Algoritmo de Colonia de Abejas Artificiales modificado para Problemas de Cobertura de Conjuntos de Gran Escala

NELSON ENRIQUE QUEMÁ TAIMBUD

Tesis de Maestría

Director: PhD. Martha Eliana Mendoza Becerra

Co-Director: PhD. Carlos Alberto Cobos Lozada

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Maestría en Computación

Grupo de I+D en Tecnologías de la Información (GTI)

Área de Investigación: Sistemas Inteligentes

Popayán, diciembre de 2023

Algoritmo de Colonia de Abejas Artificiales modificado para Problemas de Cobertura de Conjuntos de Gran Escala

NELSON ENRIQUE QUEMÁ TAIMBUD

Trabajo de grado presentado a la Facultad de Ingeniería
Electrónica y Telecomunicaciones de la Universidad
del Cauca para la obtención del Título de:
Magíster en Computación

Director: PhD. Martha Eliana Mendoza Becerra
Co-Director: PhD. Carlos Alberto Cobos Lozada

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Maestría en Computación
Grupo de I+D en Tecnologías de la Información (GTI)
Área de Investigación: Sistemas Inteligentes
Popayán, diciembre de 2023

Dedicado a:

A mi madre Aura Elisa,
por el todo el apoyo brindado,
por los consejos que me ha dado,
por corregir mis errores,
por la motivación de luchar por mis sueños,
por orientar mi camino haciendo posible este logro,
por su gran amor incondicional.

A mi padre Jesús Artemio,
por todo el cariño y apoyo brindados,
en el transcurso de mi vida y carrera profesional.

A mis hermanos William Andrés y Yesenia Elizabeth,
por estar presentes conmigo brindándome su apoyo incondicional.

AGRADECIMIENTOS

A los doctores Martha Mendoza y Carlos Cobos, por su paciencia, compromiso, empeño y destreza, logrando trasmitir su alto grado de conocimiento, el cual hizo posible la culminación de esta gran etapa.

Así mismo, a todos los profesores que hicieron parte de este proceso de formación, mil gracias por su excelente trabajo como educadores.

Gracias también a mis compañeros, con los que recorrimos esta maravillosa etapa, por su amistad, su apoyo y por los agradables y grandiosos momentos que logramos compartir.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ANEXOS

Anexo 1. Articulo “Initialization and Local Search Methods Applied to the Set Covering Problem: A Systematic Mapping”, publicado en la Revista Facultad de Ingeniería de la Universidad Pedagógica y Tecnológica de Colombia - UPTC, Vol. 32 No. 63 (2023): Enero-Marzo 2023 (Continuous Publication). https://doi.org/10.19053/01211129.v32.n63.2023.15235.

Link publicación: Initialization and Local Search Methods Applied to the Set Covering Problem: A Systematic Mapping | Revista Facultad de Ingeniería (uptc.edu.co)

(ver documento anexo)

RESUMEN

A lo largo de los años, las organizaciones se han enfrentado al desafío de optimizar el uso de sus recursos para satisfacer las necesidades de un sector específico. En este contexto surge el Problema de Cobertura de Conjuntos (PCC), un problema clásico de optimización combinatoria, informática y teoría de la complejidad computacional que se presenta al asignar recursos a un conjunto de necesidades al menor costo posible.

La complejidad del PCC aumenta con el número de necesidades (o restricciones) a satisfacer, incrementando el número de posibles soluciones y el tiempo necesario para comprobarlas. Ante este desafío, se han explorado nuevas formas de abordar el PCC recurriendo al uso de algoritmos metaheurísticos, los cuales ofrecen una alternativa eficiente para encontrar soluciones óptimas o cercanas a la óptima en un tiempo razonable, superando las limitaciones asociadas con el aumento de la complejidad del problema.

El presente trabajo propone el algoritmo ABC_SCP_IMP_RH_ILS para resolver PCC de gran escala, obtenido al reemplazar el método de inicialización y búsqueda local del algoritmo base del estado del arte colonia de abejas artificiales ABC_SCP con métodos que se han aplicado recientemente al problema. En este documento se describe el proceso de modificación del algoritmo ABC_SCP que involucró la identificación y selección de los métodos mencionados, la implementación del algoritmo base, el reemplazo de los métodos seleccionados en el algoritmo base para la obtención de seis variantes, y la selección de la variante ABC_SCP_IMP_RH_ILS con mejores resultados.

La evaluación de los algoritmos se realizó con 20 problemas de prueba del repositorio de investigación de operaciones OR-Library considerados de gran escala, tomando como referencia los mejores valores de aptitud conocidos (*BKS*) de cada problema para compararlos con el mejor valor de aptitud (*Best*) y el valor promedio (*Avg*) obtenido por las variantes. Los resultados muestran que las modificaciones realizadas al algoritmo base generaron un impacto positivo al mejorar los valores reportados en varios problemas por el mismo algoritmo y por otros del estado del arte. Estos resultados posicionan al ABC_SCP_IMP_RH_ILS como un algoritmo metaheurístico competitivo de referencia en el ámbito de investigaciones relacionadas con el PCC.

Palabras clave: Problema de cobertura de conjuntos, Metaheurísticas, Optimización combinatoria, ABC_SCP (Artificial Bee Colony for the Set Covering problem).

Capítulo 1

1. INTRODUCCIÓN

1.1. PLANTEAMIENTO DEL PROBLEMA

El problema de cobertura de conjuntos (PCC) es un problema clásico de optimización combinatoria que hace parte de los 21 problemas NP-completos de Karp [1]. Muchas aplicaciones del mundo real pueden modelarse como PCC, como la ubicación servicios de emergencia [2][3], planificación militar [4][5], toma de decisiones en situación de pandemia por COVID-19 [6][7], entre otros.

En la literatura, se han creado PCC de prueba publicados en repositorios de acceso libre. Un grupo de estos problemas fueron propuestos por [8] y se encuentran en el repositorio OR-Library [9]. La complejidad al resolver PCC es equivalente a las restricciones de un problema y a la cantidad de conjuntos que pueden cubrirlas. En OR-Library se disponen de 20 PCC denominados de gran escala, que contienen entre 500 y 1.000 restricciones y entre 5.000 y 10.000 conjuntos, implicando la evaluación de una gran cantidad de combinaciones de conjuntos. Estos PCC ponen a prueba la eficiencia de los algoritmos para encontrar soluciones óptimas.

Se han propuesto algoritmos para resolver el PCC a lo largo de los años. Un grupo de ellos son los algoritmos exactos, los cuales garantizan la optimización de las soluciones que encuentran, sin embargo, estos son capaces de resolver problemas con un tamaño limitado y en problemas de gran escala tienden a requerir un esfuerzo computacional sustancial [10][11]. En otro grupo se encuentran los algoritmos heurísticos (H) y metaheurísticos (MH), que consisten en procedimientos iterativos que guían una heurística subordinada, combinando diferentes conceptos para explorar y explotar adecuadamente el espacio de búsqueda de forma inteligente. Si bien las soluciones logradas a través de algoritmos MH no pueden garantizarse en términos de optimización, se ha demostrado empíricamente que son de calidad [12].

En la literatura se identifican dos enfoques en los algoritmos meta-heurísticos propuestos para resolver el PCC: 1) Algoritmos que no se han aplicado al PCC [13]–[19] y 2) Algoritmos que han sido aplicados al PCC y que son modificados para agregar, modificar o reemplazar sus componentes [11] [20]–[25] [26]–[30] [31]–[35].

En la revisión de la literatura se encontró que el algoritmo de colonia de abejas artificiales ABC_SCP (Artificial Bee Colony – Set Covering Problem, por sus siglas en inglés) [32] obtiene resultados iguales a los valores óptimos conocidos (*BKS*, Best Know Solutions por sus siglas en ingles) en la mayoría de PCC de OR-Library, con excepción de algunos de gran escala en los que no ha logrado obtener la mejor solución en las ejecuciones realizadas, lo cual se refleja en los valores mejor logrados (*Best*) y promedios (*Avg*) de las tablas de resultados reportadas. Esta dificultad puede generalizarse, ya que en los estudios revisados se presenta esta situación: en problemas de mediana escala el promedio de los valores *Best* y *Avg* logrados en cada iteración

de un algoritmo es igual al *BKS* y en problemas de gran escala este promedio no logra igualarse al *BKS*.

Este trabajo de maestría se enmarca en el enfoque de algoritmos que han sido aplicados al PCC y modificados, partiendo del algoritmo ABC_SCP, y en el cual serán reemplazados los métodos de inicialización y de búsqueda local por otros identificados en la literatura que han sido usados para PCC. Por lo tanto, se plantea la siguiente pregunta de investigación: ¿Qué impacto se genera en los resultados obtenidos por el algoritmo ABC_SCP al resolver los problemas de prueba NRE, NRF, NRG y NRH del repositorio de OR-Library, reemplazando su método de inicialización y de búsqueda local?

1.2. APORTES

Con la presente investigación se genera conocimiento con respecto al impacto que se genera al reemplazar el método de inicialización y de búsqueda local del algoritmo del estado del arte ABC_SCP aplicado al PCC para problemas de prueba de gran escala.

Desde el punto de vista académico y científico, con la definición e implementación del algoritmo propuesto se aporta nuevo conocimiento en el área de la computación inteligente, ya que se realizará el modelado de un nuevo algoritmo de para el PCC.

Finalmente, el algoritmo y enfoque propuesto podrán aplicarse en otras áreas como la geografía, investigación operacional y ciencias de la computación, en donde se formulan y solucionan problemas con dominio discreto modelados como PCC.

1.3. OBJETIVOS

1.3.1. Objetivo general

Proponer modificaciones al algoritmo del estado de arte de Colonia de abejas artificiales para el problema de Cobertura de conjuntos, buscando mayor exploración y explotación en la búsqueda de soluciones.

1.3.2. Objetivos específicos

- Identificar en el estado del arte los métodos de inicialización de la población y de búsqueda local que se han aplicado al problema de Cobertura de conjuntos PCC.

- Modificar el algoritmo de Colonia de abejas artificiales ABC_SCP con los métodos de inicialización de la población y de búsqueda local identificados.

- Evaluar el mejor valor de aptitud obtenido (*Best*) y el valor de aptitud promedio (*Avg*) obtenidos por las modificaciones propuestas al algoritmo de colonia de abejas artificiales, haciendo uso de los problemas de prueba NRE, NRF, NRG y NRH del repositorio OR-Library.

1.4. RESULTADOS OBTENIDOS

- **Monografía del trabajo de grado.** Presenta los conceptos teóricos necesarios para el desarrollo del proyecto. Luego, el estado del arte sobre los algoritmos recientes, los métodos de inicialización y búsqueda local que se han aplicado al PCC. Además, la modificación del algoritmo del estado del arte ABC_SCP con los métodos seleccionados, los resultados obtenidos al resolver los problemas de prueba. Finalmente, las conclusiones, recomendaciones y el trabajo futuro con base en esta investigación.

