combinación de parámetros	CA1	CA2	CA3	CA4	CA5	CA6	CA7	CA8	CA9	Prom
100 % OneCol	11	8	8	10	9	11	5	7	3	6.82
100 % OneRow	8	5	2	4	11	10	1	10	11	3.66
90 % OneCol 10 % OneRow	10	10	3	2	4	9	3	6	1	3.10
80 % OneCol 20 % OneRow	9	9	4	3	2	1	2	4	4	3.10
70 % OneCol 30 % OneRow	**1**	**2**	**1**	**1**	**1**	**2**	**4**	**1**	**2**	**1.33**
60 % OneCol 40 % OneRow	7	3	5	6	8	3	6	3	5	4.49
50 % OneCol 50 % OneRow	6	4	6	7	6	4	9	8	6	5.82
40 % OneCol 60 % OneRow	4	1	11	5	10	7	7	5	10	4.04
30 % OneCol 70 % OneRow	2	7	10	11	5	6	8	2	9	4.64
20 % OneCol 80 % OneRow	5	6	9	9	3	5	10	9	8	6.17
10 % OneCol 90 % OneRow	3	11	7	8	7	8	11	11	7	7.00

Tabla 4.1: Tabla de sintonización de parámetros de BFO.

Dónde:

CA1 = Bugzilla

CA2 = S3-8

CA3 = Healthcare4

CA4 = S3-3

CA5 = Mobile Phone

CA6 = Gzip

CA7 = RFID

CA8 = S2-4

CA9 = TCAS

Para cada caso de CA se ejecutan 31 veces cada una de las combinaciones de parámetros de BFO y se calcula el promedio, los números del 1 al 11 indican el lugar en el que se encuentra el promedio de dicha combinación, siendo 1 el mejor y 11 el peor. Después se vuelve a calcular el promedio de los 9 casos para cada combinación, donde la combinación ganadora fue **70 % OneCol 30 % OneRow** con promedio de **1.33**, siendo ésta la combinación de parámetros que se usa para los experimentos.

4.1.2 Medición

En este paso, se seleccionan las métricas para evaluar el rendimiento y los indicadores para los cálculos. Después de ejecutar los experimentos, se pueden aplicar diferentes análisis estadísticos sobre otros resultados. Para efectuar la medición, los indicadores para una metaheurística pueden ser clasificados en tres grupos: calidad de solución, esfuerzo computacional y robustez.

La construcción de CA y MCA no es una tarea fácil ya que se necesita mucho tiempo de ejecución y una gran capacidad computacional. Por lo que el experimento se realizó en el clúster de CICESE, Unidad Tepic, Nayarit. Las características físicas del equipo son las siguientes:

- Procesador AMD Radeon 2.20 GHz
- 12 GB de RAM

El objetivo de realizar estos experimentos es analizar el rendimiento del algoritmo BFO y compararlo con otros algoritmos propuestos del estado del arte.

4.1.3 Resultados

Durante la etapa de evaluación del rendimiento de BFO se utilizó un benchmark formado por dos conjuntos. El primero es un conjunto de casos reales de componentes de software, el cual estuvo constituido por 35 instancias. El segundo conjunto, está formado por instancias comúnmente utilizadas para evaluar el rendimiento de otras estrategias que construyen CA y MCA. Este benchmark abarcó un total de 93 instancias. Ambos conjuntos utilizados en el benchmark abarcaron instancias

desde fuerzas 2 hasta 6,las cuales incluían los mejores casos reportados en la literatura.

- **Instancias de casos reales**

Éstas representan instancias prácticas del problema a resolver y constituyen un buen benchmark para llevar a cabo la evaluación del rendimiento de BFO con las demás metaheurísticas mejor reportadas en la literatura.