- **Algoritmo modificado** ABC_SCP_IMP_RH_ILS**.** Su definición se presenta en el capítulo 4 del presente documento.

- **Artículo de investigación.** Publicado en una revista categoría B del PUBLINDEX de Minciencias con la siguiente citación:

  Quemá-Taimbud, N.-E., Mendoza-Becerra, M.-E., & Bedoya-Leyva, O.-F. (2023). Initialization and Local Search Methods Applied to the Set Covering Problem: A Systematic Mapping. Revista Facultad de Ingeniería, 32(63), e15235. https://doi.org/10.19053/01211129.v32.n63.2023.15235.

Capítulo 2

2. CONTEXTO TEORICO Y ESTADO DEL ARTE

En este capítulo se presentan el contexto teórico necesario para el desarrollo de la presente investigación y la revisión del estado del arte realizada con el objetivo de conocer los algoritmos que se han aplicado al PCC en los últimos años.

2.1. CONTEXTO TEORICO

En esta sección se explica el problema de cobertura de conjuntos, el algoritmo del estado del arte que se toma como referencia, y los métodos de inicialización y búsqueda local que fueron usados en esta investigación.

2.1.1. Problema de Cobertura de Conjuntos

El problema de cobertura de conjuntos (PCC) es un problema clásico de optimización combinatoria y teoría de la complejidad. Es uno de los problemas de optimización más antiguos y estudiados, por las aplicaciones que tiene en situaciones reales y por la dificultad que este presenta en problemas de mediana y gran escala. En estos casos, resolver el PCC requiere de métodos o algoritmos eficientes en el uso de recursos computacionales [16].

El PCC se define de la siguiente forma: Dado un universo de elementos $E = \{e_1, e_2, \dots, e_m\}$, los conjuntos $X = \{x_1, x_2, \dots, x_n\}$ que contienen (o cubren) uno o varios de estos elementos y los costos no negativos $C = \{c_1, c_2, \dots, c_n\}$ asignados a cada conjunto de X. El objetivo de este problema es seleccionar un subconjunto $x \in X$, de tal forma que la unión de estos contenga todos los elementos de E y que la suma de sus costos C sea mínima. Si uno o varios de los costos C del PCC son mayores a uno, se conoce como PCC No-Unicosto, de lo contrario si todos los costos son iguales a 1 como PCC Unicosto.

Siguiendo la definición anterior, los PCC pueden representarse con una matriz binaria $A = [a_{ij}] \subset \{0,1\}$ de tamaño $m \times n$ conocida como matriz de restricciones. Las filas $I = \{1, 2, \dots, i, \dots, m\}$ corresponden a los elementos del universo E o restricciones del problema, las columnas $J = \{1, 2, \dots, j, \dots n\}$ a los conjuntos de X que las cubren, y si $a_{ij} = 1$, se interpreta que la fila i puede ser cubierta por la columna j. Cada columna j de la matriz A se asocia a un costo definido en un vector c_j y como solución del problema se obtiene un vector $x_j \subset \{0,1\}$ en el cual, si $x_j = 1$ entonces la columna j hace parte de la solución. Por lo tanto, el PCC se formula con las ecuaciones (1), (2) y (3).

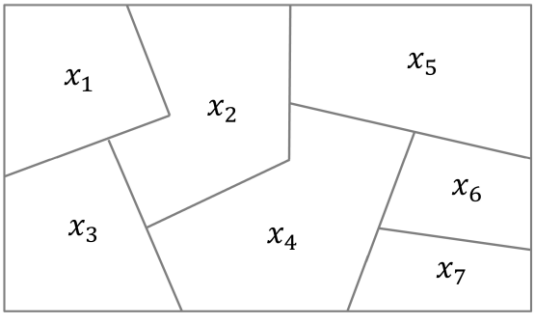
$$Minimizar\ f(x) = \sum_{j=1}^{n} c_j * x_j \quad (1)$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, \forall i \in I \quad (2)$$

$$x_j \in \{0, 1\} \tag{3}$$

La ecuación (1) define la sumatoria de los productos entre el vector x_j y los costos c_j, obteniendo como resultado de la evaluación de la función objetivo o valor de aptitud. La ecuación (2) indica que cada fila de A debe estar cubierta por al menos una columna j y la ecuación (3) que las variables de decisión de x_j son únicamente valores binarios (0 o 1).

Para un mejor entendimiento del problema se plantea el siguiente ejemplo: Se tiene un sector geográfico de siete barrios $(x_1, .., x_7)$ como se muestra en la Figura 1, y se requiere instalar estaciones de bomberos con el fin de atender de manera oportuna los llamados de emergencia de los habitantes de todo el sector. De igual forma se dispone de los costos relacionados con la instalación de las estaciones en cada barrio.



Barrio	Costo
x_1	1
x_2	1
x_3	2
x_4	2
x_5	1
x_6	2
x_7	1

Figura 1. Ejemplo PCC: Atención de emergencias en los barrios de un sector geográfico.

Se tienen las siguientes condiciones o restricciones en la instalación de las estaciones de emergencia:

- Una estación puede atender las emergencias del barrio en él está ubicada y las de los barrios vecinos o adyacentes.
- Cada barrio debe ser atendido por al menos una estación vecina o adyacente.

Una solución al problema es instalar estaciones en cada barrio, sin embargo, no es óptima ya que el costo total es igual al máximo costo posible. Por lo anterior, se busca seleccionar la combinación de uno o varios barrios según las restricciones definidas de tal forma que la suma de los costos de instalación sea lo menor posible. Este es un problema común que puede ser modelado como un PCC.

La Figura 2 muestra las restricciones Rx_j del problema de forma gráfica, observando por cada barrio las posibles estaciones adyacentes que pueden atender sus emergencias. Por ejemplo, las emergencias presentadas en el barrio x_1 pueden ser atendidas por la estación ubicada en el mismo barrio x_1 y las que están instaladas en los barrios adyacentes x_2 y x_3; las emergencias presentadas en el barrio x_2 pueden ser atendidas por la estación ubicada en el mismo barrio x_2 y las que están instaladas en los barrios adyacentes x_1, x_3, x_4 y x_5. Del mismo modo se identifican las restricciones para los demás barrios.

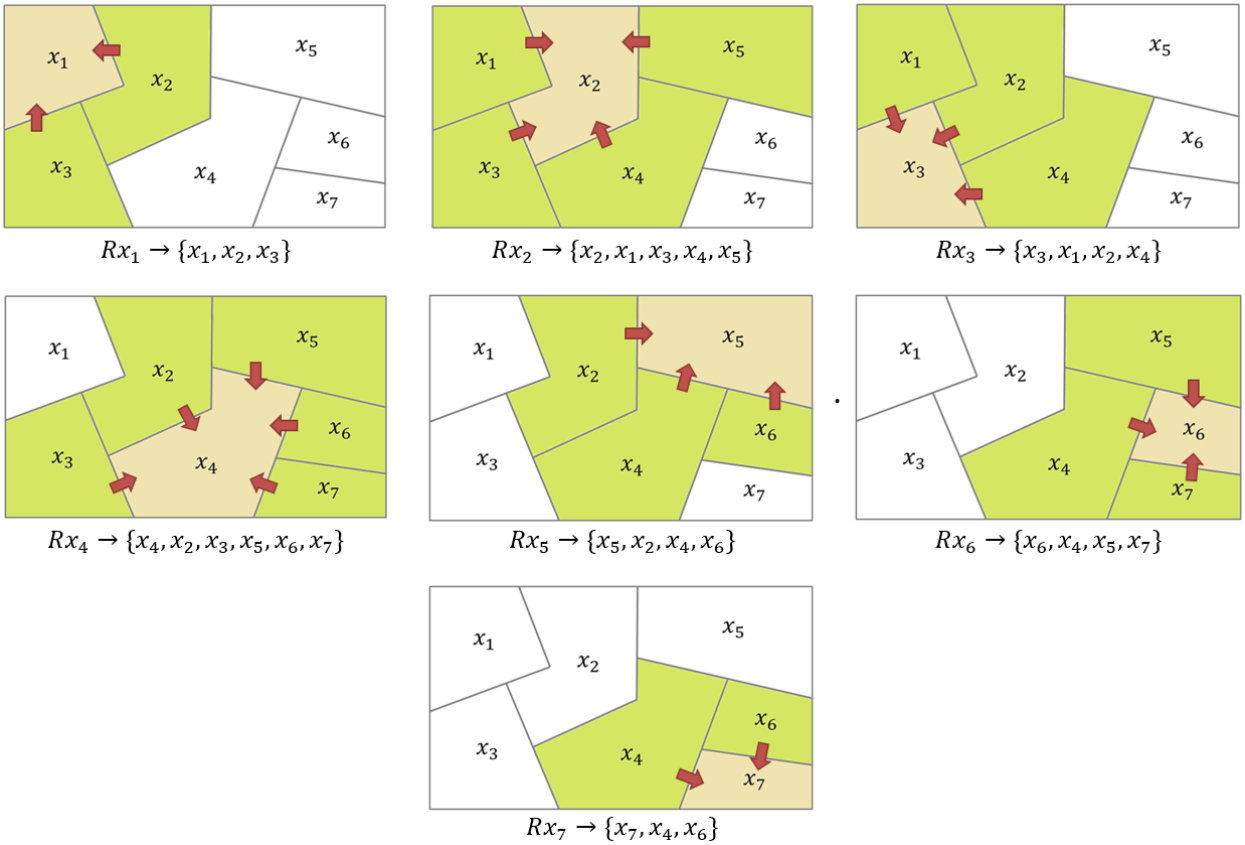


Figura 2. Ejemplo PCC: Visualización de restricciones.

Las restricciones del problema Rx_j pueden representarse en una matriz A de tamaño m x n con la cual se logran identificar los conjuntos de cobertura Cx_j y el vector solución x_j como se muestra en la Figura 3.