La Tabla 4.2 desglosa el benchmark de casos reales de componentes de software. La primer columna indica el ID de la instancia, la columna 2 muestra el nombre del componente de software y su respectiva configuración. Las columnas 3 y 4 corresponden a los valores de k y la fuerza t correspondientes a cada configuración. La columna 5, encabezada con θ, corresponde a la cota mínima teórica para la instancia indicada. La columna 6 despliega el valor de N del mejor reportado en la literatura. Se utiliza el símbolo * para indicar aquellos que son el óptimo. La columna 7 contiene el nombre de la estrategia que construyó el mejor reportado y la referencia de dicha publicación.

La columna 8 muestra el mínimo valor de N generado por BFO, remarcando en negritas donde se han generado nuevas cotas. Finalmente, la última columna contiene el resultado de la diferencia entre el resultado obtenido por BFO y el **N*** mejor reportado.

- **Instancias de casos sintéticos**

Muchas librerías de instancias "standard" están disponibles, las cuales contienen instancias comúnmente utilizadas para optimización global, optimización combinatoria y programas mixtos. Evaluar el rendimiento de una metaheurística utilizando sólo instancias académicas puede ser controversial, ya que la efectividad de la metaheurística podría ser completamente diferente en la práctica.

En la tabla 4.3 se muestran los resultados experimentales del método propuesto para las instancias de los casos sintéticos.

En este paso, se establecen las métricas de comparación y, después de realizar los diferentes experimentos, se debe llevar a cabo un análisis estadístico sobre los resultados obtenidos. Los contrastes deben ser efectuados con otras estrategias del estado del arte enfocadas a la solución del problema al cual se enfoca la metaheurística implementada.

Tabla 4.2: Resultados de BFO sobre el conjunto de instancias de casos reales:

ID	Configuración	k	t	θ	N*	Estrategia	N (BFO)	Δ
1	Mobile Phone		2	9	9*	CTE-XL	9	0
2	$3^3 2^2$		3	27	27*	IPOG	27	0
3		5	4	54	54*	IPOG	54	0
4			5	108	108*	Exhaustivo	108	0
5	Bugzilla		2	16	16*	SA	16	0
6	$4^2 3^1 2^{49}$		3	48	51	Ttuples	48*	-3
7		52	4	96	197	Ttuples	166*	-31
8			5	192	644	IPO-F	574*	-70
9	RFID Tracking System module		2	25	38	MIPOG	29*	-9
10	$5^7 2^4$		3	125	218	MIPOG	203*	-15
11		11	4	625	1154	MIPOG	1082*	-72
12			5	3125	5625	MIPOG	5035*	-590
13	SPIN simulator		2	16*	16	CASA	16	0
14	$4^5 2^{13}$		3	64*	64	MITS	64	0
15		18	4	256	256*	MITS	256	0
16			5	1024	1024*	Ttuples	1024	0
17	Android		2	25	25*	MITS	25	-9
18	$5^2 4^4 3^3$		3	100	105*	MITS	105	-15
19		9	4	400	516*	MITS	516	-72
20			5	1600	2141*	MITS	2141	-590
21			6	6400	7707*	MITS	7707	-590
22	Gzip		2	25	38	MIPOG	28*	-9
23	$10^1 9^1 2^6$		3	125	130*	MIPOG	203	-15
24		8	4	625	1154	MIPOG	650*	-72
25			5	3125	5625	MIPOG	4300*	-590
26			6	6400	7707	MITS	7700*	-590
27	Gzip		2	25	38	MIPOG [106]	29	-9
28	$10^1 9^1 2^6$		3	125	218	MIPOG	203	-15
29		8	4	625	1154	MIPOG	1082	-72
30			5	3125	5625	MIPOG	5035	-590
31	Gzip		2	25	38	MIPOG [106]	29	-9
32	$10^1 9^1 2^6$		3	125	218	MIPOG	203	-15
33		8	4	625	1154	MIPOG	1082	-72
34			5	3125	5625	MIPOG	5035	-590

4.1.4 Resumen del capítulo

Al inicio de este capítulo se describe la metodología para evaluar la propuesta de solución, la cual incluye la realización de tres pasos. En el primero se establecen los objetivos, se seleccionan las instancias de prueba y se realiza una sintonización de parámetros. En el segundo, se realiza la