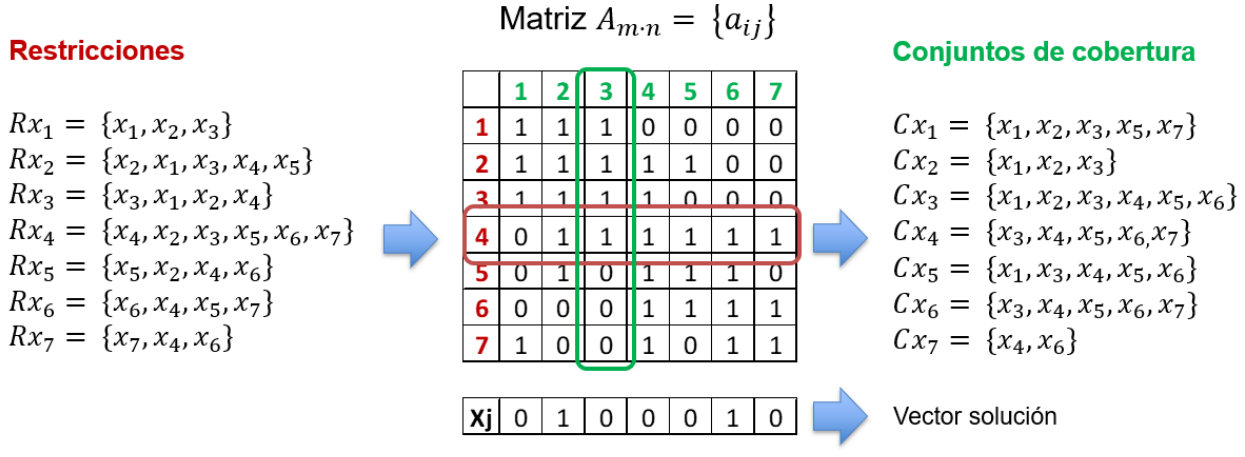
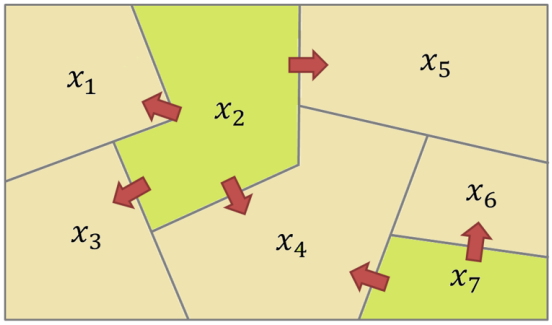


Figura 3. Identificación de conjuntos de cobertura a partir de matriz de restricciones.

Al observar los conjuntos de la matriz A se pueden inferir conjuntos que pueden dar solución al problema. Por ejemplo, la combinación de los conjuntos Cx_2 y Cx_6 cubren todas las restricciones

del problema, lo que significa que instalando estaciones en los barrios x_2 y x_6 se logran atender las emergencias de todos los barrios con un costo total de instalación de 3 unidades. Sin embargo, la combinación de los conjuntos Cx_2 y Cx_7 cubren de igual forma las restricciones del problema con un costo de 2 unidades, logrando ser la solución optima con respecto a la anterior alternativa y a otras posibles combinaciones de conjuntos (ver Figura 4). Por lo tanto, la solución al problema consiste en la instalación de estaciones de bomberos en los barrios x_2 y x_7 para atender las emergencias de todos los barrios del sector con un costo de instalación de dos unidades.



$$x_j = \{0, 1, 0, 0, 0, 0, 1\}$$

$$f(x) = \sum_{j=1}^{n} c_j * x_j = 2$$

Figura 4. Solución al PCC ejemplo

2.1.2. Algoritmo base

El algoritmo de colonia de abejas artificiales ABC_SCP (Artificial Bee Colony – Set Covering Problem, por sus siglas en inglés) es una metaheurística (MH) basada en población [32] que adopta ideas de [36], [37] para resolver el PCC, integrando componentes de aleatoriedad en la generación de fuentes de alimento a partir de una fuente seleccionada y un método de búsqueda local binaria basado en [38].

El diseño de esta MH se basa en el comportamiento de búsqueda de alimento (néctar) de las colonias de abejas que define tres roles:

- **Abejas empleadas:** recolectan el alimento y comunican la calidad de las fuentes de alimento mediante una danza con duración proporcional a la calidad de la fuente.

- **Abejas observadoras:** observan las danzas realizadas por las abejas empleadas y seleccionan fuentes de alimento para recolectar el alimento en función de su calidad.

- **Abejas exploradoras:** exploran el espacio en búsqueda de nuevas fuentes de alimento, identificación cuales han sido suficientemente explotadas, por lo cual son abandonadas, y selecciona nuevas fuentes. En el momento en el que estas abejas seleccionan una nueva fuente, se convierten en empleadas.

El algoritmo ABC_SCP se define siguiendo el comportamiento de las abejas descrito, en donde las abejas empleadas y observadoras se especializan en la explotación de soluciones, mientras

que las exploradoras se encargan de la exploración del espacio de búsqueda [36][37]. La Figura 2 muestra la implementación del algoritmo que define los pasos que se describen a continuación.

```
1  inicio ABC_SCP()
2  FA_k ← {∅}; ∀k = {1,2, … , FOOD_NUM}                    // fuentes de alimento
3  trial_k ← 0; ∀k = {1,2, … , FOOD_NUM}                   // intentos de mejora
4  fitness_k ← 0; ∀k = {1,2, … , FOOD_NUM}                 // valores de aptitud
5  Desde k = 1 hasta FOOD_NUM hacer                        // inicialización
6      S ← inicializarSolucion()
7      FA_k ← S
8      fitness_k ← calcularAptitud(S)
9  fin Desde
10 bestSol ← FA_0
11 bestFitness ← fitness_0
12 bestSol ← memorizarMejorSolucionGlobal(FA, fitness, bestSol, bestFitness)
13 Desde iter = 0 hasta MAX_ITER hacer
14   Desde k = 1 hasta EMPLOYEE_BEES hacer                 // abejas empleadas
15     S ← FA_k
16     rS ← seleccionarSolucionDistinta(FA, S)
17     dCols ← buscarColumnasDistintas(S, rS)
18     Si |dCols| > 0 hacer
19       S' ← modificarSolucion(S, dCols)
20       S' ← aplicarbusquedaLocal(S')
21       FA_k ← memorizarMejorSolucionLocal(S, S', trial_k, fitness_k)
22     Sino
23       S ← inicializarSolucion()
24       FA_k ← S
25       trial_k ← 0
26       fitness_k ← calcularAptitud(S)
27     fin Si
28   fin Desde
29   p_k ← calcularProbabilidades(FA_k); ∀k = {1, 2, … , FOOD_NUM}
30   Desde k = 1 hasta ONLOOKER_BEES hacer                 // abejas observadoras
31     S ← seleccionarSolucionMetodoRuleta(p_k, FA)
32     dCols ← seleccionarSolucionColumnasDistintas(FA, S)
33     S' ← modificarSolucion(S, dCols)
34     S' ← aplicarBusquedalocal(S')
35     FA ← memorizarMejorSolucionLocal(S, S', trial_k, fitness_k)
36   fin Desde
37   bestSol ← memorizarMejorSolucionGlobal(FA)
38   Desde k = 1 hasta FOOD_NUM hacer                      // abejas exploradoras
39     Si trial_k ≥ LIMIT hacer
40       S ← inicializarSolucion()
41       FA_k ← S
42       trial_k ← 0
43       fitness_k ← calcularAptitud(S)
44     fin Si
45   fin Desde
46 fin Desde
47 fin
```

Figura 5. Algoritmo ABC_SCP. Adaptado de [32]

Paso 1 – Inicialización de la población: Por cada $k = \{1, 2, \dots, FOOD\_NUM\}$ se crea la población FA_k de fuentes de alimento, la variable $trial_k$ que almacena el número de intentos de mejora de una solución asociada a una fuente FA_k, y la lista de valores de aptitud $fitness_k$ calculados al evaluar la función objetivo (líneas 1-4). Por cada fuente de alimento FA_k se inicializa una nueva solución S aplicando un método de inicialización aleatorio con columnas redundantes (ver sección 2.1.2.1), se adiciona a la población de fuentes de alimento FA y se procede a calcular su valor de aptitud agregando el resultado en la lista $fitness$ (líneas 5-9).

Paso 2 – Memorizar mejor solución global: Se recorren las fuentes de alimento FA para encontrar la fuente con el mejor valor de aptitud global de la población inicial, que es almacenado en la variable $bestSol$. Este valor es utilizado para validar las aptitudes obtenidas en pasos posteriores del algoritmo en la búsqueda de mejores soluciones.

Paso 3 - Abejas empleadas: En esta fase, cada solución S es comparada con otra solución rS seleccionada aleatoriamente. Se identifican las columnas en rS que son distintas de las que contiene S y se almacenan en la lista $dCols$. Si $dCols$ contiene al menos una columna distinta, se genera una nueva solución S' añadiendo un máximo de COL_ADD columnas de esta lista a S y eliminando COL_DROP columnas distintas de las añadidas. Se verifica la factibilidad de S, y si esta no cumple con las restricciones del problema, se aplica un operador de reparación. Luego, se intenta mejorar S' mediante un método de búsqueda local (ver sección 2.1.2.2) y se memoriza la mejor solución entre S' y S. Si no se encuentran columnas distintas entre las soluciones S y S' se identifica una colisión (las soluciones son iguales), por lo cual la abeja empleada se convierte en exploradora abandonando la fuente actual y eligiendo una nueva fuente que se relaciona con la inicialización de una nueva solución, recuperando al final su función como abeja empleada. Este proceso fomenta la exploración de soluciones al introducir diversidad de soluciones en la población FA.

Paso 4: Se recorre cada fuente de alimento FA_k, $k = \{1,2, \dots, FOOD\_NUM\}$ calculando la probabilidad p_k de ser seleccionada por las abejas observadoras mediante la ecuación (4).

$$p_k = \frac{fitness_k}{\sum_{t=1}^{N} fitness_t} \tag{4}$$

Como resultado de la ecuación se obtiene la relación entre el valor de aptitud $fitness_k$ de cada solución en FA y el valor total de aptitud de toda la población.

Paso 5 - Abejas observadoras: En esta fase, cada abeja elige una fuente de alimento de la población FA con base en la probabilidad pk de ser seleccionada. Luego se procede a buscar otra solución rS con columnas diferentes a las de S. A diferencia de la fase de abejas empleadas, esta búsqueda continúa hasta encontrar una solución rS con columnas distintas a las de S, descartando las colisiones que puedan encontrarse. En este punto, se realizan los procedimientos de modificación de la solución S (adición y eliminación de columnas), reparación de la solución resultante y aplicación del método de búsqueda local de la misma forma que en la fase de abejas empleadas.

Paso 6 - Memorizar mejor solución global: Al concluir la fase de abejas observadoras, se recorren las fuentes de alimentos en busca de la solución con mejor valor de aptitud logrado en la iteración actual y se compara con el valor almacenado en $bestSol$. En caso de encontrar una mejor solución, el valor $bestSol$ es reemplazado por el nuevo valor encontrado.

Paso 7 - Abejas exploradoras: En cada paso se realizan comparaciones de aptitud de las soluciones actuales con las generadas, reiniciando la variable $trial_k$ (número de intentos) a cero cuando se mejora una solución, o incrementándola en uno en caso contrario. Las abejas exploradoras evalúan este valor para identificar fuentes (soluciones) que hayan alcanzado un máximo de $LIMIT$ intentos de mejora. Estas fuentes son abandonadas y sustituidas por nuevas, lo que implica la inicialización de nuevas soluciones que se agregan a la población existente y el reinicio de la variable $trial$ correspondiente a cero. Este proceso de renovación garantiza la exploración continua del espacio de soluciones en busca de mejoras.

Paso 8: El algoritmo termina regresa al paso dos (2) realizando varias iteraciones sobre los pasos descritos, hasta cumplir con un máximo de MAX_ITER iteraciones.

En los pasos 2 y 6 se considera los casos en los que, al evaluar la función objetivo del PCC mediante la ecuación (1), se encuentran soluciones con valores de aptitud iguales. En estos el algoritmo ABC_SCP evalúa una función objetivo secundaria que calcula el número de filas cubiertas por las columnas de una solución S, manteniendo en la población la que cubre la menor cantidad de filas (ver ecuación (5)).

$$Min\ f(j) = \sum_{j \in S} |\beta_j| \tag{5}$$

La ecuación (5) identifica por cada columna de una solución S la lista de filas β_j que puede cubrir para luego sumar sus tamaños. Este enfoque se adopta para disponer de soluciones con columnas que cubren filas a un menor costo en la búsqueda de soluciones óptimas [32].

2.1.2.1. Método de inicialización ABC_SCP

El método de inicialización del algoritmo ABC_SCP se muestra en la Figura 6. Este inicializa un vector solución S de tamaño n y una lista u_i de tamaño m que almacena por cada fila i el conteo de las columnas de S que la cubren; inicialmente cada posición de esta lista se inicia en cero (líneas 1-3). Por cada fila $i \in I$ se identifica la lista α_i que contiene las columnas que pueden cubrir la fila i, se define la lista restringida de columnas RCL de tamaño RC_SIZE que contiene las columnas de menor costo de α_i, se selecciona aleatoriamente una de estas columnas y se adiciona a S (líneas 4-7). Luego, se identifica la lista β_j que contiene las filas i que la columna j puede cubrir y se incrementa en uno las posiciones de β_j en la lista u_i (línea 8-9). Al finalizar este paso, se obtiene una solución S factible que cumple con las restricciones del problema (línea 10).

```
1  inicio inicializarSolucionABC_SCP()
2  S ← {∅}
3  u_i ← 0, ∀i ∈ I
4  Para cada i ∈ I hacer
5     α_i ← buscarColumnasQueCubrenFila(i)
6     j ← seleccionarColumnaListaRC(α_i, RC_SIZE)
7     S ← S ∪ j
8     β_j ← buscarFilasCubiertasPorColumna(j)
9     u_i ← u_i + 1, ∀i ∈ β_j
10 fin Para
11 t ← |S|
12 Mientras t > 0 hacer
13    j ← seleccionarColumnaAleatoria(t)
14    β_j ← buscarFilasCubiertasPorColumna(j)
15    Si u_i ≥ 2, ∀i ∈ β_j hacer
16       S ← S − j
17       u_i ← u_i − 1, ∀i ∈ β_j
18    fin Si
19    t ← t − 1
20 fin Mientras
21 retornar S;
22 fin
```

Figura 6. Método de inicialización algoritmo ABC_SCP. Adaptado de [32].

Posteriormente se eliminan aleatoriamente columnas repetidas o redundantes en S, es decir, aquellas con las filas de su lista β_j que ya están siendo cubiertas por otras columnas, se identifican a partir de la lista u_i. Para esto se inicializa una variable t y se iguala al número de columnas que contiene S y se inicia un ciclo que itera decrementando este valor hasta cero (líneas 11-12). Por cada iteración de este ciclo se selecciona una columna j aleatoriamente en el rango [0, t], se obtiene la lista β_j que corresponde a la columna j seleccionada y se valida si los valores en u_i que corresponden a β_j son mayores o iguales a dos; si esta condición se cumple la columna j se elimina de la solución, se actualiza la lista u_i restando las posiciones de la lista β_j de la columna eliminada en uno, y se decrementa el valor de t en uno (líneas 13-20). Al finalizar, se tiene una solución S que contiene algunas columnas redundantes, esto con el fin de disponer de opciones adicionales en el proceso de búsqueda y contribuir con la reducción del costo computacional del operador de reparación del algoritmo ABC_SPC [32].

2.1.2.2. Método de búsqueda local ABC_SCP

Este método que se aplica en las etapas de abejas empleadas y exploradoras del algoritmo ABC_SCP, es una extensión de la búsqueda local codiciosa propuesta por [38]. En la Figura 7 se muestra la implementación del método que inicia calculando por cada fila el número de columnas u_i en la solución S actual que la cubre, luego se recorre cada columna j de la solución una por una en orden decreciente de su costo y se calcula el conjunto de filas P_j cubiertas únicamente por la columna j (líneas 1-8).

- Si el número de filas en P_j es cero, entonces la columna j se elimina de la solución y se considera la siguiente columna (líneas 9-11).
- Si el número de filas en P_j es una, se verifica si la columna j es la columna de menor costo que cubre la única fila i de P_j; si la condición se cumple la columna j se conserva en la solución, de lo contrario se reemplaza por la columna de menor costo i' (líneas 13-22).
- Si el número de filas en P_j es dos, significa que hay dos filas, i_1 e i_2, cubiertas únicamente por la columna j. En este caso se debe buscar la columna de menor costo j_1 que cubre a i_1 y la columna de menor costo j_2 que cubre i_2. Si j_1 y j_2 son distintos y la suma de los costos de j_1 y j_2 es menor o igual al costo de la columna j, entonces la columna j se reemplaza por j_1 y j_2 en la solución. Si j_1 y j_2 representan una misma columna que es diferente de j, entonces la columna j se reemplaza por la columna de menor costo j_1 (o j_2). Se actualizan los valores de u_i antes de considerar la siguiente columna (líneas 23-32).
- Si el número de filas en P_j es tres, significa que hay tres filas i_1, i_2 e i_3 cubiertas únicamente por la columna j. En este caso se busca la columna de menor costo j_1 que cubre i_1, la columna de menor costo j_2 que cubre i_2 y la columna de menor costo j_3 que cubre i_3. Si j_1, j_2 y j_3 son columnas distintas y la suma de sus costos es menor que el costo de la columna j, entonces la columna j se reemplaza por j_1, j_2 y j_3 en la solución. Si j_1, j_2 y j_3 representan una misma columna que es diferente de j entonces la columna j se reemplaza por la columna de menor costo j_1 (o j_2 o j_3). Si j_1, j_2 son iguales y j_3 es diferente, y la suma de costos es menor que el costo de la columna j, entonces la columna se reemplaza con estas dos columnas. Si j_1, j_3 son iguales y j_2 es diferente, y la suma de costos es menor que el costo de la columna j, entonces la columna se reemplaza con estas dos columnas. Finalmente, si j_2, j_3 son iguales y j_1 es diferente, y la suma de costos es menor que el costo de la columna j, entonces la columna se reemplaza con estas dos columnas (líneas 33-44).

```
1  inicio busquedaLocalABC_SCP(S)
2  u_i ← calcularNroColumnasQueCubrenCadaFila(S)
3  mejora ← true
4  Mientras mejora hacer
5     mejora ← false
6     S_orden ← ordenarColumnasSolucionPorCostoDecreciente(S)
7     Para cada columna j ∈ S_orden hacer
8        P_j ← obtenerFilasCubiertasSoloPorLaColumna(j, u_i)
9        Si |P_j| = 0 hacer
10          S ← eliminarColumna(S, j)
11          u_i ← actualizarNroColumnasQueCubrenCadaFila(u_i, j)
12          mejora ← true
13       de lo contrario Si |P_j| = 1 hacer
14          i1 ← P_j[0]
15          l_i1 ← buscarColumnaMenorCostoQueCubreFila(i1)
16          columnasMenorCosto ← validarColumna(j, l_i1)
17          Si columnasMenorCosto = true hacer
18             S ← eliminarColumna(S, j)
19             S ← adicionarColumnas(S, l_i1)
20             u_i ← actualizarNroColumnasQueCubrenCadaFila(u_i, j, l_i1)
21             mejora ← true
22          fin Si
23       de lo contrario Si |P_j| = 2 hacer
24          i1, i2 ← P_j[0], P_j[1]
25          l_i1, l_i2 ← buscarColumnasMenorCostoQueCubreFila(i1, i2)
26          columnasMenorCosto ← validarColumnas(S, l_i1, l_i2)
27          Si columnasMenorCosto = true hacer
28             S ← eliminarColumna(S, j)
29             S ← adicionarColumnas(S, l_i1, l_i2)
30             u_i ← actualizarNroColumnasQueCubrenCadaFila(u_i, j, l_i1, l_i2)
31             mejora ← true
32          fin Si
33       de lo contrario Si |P_j| = 3 hacer
34          i1, i2, i3 ← P_j[0], P_j[1], P[2]
35          l_i1, l_i2, l_i3 ← buscarColumnasMenorCostoQueCubreFila(i1, i2, i3)
36          columnasMenorCosto ← validarColumnas(S, l_i1, l_i2)
37          Si columnasMenorCosto = true hacer
38             S ← eliminarColumna(S, j)
39             S ← adicionarColumnas(S, l_i1, l_i2, l_i3)
40             u_i ← actualizarNroColumnasQueCubrenCadaFila(u_i, j, l_i1, l_i2, l_i3)
41             mejora ← true
42          fin Si
43       fin Si
44    fin Para cada
45 fin Mientras
46 retornar S
47 fin
```

Figura 7. Método de búsqueda local ABC_SCP. Adaptado de [32].

2.1.3. Métodos de inicialización

Se tomaron como referencia los métodos de inicialización del estado del arte que fueron identificados en el mapeo sistemático de la literatura realizado en la presente investigación [39], la selección de estos métodos se explica en la sección 3.1.1.

2.1.3.1. Inicialización aleatoria

La inicialización aleatoria (Random Method) es un método que genera soluciones de forma aleatoria utilizando un parámetro de probabilidad R [29]. Como se muestra en la Figura 3, este método primero inicializa el parámetro R con valor 0.5 y una solución vacía (líneas 1-2). Por cada columna j se genera un número aleatorio entre 0 y 1 (líneas 3-4) y si el número generado es menor al valor R la columna se adiciona a la solución, de lo contrario la columna j se descarta (líneas 5-8).