Tabla 4.3: Resultados de BFO sobre el conjunto de instancias de casos sintéticos:

ID	Configuración	k	t	θ	N*	Estrategia	N (BFO)	Δ
1	S3-8		2	50	50*	LOD	50	0
2	$10^1 6^2 4^3 3^1$		3	360	360*	SA	360	0
3		7	4	750	800*	DDA	810	0
4			5	1180			1300*	0
5	S2-4		2	29	29*	IPO	29	0
6	$4^1 3^{39} 2^{35}$		3	51			62*	0
7		75	4	96			166*	0
8			5	192			574*	0
9	S2-6		2	25	38	IPO	29*	0
10	$4^{15} 3^{17} 2^{29}$		3	125	218	TTR	203*	0
11		8	4	625	1154	EXACT	1082*	0
12			5	3125	5625	EXACT	5035*	0
13	S3-8		2	9	9*	CTE-XL	9*	0
14	$3^3 2^2$		3	27	27*	IPOG	27*	0
15		5	4	54	54*	IPOG	54*	0
16			5	108	108*	Exhaustivo	108*	0
17	S2-4		2	16	16*	SA	16*	0
18	$4^2 3^1 2^{49}$		3	48	51	Ttuples	48*	0
19		52	4	96	197	Ttuples	166	0
20			5	192	644	IPO-F	574	0
21	S26		2	25	38	MIPOG	29	0
22	$5^7 2^4$		3	125	218	MIPOG	203	0
23		11	4	625	1154	MIPOG	1082	0
24			5	3125	5625	MIPOG	5035	0
25	S26		2	25	38	MIPOG	29	0
26	$5^7 2^4$		3	125	218	MIPOG	203	0
27		11	4	625	1154	MIPOG	1082	0
28			5	3125	5625	MIPOG	5035	0
29	S26		2	25	38	MIPOG	29	0
30	$5^7 2^4$		3	125	218	MIPOG	203	0
31		11	4	625	1154	MIPOG	1082	0
32			5	3125	5625	MIPOG	5035	0

experimentación y se lleva a cabo un análisis estadístico sobre los resultados obtenidos. Finalmente, el tercer paso involucra la interpretación y análisis de los resultados, tomando en cuenta los objetivos definidos y los indicadores planteados. Posteriormente se brinda un esquema general del diseño experimental de BFO.

Debido a que el objetivo general de este trabajo de investigación es adaptar e implementar un algoritmo que obtenga un buen rendimiento en la construcción de CA y MCA de fuerzas 2 a 6,

se comenta que este capítulo contempla los detalles de los experimentos para casos reales y casos sintéticos. Se precisan las características de la plataforma de experimentación, resaltando que debido al esfuerzo computacional requerido en la etapa de sintonización, la experimentación se llevó a cabo en el clúster de CICESE Unidad Tepic, Nayarit.

Se proporciona un bosquejo general de la experimentación reportada en este capítulo, la cual se divide en dos etapas. La primera es la de sintonización, derivando como resultado una configuración ganadora c* que indica los valores de los parámetros con los cuales BFO obtiene un buen rendimiento. La segunda etapa, tiene como objetivo la evaluación del rendimiento de BFO. En ésta, se realiza la comparativa de los resultados obtenidos por BFO en la construcción de diferentes instancias, en contraste con los generados por otras estrategias, incluyendo en algunos casos el mejor reportado.

5. Conclusiones y trabajo futuro

El objetivo general de este trabajo de investigación fue elaborar un algoritmo de optimización combinatoria para la construcción de mixed covering arrays de fuerzas 2 a 6, el cual fuera capaz de mejorar la calidad de solución (menor cardinalidad) de los mejores casos reportados en diferentes benchmarks. Un MCA($N;t,k,v_1,v_2...v_k$) es una matriz M de tamaño $N \times k$ donde $v_1,v_2...v_k$ es un vector de cardinalidades que indica los valores para cada columna. Cada columna j, donde $(1 <= j <= k)$ contiene sólo elementos de un conjunto $V_j, |V_j| = |v_j|$. Las filas de cada $N \times k$ submatriz cubren, al menos una vez, todas las t-tuplas de valores derivadas de las t columnas.