1	$R \leftarrow 0.5$
2	$S \leftarrow \{\emptyset\}$
3	$\boldsymbol{Para}\ cada\ j \in J\ \boldsymbol{hacer}$
4	$\quad rand \leftarrow generarAleatorio(0,1)$
5	$\quad \boldsymbol{Si}\ rand < R\ \boldsymbol{hacer}$
7	$\quad\quad S \leftarrow S \cup j$
8	$\quad fin\ \boldsymbol{Si}$
9	$fin\ \boldsymbol{Para}$
10	$\boldsymbol{retornar}\ S$

Figura 8. Método de inicialización aleatoria. Adaptado de [29]

2.1.3.2. Inicialización aleatoria con operador heurístico

Este método integra la identificación de mejores filas ($Best_i$), mejores columnas ($Best_j$) y la selección de forma aleatoria de las columnas que se adicionan a una solución, estos pasos se ejecutan mediante la aplicación de un operador heurístico. Este operador hace uso de la ecuación (5) para calcular el peso de cada fila (W_i) con base en las columnas que la cubren (α_i), y de la ecuación (6) para calcular los pesos de cada columna (W_j) con base en el costo de cada columna (c_j) las mejores filas encontradas y las filas que puede cubrir la columna j (β_j).

$$W_i = \frac{1}{|\alpha_i|} \tag{6}$$

$$W_j = \frac{c_j}{\left|Best_i \cap \beta_j\right|} \tag{7}$$

La Figura 4 muestra el pseudocódigo del método que inicia con la creación de una solución vacía S y la selección de una columna al azar que se adiciona a la solución (línea 1-3). Luego se identifican la lista de filas I_u que no están cubiertas por S y se inicia un ciclo que itera hasta que esta lista no este vacía (líneas 4-5). Por cada iteración, se calculan los pesos W_i de cada fila en I_u haciendo uso de la ecuación (6) y se identifican las 10 mejores filas con el menor valor posible

de W_i (líneas 6-7). Luego se buscan las columnas $cols$ que cubren a las filas $Best_i$, se calculan los pesos W_j de estas columnas utilizando la ecuación (7) y se identifican las 5 mejores columnas con el menor valor posible de W_j (líneas 8-10). Entre las columnas encontradas se selecciona aleatoriamente una que será adicionada a la solución (líneas 11-12), y se actualizan las filas no cubiertas para iniciar una nueva iteración (línea 13-14).

1	$S \leftarrow \{\emptyset\}$
2	$j \leftarrow seleccionarColumnaAleatoria(J)$
3	$S \leftarrow S \cup j$
4	$I_u \leftarrow buscarFilasNoCubiertas(S)$
5	$\boldsymbol{Mientras}\ I_u \neq \{\emptyset\}\ \boldsymbol{hacer}$
6	$\quad W_i \leftarrow calcular\ Pesos\ (I_u)$
7	$\quad Best_i \leftarrow buscarMejoresFilas(I_u, W_j, M = 10)$
8	$\quad cols \leftarrow buscarColumnasQueCubrenFilas(Best_i)$
9	$\quad W_j \leftarrow calcular\ Pesos\ (cols)$
10	$\quad Best_j \leftarrow buscarMejoresColumnas(cols, W_j, N = 5)$
11	$\quad j \leftarrow seleccionarColumnaAleatoria(Best_j)$
12	$\quad S \leftarrow S \cup j$
13	$\quad \beta_j \leftarrow identificarFilasQueCubreColumna(j)$
14	$\quad I_u \leftarrow actualizarColumnasNoCubiertas(I_u, \beta_j)$
15	$fin\ \boldsymbol{Mientras}$
16	$\boldsymbol{retornar}\ S$

Figura 9. Método de inicialización aleatoria con operador heurístico. Adaptado de [24]

2.1.3.3. Construcción iterada de soluciones

Este método [40] inicializa las soluciones basado en un mapa que contiene las columnas que puede cubrir cada fila ordenadas de menor a mayor cantidad. La Figura 5 muestra un ejemplo en el que a partir de una matriz de restricciones A_{ij}, se identifican la lista columnas j de cada fila i y se ordenan de menor a mayor cantidad.

$$A_{ij} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \Rightarrow \begin{matrix} i_2 \leftarrow j_2, j_3 \\ i_1 \leftarrow j_1, j_4, j_7 \\ i_3 \leftarrow j_2, j_4, j_5 \\ i_4 \leftarrow j_4, j_5, j_7 \\ i_5 \leftarrow j_1, j_3, j_5, j_7 \end{matrix}$$

Figura 10. Creación de mapa de columnas. Adaptado de [40]

En la evaluación y adición de columnas a una nueva solución S, este método calcula el peso de una columna W_j seleccionando aleatoriamente una de 6 funciones [41] que corresponden a las ecuaciones (8), (9), (10), (11) y (12), en donde c_j es el costo de cada columna y I_{uj} representa el número de columnas no cubiertas que puede cubrir la columna j.

$$W_j = \frac{c_j}{I_{uj}} \tag{8}$$

$$W_j = \frac{c_j}{\sqrt{I_{uj}}} \tag{9}$$

$$W_j = \frac{\sqrt{c_j}}{I_{uj}} \tag{10}$$

$$W_j = \frac{c_j}{\log\left(1 + I_{uj}\right)} \tag{11}$$

$$W_j = \frac{c_j}{{I_{uj}}^2} \tag{12}$$

$$W_j = \frac{c_j}{I_{uj} \log I_{uj}} \tag{13}$$

La Figura 6 muestra el pseudocódigo del método, este inicia con la creación de una solución vacía S, la creación del mapa de columnas y la ejecución de un ciclo que recorre los elementos del mapa hasta que esté vacío (líneas 1-3). En cada iteración se obtiene el primer elemento del mapa. se selecciona una función de evaluación de forma aleatoria y se calculan los pesos W_j de cada columna del elemento seleccionado (líneas 4-8). Luego se genera un numero aleatorio entre 0, 1 y se compara con el resultado de la división $1/|cols|$, si esta condición se cumple se selecciona la columna con probabilidad proporcional a W_j, de lo contrario se selecciona la columna que minimiza el valor de W_j (líneas 10-17). Luego se adiciona la columna seleccionada a S y se eliminan los elementos del mapa que contenga a la columna j para iniciar con un nuevo ciclo.

1	$S \leftarrow \{\emptyset\}$
2	$mapa \leftarrow crearMapa(A_{ij})$
3	$\boldsymbol{mientras}\ mapa\ \neq \{\emptyset\}\ \boldsymbol{hacer}$
4	$\quad cols \leftarrow obtenerPrimerElemento\ (mapa)$
5	$\quad f \leftarrow seleccionarFuncionAleatoria()$
6	$\quad \boldsymbol{Para}\ cada\ j \in cols\ \boldsymbol{hacer}$
7	$\quad\quad W_j \leftarrow calcularPesosColumnas(cols, f)$
8	$\quad fin\ \boldsymbol{Para}$
9	$\quad random \leftarrow generarAleatorio\ (0, 1)$
10	$\quad \boldsymbol{Si}\ random \leq \frac{1}{\lvert cols \rvert}\ \boldsymbol{hacer}$
14	$\quad\quad j \leftarrow seleccionar\ j \in cols\ con\ probabilidad\ proporcional\ a\ W_j$
15	$\quad \boldsymbol{Sino}$
16	$\quad\quad j \leftarrow seleccionar\ j \in cols\ que\ minimiza\ W_j$
17	$\quad fin\ \boldsymbol{Si}$
19	$\quad S \leftarrow S \cup j$
20	$\quad eliminarElementosEnMapaQueContenganColumna(j)$
21	$fin\ \boldsymbol{mientras}$
22	$\boldsymbol{retornar}\ S$

Figura 11. Método de construcción iterada de soluciones. Adaptado de [40]

2.1.4. Métodos de búsqueda local

Se tomaron como referencia los métodos de búsqueda local del estado del arte que fueron identificados en el mapeo sistemático de la literatura realizado en la presente investigación [39], la selección de estos métodos se explica en la sección 3.1.2.

2.1.4.1. Búsqueda local iterada

La búsqueda local iterada (IteratedLS) [40] es un método codicioso que se basa en la idea de eliminar de forma aleatoria grupos de columnas de una solución S, buscar nuevas columnas y adicionarlas para obtener una nueva solución factible. Este método utiliza el procedimiento de adición de columnas del método de construcción iterada de soluciones IterConstruct, definiendo un mapa de restricciones con las columnas no cubiertas y la aplicación de la función de evaluación que se muestra en la ecuación (8) para evaluar las soluciones.

La Figura 7 muestra el pseudocódigo del método IteratedLS que inicia con la creación de una lista vacía Q para almacenar grupos de columnas que será eliminadas de la solución, y una variable $improved$ que identifica si la solución S fue mejorada (líneas 1-3).

```
1  inicio busquedaLocalIterada(S)
2     Q ← {∅}
3     improved ← true
4     Mientras improved hacer
5        improved ← false
6        L ← {1, 2, … , J}
7        {L1, L2} ← agruparColumnas(L)
8        Q ← adicionarGrupo(Q, L1)
9        Q ← adicionarGrupo(Q, L2)
10       Mientras Q ≠ {∅} hacer
11          G ← obtenerUltimoGrupo(Q)
12          Q ← eliminarGrupo(G)
13          S' ← S
14          Para cada j ∈ G hacer
15             S' ← S' − j
16          fin Para
17          Iu ← buscarFilasNoCubiertas(S')
18          S' ← aplicarBusquedaGreedy(S', Iu)
19          Si aptitud(S') < aptitud(S) hacer
20             S ← S'
21             {L1, L2} ← agruparColumnas(L)
22             Q ← adicionarGrupo(Q, L1)
23             Q ← adicionarGrupo(Q, L2)
24             improved ← true
25          fin Si
26       fin Mientras
21    fin Mientras
22 retornar S
23 fin
```

Figura 12. Método de búsqueda local iterada IterLS. Adaptado de [40]

Luego se define un ciclo principal que itera mientras la solución S sea mejorada, en el que se cambia el valor de $improved$ a $false$ para ser actualizada en caso de encontrar mejores soluciones, se crea los vectores L con las columnas J del problema, L_1 y L_2 que contienen grupos con columnas de L seleccionadas aleatoriamente, para luego ser almacenados en la lista Q (líneas 5-9). Se inicia un ciclo secundario para recorrer cada grupo almacenado en Q hasta que esté vacío, en el que inicialmente se crea una copia de S en una variable S' y una variable G con el último grupo de columnas almacenado en Q, para eliminarlas de S (líneas 10-16). Luego de esto se identifican las columnas I_u que no están cubiertas por S' y se aplica el procedimiento Greedy que adiciona columnas a partir de un mapa de restricciones creado a partir de las filas I_u, para hacer que S' sea una solución factible (líneas 17-18).

Si el valor de aptitud de S' es menor al actual se reemplaza S por la nueva solución, se genera otro par de grupos con columnas aleatorias de L que se adicionan a Q y se cambia el valor de la variable $improved$ a verdadero para realizar intentos adicionales de mejorar la nueva solución (líneas 19-25).

2.1.4.2. Búsqueda local basada en ponderación de filas

La búsqueda local basada en ponderación de filas (RWLS, Row Weight based Local Search por sus siglas en inglés) [11], [42] se basa en la evaluación de puntuaciones asignadas a cada columna mediante el cálculo de pesos de las filas y prioridades de selección de columnas buscando evitar ciclos en la adición o eliminación de estas en las soluciones.