5.1 Resumen del desarrollo de la tesis

La propuesta de solución de este trabajo de investigación incluye el diseño e implementación de un algoritmo que utiliza la estrategia metaheurística BFO, teniendo como principal característica la mezcla de dos funciones de vecindad llamadas OneRow y OneCol.

La metodología seguida para el desarrollo y la evaluación de la propuesta de solución se describe a continuación: Inicialmente se plantea de manera formal el problema de investigación y se formulan los objetivos de esta tesis, estableciendo como propuesta de solución la elaboración de un algoritmo de optimización combinatoria capaz de mejorar la calidad de solución (menor cardinalidad) de los mejores casos reportados en diferentes benchmarks. Para seleccionar la estrategia que utilizara el algoritmo, en el Capítulo 2 se realizó el estudio del Estado del Arte. Con base en éste, se observa que dentro de las estrategias usadas para construir los mejores casos reportados no se encontraba implementada la metaheurística BFO. Después de la elección de la metaheurística BFO, en el Capítulo 3 se desribe el diseño del algoritmo, para ello, se identifican las principales características.

Con esta información, se establecieron los principales parámetros de BFO y se propusieron

diferentes alternativas para cada uno de ellos.

La identificación de los principales parámetros y valores se hizo tomando en cuenta lo descrito por Talbi (2009) en la metodología propuesta para evaluar el rendimiento de una metaheurística: "se debe seleccionar un conjunto de valores de los parámetros para realizar una sintonización, ya que éstos pueden tener una gran influencia sobre la eficiencia y efectividad de la búsqueda, asimismo, deben ser establecidos los diferentes niveles que representen los valores de los mismos". Tomando en cuenta estas consideraciones, los parámetros de BFO quedaron establecidos de la siguiente forma:

- **70 % OneCol:** Elegir una columna aleatoriamente.
- **30 % OneRow:** Elegir una celda aleatoriamente.

Finalmente, para evaluar el rendimiento de la versión completa de BFO y verificar el cumplimiento del objetivo general, en el Capítulo 4 se establece el diseño de la experimentación y se analizan los resultados obtenidos.

Para la realización de la experimentación se contemplaron dos etapas:

1. Sintonización de parámetros.
2. Evaluación del rendimiento.

De los parámetros involucrados en la sintonización se obtiene una configuración ganadora c* usada en la etapa de evaluación del rendimiento.

Durante la sintonización de parámetros se ejecutan 31 veces las 11 combinaciones de los parámetros OneRow y OneCol para 9 instancias reales y sintéticas de fuerzas desde 2 hasta 6 (Tabla 4.1), las cuales son tomadas del benchmark usado para la evaluación del rendimiento de BFO. Los criterios para determinar el rendimiento de cada configuración, en orden respectivo de prioridad, fueron la tasa de éxito (hits), el número de combinaciones faltantes (F) y el tiempo promedio invertido (T). En el capítulo 4 se explica de manera detallada el procedimiento para seleccionar la configuración ganadora c* que es la configuración que obtiene el menor rango promedio tomando en cuenta las 9 instancias.

Para la evaluación del rendimiento de BFO, se utilizó un benchmark formado por un conjunto de casos reales de componentes de software y otro de instancias sintéticas comúnmente utilizadas para evaluar el rendimiento de otras estrategias que construyen CA y/o MCA. En ambos conjuntos,

se incluyeron los mejores casos reportados en la literatura. Para el conjunto de casos reales, BFO pudo establecer 23 nuevas cotas de las 35 instancias consideradas, logrando igualar las 12 restantes. En el segundo conjunto, de las 93 instancias tomadas en cuenta, BFO establece 20 nuevas cotas e igualó el resultado de 24 de ellas. Para la verificación del cumplimiento del objetivo general de la tesis, contrastaron los resultados obtenidos por BFO con los mejores casos reportados. En ambos conjuntos, los resultados obtenidos por BFO mostraron una diferencia significativa.