El método RWLS inicia con el procedimiento para obtener los puntajes iniciales de las columnas que se muestra en el pseudocódigo de la Figura 8.

```
1  inicio calcularPuntajesRWMLS(S)
2    weight_i ← 1, ∀i ∈ I
3    priortty_j ← 0, ∀j ∈ J
4    score_j ← 0, ∀j ∈ J
5    Para cada j ∈ J hacer
6      β_j ← buscarFilasQuePuedeCubrirColumna(j)
7      cp_j ← calcularPrioridad(β_j, c_j)
8      Si S_j = 1 hacer
9        score_j ← (−1) * pj
10     Sino
11       score_j ← p_j
12     fin Si
13   fin Para
14 retornar weight_i, priority_j, score_j
```

Figura 13. Cálculo inicial de puntajes método RWLS. Adaptado de [11]

Este inicia con la creación de variables para almacenar los pesos de cada fila, las prioridades y los puntajes de cada columna (líneas 2-4). Por cada columna $j \in J$ se obtiene la lista β_j que contiene las filas que j puede cubrir y se calculan las prioridades $priority_j$ de cada columna

aplicando la ecuación (14), la cual calcula la sumatoria del cociente entre los pesos de las filas β_j y el costo c_j de la columna j (líneas 6-7):

$$priority_j = \sum_{i \in \beta_j} \frac{weight_i}{c_j} \tag{14}$$

Luego, se asignan los puntajes a cada columna $j \in J$, en donde el valor $score_j$ de una columna j es igual a valor negativo de la prioridad $priority_j$, si j hace parte de la solución, en caso contrario el puntaje de j será el valor positivo de la prioridad $priority_j$ (línea 8-12). Como resultado de este procedimiento se tiene los pesos, prioridades y puntajes de cada columna que serán procesado por el ciclo principal del método RWLS (líneas 13-14).

El pseudocódigo del método RWLS que se muestra en la Figura 9, inicia con el cálculo de pesos, prioridades y puntajes de cada columna retornados por el procedimiento de la Figura 7, la creación de una copia S'de la solución S que ingresa al método, la variable $iter$ para controlar el ciclo principal del método y la variable t_j que incrementa su valor en uno al adicionar una columna o eliminar una columna j de una solución (líneas 1-5).

Se define un ciclo principal que itera hasta que se realicen MAX_ITER iteraciones que inicia con un ciclo secundario que itera hasta encontrar filas no cubiertas la validación del valor de aptitud de una solución actual S y una nueva solución S' y realizando el reemplazo en caso de ser mejorada (líneas 6-11).

En caso de no encontrar mejor solución se procede a eliminar la columna j' con el mayor puntaje $score_j$ incrementando su tiempo t, actualizando la lista de filas no cubiertas I_u y actualizando el puntaje de la columna haciendo uso de la ecuación (15), la cual calcula la sumatoria del cociente entre los pesos de las filas que están contenidas entre β_j y I_u y el costo c_j de la columna j (líneas 12-17):

$$score_j = \sum_{i \in \beta_j \cap I_u} \frac{weight_i}{c_j} \tag{15}$$

Luego, se elimina una segunda columna j'' aplicando el mismo procedimiento de eliminación para la columna j' (línea 18-22) y se busca con la adición de la columna con mayor puntaje $score_j$ y mayo tiempo t_j para ser adicionada a S' (líneas 23-24), se actualizan las filas no cubiertas I_u aplicando la ecuación (15) y se actualiza el puntaje de la columna ingresada con el valor negativo actual (líneas 23-26). Se actualiza el puntaje de las columnas que están contenidas en S y que son diferentes a la que fue ingresada aplicando la ecuación (16) (líneas 27-29), se actualizan los pesos $weight_i$ incrementando en 1 las filas que no han sido cubiertas (líneas 30-32) y se actualiza el puntaje de las columnas que no hacen parte de S con la ecuación (17) y se incrementa el valor de la variable $iter$ en 1 para iniciar con una nueva iteración (líneas 33-36). Finalmente, al culminar con las MAX_ITER iteraciones del método, este retorna la mejor solución encontrada (líneas 37-38).

$$score_h = score_h + \sum_{i \in \beta_j \cap I_h} \frac{weight_i}{c_h} \tag{16}$$

$$score_h = score_h - \sum_{i \in \beta_j \cap I_h} \frac{weight_i}{c_h} \tag{17}$$

```
1   inicio busquedaLocalRWMLS(S)
2     weight_j, priority, score_j ← calcularPuntajesRWMLS(S)
3     S' ← S
4     iter ← 0
5     t_j ← 0, ∀j ∈ J
6     Mientras iter < MAX_ITER hacer
        // eliminación de columnas
7       Mientras I_u ≠ {∅} hacer
8         Si aptitud(S') < aptitud(S) hacer
9           S ← S'
10          Ir a linea 18
11        fin Si
12        j' ← buscarColumnaConMayorPuntaje(S', score_j)
13        S' ← S' − j'
14        t_j' ← t_j' + 1
15        I_u ← actualizarFilasNoCubiertas(S')
16        score_j' ← actualizarPuntajes(j', I_u, weights_i)
17      fin Mientras
18      j'' ← buscarColumnaConMayorPuntaje(S', score_j)
19      S' ← S' ∪ j''
20      t_j'' ← t_j'' + 1
21      I_u ← actualizarFilasNoCubiertas(S')
22      score_j'' ← actualizarPuntajes(j'', I_u, weights_j)
        // adición de columnas
23      j ← seleccionarColumnaConMayorPuntajeYMayorTiempo(J, score_j, t_j)
24      S' ← S' ∪ j
25      I_u ← actualizarFilasNoCubiertas(S')
26      scores_j ← (−1) * scores_j
27      Para cada h ∈ S' y h ≠ j hacer
28        score_h ← actualizarPuntajeColumnasEnSolucion(h, I_u, weights_i)
29      fin Para
30      Para cada i ∈ I_u hacer
31        weights_i ← weights_i + 1
32      fin Para
33      Para cada h ∈ J y h ∉ S hacer
34        score_h ← actualizarPuntajeColumnasNoEnSolucion(h, I_u, weights_i)
35      fin Para
36      iter ← iter + 1
37    fin Mientras
38  retornar S
```

Figura 14. Método de búsqueda local RWLS. Adaptado de [32]

2.2. ESTADO DEL ARTE

Se realizó un mapeo sistemático para identificar los algoritmos que se han aplicado al PCC, siguiendo algunas de las guías planteadas en [43]. A continuación, se describe el proceso de búsqueda y selección de estudios realizado.

2.2.1. Planeación

En esta etapa se definen los lineamientos para realizar el mapeo iniciando con la definición de la pregunta de investigación y su motivación que se muestran en la Tabla 1.

Tabla 1. Pregunta de investigación y motivación del mapeo sistemático

	Pregunta	**Motivación**
P1	¿Qué algoritmos se han aplicado al Problema de Cobertura de Conjuntos (PCC) utilizando los problemas de prueba de OR-Library?	Reconocer los algoritmos del estado del arte que se han aplicado al Problema de Cobertura de Conjuntos.

Con el fin de buscar los estudios relacionados a las preguntas de investigación, se diseñó la cadena de búsqueda que se muestra en la Tabla 2.

Tabla 2. Cadena de búsqueda

Pregunta	**Cadena de búsqueda**
P1	algorithm **AND** *heuristic* **AND** "set covering problem" **AND "**OR-Lib***"

Además, para la ejecución de esta cadena se seleccionaron las bases de datos *Scopus, Springerlink, Science Direct y ACM,* y se definieron los criterios de inclusión (CI) y exclusión (CE) que serán aplicados a los resultados obtenidos (ver Tabla 3).

Tabla 3. Criterios de inclusión y exclusión aplicados

Criterios de inclusión	
Criterio	**Descripción**
CI1	Aborda el PCC clásico.
CI2	Comparan sus resultados con otros algoritmos del estado del arte.
CI3	Han sido publicados desde el año 2017 hasta el 2022.
CI4	Están publicados en idioma inglés.
Criterios de exclusión	
Criterio	**Descripción**
CE1	Abordan variantes o aplicaciones del PCC.
CE2	Estudios duplicados o repetidos.

2.2.2. Ejecución

En esta etapa se aplican los lineamientos de búsqueda definidos a cada base de datos seleccionada. Los estudios relevantes se obtienen al revisar el título, palabras clave y resumen de los documentos, y los estudios primarios al revisar el contenido de los estudios relevantes. La Tabla 4 presenta el número total de estudios encontrados, los estudios relevantes, repetidos y los primarios que se toman en cuenta para la definición del estado del arte.

Tabla 4. Conteo de estudios encontrados

Base de datos	Estudios encontrados	Estudios relevantes	Estudios relevantes repetidos	Estudios primarios seleccionados
Scopus	91	55	0	16
Springerlink	67	34	10	8
Science Direct	56	20	10	6
ACM	16	8	7	0
Total	230	117	27	30

Después de la revisión de estudios primarios seleccionados, se encontraron dos tipos de algoritmos: los heurísticos (H) diseñados para solucionar un único problema y los metaheurísticos (MH) que pueden ser aplicados a diversos problemas o contextos. A continuación, se presenta un resumen de los enfoques encontrados.

- **Algoritmos heurísticos.** En [10] se propone el algoritmo uSCL, basado en hipergrafos no dirigidos asociados a pesos, para resolver el PCC Unicosto, el cual, introduce estrategias de verificación de configuración de hiperborde para evitar ciclos en la adición eliminación de columnas, selección de hiperbordes para evitar soluciones iguales y una estrategia de asignación diversidad de pesos a las columnas para explorar ampliamente el espacio de búsqueda. En [40] se presenta un enfoque basado en GRASP, que incorpora la construcción iterada de soluciones, un método de búsqueda local iterada y un mecanismo de recompensa/penalización de columnas; este algoritmo sigue el esquema divide y vencerás, dividiendo la solución, reiniciando una parte de ella y reanudando la búsqueda desde este punto. Por otra parte, en [44] se introduce el algoritmo MLSES-CC centrado en la estrategia de verificación de configuración (CC) para abordar el PCC Unicosto, el cual, incorpora técnicas como verificación de configuración del estado del elemento (ES-CC), puntuación multinivel de subconjuntos (MSL) y una búsqueda local agresiva que altera el estado de tres soluciones simultáneamente.

- **Algoritmos metaheurísticos.** Dentro de estos algoritmos se identificaron dos enfoques principales que se describen a continuación.