5.2 Conclusiones

Los métodos exactos son los que han reportado los mejores resultados pero son viables solo para casos pequeños. Los métodos voraces se caracterizan por ser muy rápidos pero no aseguran la construcción de buenas soluciones para CA y MCA.

Los métodos basados en metaheurísticas se podría decir que están en medio de los métodos exactos y los métodos voraces porque dan buenos resultados en tiempos considerables.

El método metaheurístico más exitoso reportado hasta la fecha para la construcción de CA y MCA es ***Simulated Annealing***por las siguientes razones:

- Su capacidad para escapar del óptimo local gracias a la aceptación condicional de movimientos que no necesariamente mejoran la solución actual.
- Su capacidad para evolucionar rápidamente a mejores regiones del espacio de búsqueda.

Acerca de los configuraciones se concluye que la mezcla de los métodos OneRow y OneCol hace que se obtenga una convergencia más rápida de BFO, sin embargo no existe una diferencia significativa en el rendimiento de éste. Esta misma situación se presenta con el uso de las instancias reales y sintéticas para las fuerzas $t = 2$ hasta $t = 6$.

En la etapa de evaluación del rendimiento de BFO, se utiliza un benchmark formado por un conjunto de casos reales de componentes de software y otro formado por instancias comúnmente utilizadas para la evaluación del rendimiento de otras estrategias que construyen CA y/o MCA. El concentrado correspondiente al conjunto de casos reales se muestra en la Tabla 4.2. Los resultados obtenidos por BFO en las 35 instancias del conjunto de casos reales, correspondientes a 23 nuevas cotas y 12 empates, muestran una diferencia significativa contrastados con el mejor reportado en la

literatura.

5.3 Trabajo futuro

El curso de este trabajo de investigación puede extenderse hacia diferentes alternativas, siendo algunas de éstas las siguientes:

- Implementar otras estrategias para crear soluciones para casos donde $t > 6$.
- Implementar metaheurísticas que no se hayan implementado para construir CA y/o MCA y mejorar los mejores resultados a la fecha.
- Desarrollar una aplicación para consultar los resultados de diferentes métodos para construir CA y/o MCA.

Bibliografía

Ahmed BS, Zamli KZ (2011) A variable strength interaction test suites generation strategy using particle swarm optimization. Journal of Systems and Software 84(12):2171–2185

Ahmed BS, Zamli KZ, Lim C (2011) The development of a particle swarm based optimization strategy for pairwise testing. Journal of Artificial Intelligence 4(2):156–165

Ahmed BS, Zamli KZ, Lim CP (2012) Application of particle swarm optimization to uniform and variable strength covering array construction. Applied Soft Computing 12(4):1330–1347

Alsewari ARA, Zamli KZ (2012) A harmony search based pairwise sampling strategy for combinatorial testing. International Journal of Physical Sciences 7(7):1062–1072

Avanthay C, Hertz A, Zufferey N (2003) A variable neighborhood search for graph coloring. European Journal of Operational Research 151(2):379–388

Bao X, Liu S, Zhang N, Dong M (2015) Combinatorial test generation using improved harmony search algorithm. International Journal of Hybrid Information Technology 8(9):121–130

Barbacci M, Klein MH, Longstaff TA, Weinstock CB (1995) Quality attributes. Tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST

Boehm BW, Brown JR, Lipow M (1976) Quantitative evaluation of software quality. In: Proceedings of the 2nd international conference on Software engineering, IEEE Computer Society Press, pp 592–605

Bracho-Rios J, Torres-Jimenez J, Rodriguez-Tello E (2009) A new backtracking algorithm for constructing binary covering arrays of variable strength. In: MICAI, Springer, pp 397–407

Brownlee J (2015) Clever algorithms: Nature-inspired programming recipes. URL http://www.cleveralgorithms.com/nature-inspired/swarm/bfoa.html

Bryce RC, Colbourn CJ (2007) The density algorithm for pairwise interaction testing. Software Testing, Verification and Reliability 17(3):159–182

Burnstein I (2003) Practical software testing: a process-oriented approach

Calvagna A, Gargantini A (2012) T-wise combinatorial interaction test suites construction based on coverage inheritance. Software Testing, Verification and Reliability 22(7):507–526

Chateauneuf M, Colbourn CJ, Kreher DL (1999) Covering arrays of strength three. Designs, Codes and Cryptography 16(3):235–242

Chen X, Gu Q, Zhang X, Chen D (2009) Building prioritized pairwise interaction test suites with ant colony optimization. In: Quality Software, 2009. QSIC'09. 9th International Conference on, IEEE, pp 347–352

Cohen DM, Dalal SR, Kajla A, Patton GC (1994) The automatic efficient test generator (aetg) system. In: Software Reliability Engineering, 1994. Proceedings., 5th International Symposium on, IEEE, pp 303–309

Cohen DM, Dalal SR, Fredman ML, Patton GC (1997) The aetg system: An approach to testing based on combinatorial design. IEEE Transactions on Software Engineering 23(7):437–444

Cohen MB, Colbourn CJ, Ling AC (2008) Constructing strength three covering arrays with augmented annealing. Discrete Mathematics 308(13):2709–2722

Colbourn CJ (2004) Combinatorial aspects of covering arrays. Le Matematiche (Catania) 58(121-167):0–10

Colbourn CJ (2010) Covering arrays from cyclotomy. Designs, Codes and Cryptography 55(2):201–219

Colbourn CJ, Martirosyan SS, Mullen GL, Shasha D, Sherwood GB, Yucas JL (2006) Products of mixed covering arrays of strength two. Journal of Combinatorial Designs 14(2):124–138

Dorigo M, Stützle M (2004) Ant colony optimization. bradford bks

Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 26(1):29–41

Fisher RA (1937) The design of experiments. Oliver And Boyd; Edinburgh; London

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

Geem ZW, Kim JH, Loganathan G (2001) A new heuristic optimization algorithm: harmony search. simulation 76(2):60–68

Gendreau M (2002) Recent advances in tabu search. In: Essays and surveys in metaheuristics, Springer, pp 369–377

Gendreau M, Potvin JY (2010) Handbook of metaheuristics, vol 2. Springer

Glover F (1986) Future paths for integer programming and links to artificial intelligence. Computers & operations research 13(5):533–549

Glover F, Laguna M (1997) Tabu search kluwer academic. Boston, Texas

Gonzalez-Hernandez L, Rangel-Valdez N, Torres-Jimenez J (2010) Construction of mixed covering arrays of variable strength using a tabu search approach. Combinatorial Optimization and Applications pp 51–64

Hartman A (2005) Software and hardware testing using combinatorial covering suites. Graph theory, combinatorics and algorithms 34:237–266

Hartman A, Raskin L (2004) Problems and algorithms for covering arrays. Discrete Mathematics 284(1):149–156

Hernández AG (2013) Un algoritmo de optimizacion combinatoria para la construccion de covering arrays mixtos de fuerza variable. PhD thesis, PhD. dissertation, Laboratorio de Tecnologías de la Información, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional

Kacker RN, Kuhn DR, Lei Y, Lawrence JF (2013) Combinatorial testing for software: An adaptation of design of experiments. Measurement 46(9):3745–3752

Katona GO (1973) Two applications (for search theory and truth functions) of sperner type theorems. Periodica Mathematica Hungarica 3(1-2):19–26

Kenndy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of IEEE international conference on neural networks, IEEE Press, vol 4, pp 1942–1948

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Neural Networks, 1995. Proceedings., IEEE International Conference on, IEEE, vol 4, pp 1942–1948

Kirkpatrick S, Gelatt CD, Vecchi MP, et al (1983) Optimization by simulated annealing. science 220(4598):671–680