    - **Primer enfoque,** que corresponde a los algoritmos que no han sido aplicados al PCC anteriormente: en [13], propone un algoritmo basado en Hormigas León, donde las hormigas son utilizadas como mecanismos de búsqueda, y las hormigas León perturban

su movimiento para mejorar la eficiencia en la búsqueda de soluciones. Los conceptos clave incluyen paseos aleatorios, construcción de vagabundos, captura de hormigas y reconstrucción de trampas. En [14], se presenta un algoritmo basado en el comportamiento de los saltamontes, considerados plagas agrícolas, estos insectos exhiben un comportamiento de enjambre único en sus fases larvaria y adulta, reflejando tendencias de exploración y explotación presentes en algoritmos inspirados en la naturaleza. En [17] se presenta el algoritmo de enjambre de moscas de la fruta (FFSA), basado en el comportamiento de búsqueda de alimento de estas moscas, que utiliza búsquedas locales (olfato) y globales (visión) para mejorar la calidad de las soluciones. En [18] se propone la optimización basada en la enseñanza-aprendizaje (TLBO), un algoritmo poblacional inspirado en la dinámica de un profesor que enseña a estudiantes, donde, las fases del Profesor y del Alumno buscan elevar el conocimiento de los estudiantes buscando que la clase muestra fluidez, basado en evaluaciones o un límite de iteraciones. En [19] se presenta el Algoritmo de Optimización de Redes Sociales, inspirado en la dinámica interactiva de plataformas como Twitter, que emula el comportamiento de usuarios que elevan ciertos tweets a mayor relevancia, utilizando relaciones sociales para mejorar la resolución de problemas complejos de optimización, como el PCC.

En varios de los estudios mencionados se destaca la aplicación de esquemas de binarización de dos pasos para hacer posible su adaptación al PCC, estos conservan las características de búsqueda en espacio continuo de un algoritmo permitiendo transferirlas a un espacio binario buscando equilibrar la exploración y explotación del espacio de soluciones [27]. En un primer paso, se aplica un componente que transfiere uno o varios valores de decisión continuos generados por un algoritmo MH (ej. coeficientes de desplazamiento, probabilidad de permutación de bits) a valores dentro del intervalo [0,1], y en el segundo paso se aplica otro componente que convierte estos valores en binarios mediante la validación de reglas. Las configuraciones de componentes para cada paso incluyen Funciones de transferencia y Binarizacion [17][18] y Funciones de transferencia con Operador percentil para el agrupamiento de distancias [13][14][31][15].

- **Segundo enfoque**, que corresponde a los algoritmos que se han aplicado al PCC y que modifican algún componente para evaluar el impacto en los resultados obtenidos: En [32] se presenta un enfoque híbrido que combina un algoritmo de colonia de abejas artificiales (ABC) con una búsqueda local para resolver el PCC No Unicosto. En [11] se propone el algoritmo Búsqueda Local de Ponderación de Filas (RWLS, por sus siglas en inglés) basado en un algoritmo genético que consta de tres elementos: un esquema de ponderación que actualiza los pesos de las filas para evitar óptimos locales, estrategias para asignar prioridades y puntajes en la selección de columnas, y un método de marca de tiempo que favorece la selección de conjuntos que no han sido adicionados en la solución candidata. Estos componentes combinados buscan mejorar la exploración del espacio de búsqueda, la calidad de las soluciones y la diversificación. En [29] se presenta una variante del algoritmo de búsqueda de monos binario, un enfoque de inteligencia de

enjambre inspirado en el comportamiento de los monos al buscar alimentos en nuevas ubicaciones. El algoritmo se compone de un proceso de escalada de montañas, salto con vigilancia buscando identificar puntos más altos y reinicia la escalada, cooperación entre monos para moverse hacia la posición del mono con la montaña más alta y un proceso de "salto mortal" que amplía el dominio de búsqueda, evitando caer en óptimos locales, aunque su eficacia puede disminuir con el tiempo. En [31] se propone el algoritmo de agujeros negros binario multidinámico (MDBBH, por sus siglas en inglés), que asigna probabilidades de transición a grupos de partículas para favorecer transiciones de mayor o menor distancia y equilibrar la exploración y explotación en el espacio de búsqueda. El algoritmo utiliza una ecuación para actualizar partículas cercanas a un agujero negro, incluyendo un operador de reparación para manejar violaciones de restricciones, inspirándose en una heurística codiciosa.

En [33] se propone un algoritmo metaheurístico de dos enfoques basados en vecindades para resolver el PCC: (i) utiliza el descenso de vecindario variable (VND, por sus siglas en inglés), que define dos estructuras de vecindad basadas en el intercambio de conjuntos dentro y fuera de la solución actual; (ii) utiliza recocido simulado (SA, por sus siglas en inglés), que define los vecinos añadiendo o eliminando un único conjunto y los parámetros como la temperatura inicial y la velocidad de enfriamiento se estiman heurísticamente. En [30] se presenta el algoritmo Ant-Set, una propuesta basada en colonia de hormigas artificiales (ACO, por sus siglas en ingles) para abordar el problema PCC, con dos características principales que son: (i) la orientación de filas, que implica construir soluciones mediante la selección de filas descubiertas; y (ii) la manipulación de feromonas asociada con las conexiones entre los componentes del gráfico de construcción, indicando la conveniencia de incluir conjuntos de componentes en una solución. Además, Ant-Set incorpora una búsqueda local basada en el trabajo de Jacobs y Brusco (1995), que ajusta la solución según un umbral de costo de columna para mejorar las soluciones obtenidas por las hormigas artificiales.

En [45] se propone un algoritmo de gotas de agua adaptativo inspirado en el recorrido inteligente que realizan las gotas de agua en ríos, lagos y océanos buscando el mejor camino (solución óptima) para llegar a su destino, adaptando en cada iteración la cantidad de suelo presente en el recorrido de un punto a otro y la velocidad de desplazamiento utilizando información interna (sobre la eficiencia en cuanto a la calidad de las soluciones obtenidas) y externa (sobre las características del problema). En [46] se propone un algoritmo adaptativo basado en el fenómeno de los agujeros negros y la absorción de estrellas cercanas causada por su fuerza gravitacional, en el que se busca la estrella más alejada de los agujeros (solución óptima), adaptando en cada iteración el número de estrellas de la población, este es el único parámetro disponible para ser modificado.

En [47] se propone un algoritmo de búsqueda adaptativo que consta de tres componentes principales: procedimiento de recocido simulado que comienza con una temperatura inicial y se ajusta iterativamente según un programa de enfriamiento, un procedimiento

de mutación que pretende diversificar la búsqueda partiendo una probabilidad de mutación por cada columna de una solución, y una función que evalúa la calidad de la solución mediante un parámetro de penalización que se adapta en la evolución del algoritmo. En [48] se propone una variante del algoritmo ABC para el PCC y un algoritmo genético secundario (AG) que afina los parámetros número de fuentes de alimento, límite de intentos de mejora de una solución, porcentaje de columnas a agregar y porcentaje de columnas a eliminar de forma automática al inicio de cada iteración. Para el afinamiento de parámetros, el AG implementa las etapas de generación de población inicial con los parámetros definidos, selección de individuos mediante el método de torneo, cruce, mutación y reemplazo de generaciones.

En varios de estos estudios se identificó la modificación de algoritmos con esquemas de binarización de dos pasos buscando evaluar el impacto que se tiene en los resultados. Se identifica la configuración de funciones de transferencia con un operador percentil de agrupamiento de distancias [24], [25], [26], Funciones de transferencia y de Binarización [27], [28], [35] y funciones de transferencia con operadores basados en aprendizaje de máquina [20]–[23], [34].

De igual forma, en esta revisión sistemática se identificó que el algoritmo con mejores resultados al resolver los problemas de OR-Library fue el de abejas artificiales ABC_SCP [32], el cual fue tomado como base para la presente investigación.

Capítulo 3

3. PROCESO DE MODIFICACIÓN ALGORITMO ABC_SCP

En este capítulo se muestra el proceso de modificación del algoritmo base (ABC_SCP), iniciando con la selección de los métodos de inicialización y búsqueda local, luego con la implementación de ABC_SCP, seguido de la modificación de este con los métodos, y por último los resultados obtenidos en comparación con las diferentes modificaciones del ABC_SCP implementado.

3.1. CICLO 1 – SELECCIÓN DE MÉTODOS

Se presenta la selección de los métodos para modificar el algoritmo ABC_SCP. Para esto se tomó como punto de partida los resultados del mapeo sistemático de la literatura [39] realizado en la presente investigación, en el que fueron identificados los métodos de inicialización y búsqueda local que han sido utilizados recientemente en el estado del arte.

3.1.1. Selección métodos de inicialización

Los métodos de inicialización que han sido utilizados recientemente en el estado del arte, los estudios y la cantidad en los que es aplicado se muestra en la Tabla 5.

Tabla 5. Métodos de inicialización. Adaptado de [39]

Método de inicialización	Estudios	Cantidad
Aleatorio (Random)	[17], [29]	2
Aleatorio con operador heurístico (RHeuristic)	[13]–[16], [20]–[26], [34]	12
Construcción iterada de soluciones (IterConstr)	[40]	1
Basados en algoritmos n-aproximados (NAprox)	[49], [50]	2
Total		**18**

Para la selección de los métodos de inicialización se definieron tres criterios: dos basados en las recomendaciones presentadas en [32], [50] respecto a la inicialización de soluciones y un criterio definido en esta investigación relacionado con la implementación.

1. **Aleatoriedad:** Involucra generar nuevas soluciones adicionando columnas de forma aleatoria, buscando explorar el espacio de búsqueda con un costo computacional menor [32], [50].

2. **Redundancia:** La presencia de columnas redundantes en las soluciones permite disponer de opciones que favorecen la explotación de soluciones en el algoritmo ABC_SCP y contribuye con la reducción del costo computacional del operador de reparación utilizado en el algoritmo ABC_SPC [32].

3. **Adaptación al algoritmo base:** Se verifica en el estudio primario si su especificación es suficiente para implementarlo y si este requiere recursos software adicionales.

Los criterios aplicados a los métodos de inicialización, indicando si estos lo cumplen (Cumple) o no (No cumple), se presentan en la Tabla 3. A continuación se explica el cumplimiento del primer criterio de **Aleatoriedad**:

- El método aleatorio con operador heurístico (RHeuristic) cumple, debido a que selecciona columnas de forma aleatoria a partir de la selección de mejores filas y columnas que las cubren.

- El método aleatorio (Random) lo cumple, porque por cada columna evalúa un parámetro de probabilidad y valida si esta se adiciona a una solución.

- El método de construcción iterativa (IterCostruct) cumple, debido a que a partir de la evaluación de un criterio de probabilidad selecciona columnas para adicionar ya sea por el resultado de la relación entre el costo y la cantidad de filas que puede cubrir o según la probabilidad proporcional a esta relación.

- Los métodos basados en algoritmos aproximados (AA) Determinista (AAD), Aleatorio (AAR) y Aleatorio II (AAR2) cumplen, dado que inician buscando una solución del PCC interpretado como un problema lineal relajado [51] que contiene por cada columna, valores continuos en el rango [0, 1]. Estos son evaluados bajo distintos criterios con valores generados aleatoriamente similar al método de inicialización aleatoria (Random) (ver sección 2.1.3.1) adicionando columnas a una nueva solución binaria.

- Por el contrario, las variantes de los AA que integran búsqueda local (LO) no cumplen, debido a que al incluir este elemento se obtiene soluciones cercanas a los *BKS*, incrementando la probabilidad que esta correspondan a óptimos locales del problema.