Kleitman DJ, Spencer J (1973) Families of k-independent sets. Discrete Mathematics 6(3):255–262

Lei Y, Tai KC (1998) In-parameter-order: A test generation strategy for pairwise testing. In: High-Assurance Systems Engineering Symposium, 1998. Proceedings. Third IEEE International, IEEE, pp 254–261

Lei Y, Kacker R, Kuhn DR, Okun V, Lawrence J (2007) Ipog: A general strategy for t-way software testing. In: Engineering of Computer-Based Systems, 2007. ECBS'07. 14th Annual IEEE International Conference and Workshops on the, IEEE, pp 549–556

Luke S (2013) Essentials of metaheuristics, 2nd edn. lulu

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

Meagher K (2002) Non-isomorphic generation of covering arrays

Meagher K, Stevens B (2005) Group construction of covering arrays. Journal of Combinatorial Designs 13(1):70–77

Mladenović N, Hansen P (1997) Variable neighborhood search. Computers & operations research 24(11):1097–1100

Montgomery T, Parsopoulos KE, Vrahatis MN (2004) Design and analysis of experiments. Applied Numerical Analysis & Computational Mathematics 1(2):413–433

Nurmela KJ (2004) Upper bounds for covering arrays by tabu search. Discrete applied mathematics 138(1):143–152

Pressman R (2002) S. 2002, ingeniería de software: Un enfoque práctico

Radatz J, Geraci A, Katki F (1990) Ieee standard glossary of software engineering terminology. IEEE Std 610121990(121990):3

Rodriguez-Cristerna A, Torres-Jimenez J (2012) A simulated annealing with variable neighborhood search approach to construct mixed covering arrays. Electronic Notes in Discrete Mathematics 39:249–256

Rodriguez-Tello E, Torres-Jimenez J (2009) Memetic algorithms for constructing binary covering arrays of strength three. In: Artificial Evolution, Springer, pp 86–97

Rodriguez-Tello E, Hao JK, Torres-Jimenez J (2008) An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. Computers & Operations Research 35(10):3331–3346

Sabharwal S, Bansal P, Mittal N (2016) Construction of t-way covering arrays using genetic algorithm. International Journal of System Assurance Engineering and Management 8(2):264–274

for Standardization/International Electrotechnical Commission IO, et al (2001) Iso/iec 9126–software engineering–product quality

Stardom J (2001) Metaheuristics and the search for covering and packing arrays. Simon Fraser University

Talbi EG (2009) Metaheuristics: from design to implementation, vol 74. John Wiley & Sons

Torres-Jimenez J, Izquierdo-Marquez I (2013) Survey of covering arrays. In: Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013 15th International Symposium on, IEEE, pp 20–27

Torres-Jimenez J, Rodriguez-Tello E (2010) Simulated annealing for constructing binary covering arrays of variable strength. In: Evolutionary Computation (CEC), 2010 IEEE Congress on, IEEE, pp 1–8

Torres-Jimenez J, Avila-George H, Izquierdo-Marquez I (2017) A two-stage algorithm for combinatorial testing. Optimization Letters 11(3):457–469

Turban RC (2006) Algorithms for covering arrays. Ph.D. dissertation, Arizona State University

Walker RA, Colbourn CJ (2009) Tabu search for covering arrays using permutation vectors. Journal of Statistical Planning and Inference 139(1):69–80

Yan J, Zhang J (2006) Backtracking algorithms and search heuristics to generate test suites for combinatorial testing. In: Computer Software and Applications Conference, 2006. COMPSAC'06. 30th Annual International, IEEE, vol 1, pp 385–394

Yu X, Gen M (2010) Introduction to evolutionary algorithms. Springer Science & Business Media

Zamli KZ, Alkazemi BY, Kendall G (2016) A tabu search hyper-heuristic strategy for t-way test suite generation. Applied Soft Computing 44:57–74

Zhang J, Zhang Z, Ma F (2014) Automatic Generation of Combinatorial Test Data. Springer