Tabla 3. Aplicación de criterios de selección a métodos de inicialización

Método de Inicialización	**Criterios definidos**		
	Aleatoriedad	**Redundancia**	**Adaptable**
RHeuristic	Cumple	Cumple	Cumple
Random	Cumple	Cumple	Cumple
IterConstr	Cumple	Cumple	Cumple
AAD	Cumple	Cumple	No cumple
AAR	Cumple	Cumple	No cumple
AAR2	Cumple	Cumple	No cumple
AAD + LO	No cumple	No cumple	No cumple
AAR + LO	No cumple	No cumple	No cumple
AAR2 + LO	No cumple	No cumple	No cumple

En cuanto al criterio de **Redundancia**, con la adición de columnas basada en componentes de aleatoriedad, las soluciones generadas con los métodos RHeuristic, Random, IterConstruct, AAD, AAR y AAR2 contienen columnas redundantes o repetidas. Las variantes de los métodos basados en algoritmos aproximados AAD + LO, AAR + LO y ARR2 + LO no cumple ya que integran un procedimiento de búsqueda local mejorando las soluciones generadas para lo cual eliminan las columnas redundantes.

En cuanto al criterio **Adaptación al algoritmo base**, los métodos RHeuristic, Random, IterConstruct lo cumplen ya que el estudio dispone de la especificación y pseudocódigo con los detalles necesarios para su implementación y adaptación al algoritmo ABC_SPC. Los métodos basados en AA no cumplieron, porque para obtener las soluciones al PCC lineal relajado que requieren estos métodos, fue utilizado el software comercial CPLEX utilizado para resolver problemas de optimización e investigación de operaciones el cual requiere licenciamiento para su uso.

Después de aplicar los criterios definidos a los métodos de inicialización, se seleccionaron los métodos **RHeuristic, Random, IterConstruct**.

3.1.2. Selección métodos de búsqueda local

Los métodos de inicialización que han sido utilizados recientemente en el estado del arte, los estudios en los que es aplicado y la cantidad de estos, se muestran en la Tabla 6.

Tabla 6. Métodos de búsqueda local. Adaptado de [39]

Método de búsqueda local	Estudios	Cantidad
Mejor vecino local (BNLS)	[17]	1
Operador basado en KNN	[23]	1
búsqueda local Iterada (IterLS)	[40]	1
Estrategia de verificación (CSLS)	[10], [44]	2
Basada en ponderación de filas (RWLS)	[11], [42], [52]	3
Búsqueda local Jacobs & Brusco (JBLS)	[30]	1
Total		**9**

Para la selección de estos métodos se definieron dos criterios en esta investigación los cuales son:

1. **Resultados reportados:** Se comparan los resultados reportados al aplicar los métodos de búsqueda local para los problemas NRE, NRF, NRG, NRH del repositorio OR-Library, buscando identificar los mejores resultados. Para esto, se verifica que el valor promedio reportado (*Avg*) en las ejecuciones realizadas sea igual o cercano a los *BKS*.

2. **Adaptación al algoritmo base:** Se verifica la especificación del método en el estudio del estado del arte, validando que contenga los elementos suficientes para su implementación.

Los criterios aplicados a los métodos de búsqueda local se muestran en la Tabla 6, indicando si estos cumplen con el criterio (Cumple) o no (No cumple).

Tabla 7. Aplicación de criterios de selección a métodos de búsqueda local

Método de búsqueda local	**Criterio definido**	
	Resultados reportados	**Adaptable**
RWLS	Cumple	Cumple
IterLS	Cumple	Cumple
KNN	Cumple	No cumple
S-LV	No cumple	Cumple
CCS	No cumple	No cumple
JBLS	No cumple	No cumple

Para el primer criterio de **Resultados reportados**, se realizó la comparación de resultados (ver Tabla 8) identificando resultados relevantes, que son iguales a los *BKS* y cercanos (menores que el *BKS* incrementado en 1):

- La búsqueda local basado en filas ponderadas RWLS reporta los mejores resultados al obtener en los grupos de problemas NRE, NRF, NRH y en los problemas NRG1, NRG5 el valor de aptitud promedio igual al *BKS*. En los problemas NRG2, NRG3, NRG4 se observa un valor promedio cercano.

- En los resultados de la búsqueda local Iterada IterLS se observan resultados iguales a los *BKS* en 3 problemas (NRE1, NRE5 y NRF2) y resultados cercanos en 7 problemas (NRE2, NRE3, NRE4, NRF1, NRF3, NRF4 y NRF5).

- En los resultados del operador basado en KNN (K Nearnest Neighbors, por sus siglas en inglés) se observa 12 resultados cercanos a los *BKS* (grupos NRE y NRF, problemas NRG5 y NRH5) y ninguno con valor promedio igual.

- En los resultados del método S-LV se observa un solo valor igual al *BKS* (NRE1) y dos valores cercanos (NRE5, NRF2). Con base en estos resultados se define que este método no cumple con este criterio.

Por otra parte, al verificar los resultados de los métodos Estrategia Verificación de Configuración (CCS, Configuratión Checking Strategy por sus siglas en ingles) y búsqueda local JB (JB, Jacobs and Brusco por las iniciales de sus autores) no fue encontrado el valor promedio que se verifica en este criterio en los estudios relacionados, por lo cual estos no cumplieron con el criterio.

Con respecto al segundo criterio de **Adaptación al algoritmo base**, se encontró lo siguiente:

- El método S-LV lo cumple, ya que se compone de dos pasos principales: (i) Generación de *S* vecinos a partir del intercambio de *L* bits de una solución y (ii) la evaluación de las

soluciones generadas. Sin embargo, se requiere adicionar un ciclo con terminación que permita explotar una solución mejorada.

- El método IterLS cumple, ya que se compone de etapas en las que se generan nuevas soluciones a partir de una mejorada y aplica un parámetro de culminación en el caso que las soluciones no logren mejorar, este método no requiere modificaciones.

- Los métodos KNN y CCS se componen de varias etapas que realizan perturbaciones a las soluciones, lo cual incrementaría la complejidad del algoritmo ABC_SCP. Por este motivo estos métodos se descartan de la investigación.

- De igual forma, se descartó la búsqueda local JBLS debido a la especificación incompleta en el estudio de referencia. Adicional a ello, en la búsqueda de detalles en el estudio del que fue tomado como referencia, la especificación encontrada y del algoritmo JBLS no coincide.

- Por último, se define que el método RWLS cumple con este criterio debido a los resultados destacados que reporta reflejados en la aplicación de una estrategia de ponderación de filas y asignación de puntajes a las columnas del problema; este método será adaptado previamente para ser aplicado al algoritmo base.

Tabla 8. Resultados reportados al aplicar los métodos de búsqueda local

Problema	BKS	S-LV	KNN	IterLS	RWLS
		Avg	Avg	Avg	Avg
NRE1	29	**29**	**29.4**	**29**	**29**
NRE2	30	32.13	**30.2**	**30.5**	**30**
NRE3	27	28.7	**27.6**	**27.83**	**27**
NRE4	28	29.63	**28.7**	**28.1**	**28**
NRE5	28	**28.93**	**28.4**	**28**	**28**
NRF1	14	15	**14.6**	**14.2**	**14**
NRF2	15	**15.9**	**15.2**	**15**	**15**
NRF3	14	16.73	**14.7**	**14.7**	**14**
NRF4	14	15.03	**14.8**	**14.2**	**14**
NRF5	13	15.1	**13.9**	**13.77**	**13**
NRG1	176	180.3	177.2	177.2	**176**
NRG2	154	160.43	156.6	157.47	**154.2**
NRG3	166	171.57	169.2	170.97	**166.6**
NRG4	168	172.2	170.2	174.27	**168.2**
NRG5	168	175	**168.6**	172.5	**168**
NRH1	63	67.47	64.6	65.37	**63**
NRH2	63	66	64.8	65	**63**
NRH3	59	63	60.7	60.97	**59**
NRH4	58	63.5	59.4	59.57	**58**
NRH5	55	58.07	**55.2**	**55.83**	**55**

Después de aplicar los criterios definidos a los métodos de búsqueda local se seleccionaron los métodos **búsqueda local iterada ILS y búsqueda local basada en la ponderación de filas RMLS** para modificar el algoritmo ABC_SCP.

3.2. CICLO 2 – MODIFICACIÓN DEL ALGORITMO

En este ciclo se implementa el algoritmo base ABC_SCP con los problemas de prueba del repositorio OR-Library. Luego se presenta el proceso de modificación del algoritmo ABC_SCP implementado, identificando las adaptaciones que fueron realizadas a los métodos de inicialización y búsqueda local. Por último, se muestran los resultados obtenidos al ejecutar con las diferentes modificaciones del ABC_SCP implementado con los métodos seleccionados.

3.2.1. Implementación del algoritmo base

Se implementa el algoritmo base (ABC_SCP_IMP) siguiendo la especificación de la sección 2.1.2. A continuación, se describe aspectos de la implementación.

3.2.1.1. Interpretaciones del algoritmo

En algunos pasos no fue posible disponer de los detalles necesarios, por lo cual se realizó una interpretación propia basada en el conocimiento del problema. La Tabla 9 muestra un resumen de estas interpretaciones.

La primera interpretación (Tabla 9, primera fila) se relaciona con el Paso 5 - Abejas exploradoras y el cálculo de probabilidades de selección p_j de las fuentes de alimento. El algoritmo ABC_SCP no especifica si al encontrar una mejor solución se debe actualizar el listado p_j con la probabilidad, por lo cual, en esta implementación se definió actualizar este listado y así tenerla en cuenta en el momento de las abejas exploradoras seleccionen otra fuente.

Tabla 9. Interpretaciones en la implementación del algoritmo ABC_SCP_IMP

Paso	Procedimiento	Interpretación
Paso 5 - Abejas observadoras	Evaluación de soluciones con la función objetivo principal (ver ecuación (1)).	Al encontrar una mejor solución, se actualiza la lista de probabilidades de selección p_j utilizada en la fase de abejas observadoras.
Paso 3 - Abejas empleadas, Paso 5 - Abejas observadoras	Evaluación de soluciones con la función objetivo secundaria (ver ecuación (5)).	Al encontrar una mejor solución, se reinicia la variable $trial$ en 0, de lo contrario se incrementa en 1.
Paso 2, Paso 6 - Memorizar mejor solución global	Evaluar la mejor solución de la ejecución.	Evaluar soluciones con las funciones objetivo principal (ver ecuación (1)) y secundaria (ver ecuación (5)).

En el Paso 3 – Abejas empleadas y Paso 5 – Abejas observadoras (ver Tabla 9, segunda fila) en el procedimiento de memorizar las mejores soluciones, al encontrar dos soluciones con igual

valor de aptitud al evaluar la función objetivo principal y encontrar una mejor solución al evaluar la función secundaria, no se especifica si se tiene que reiniciar a cero el número de intentos de mejora $trial$ de la solución o se deba incrementar en uno en caso contrario. En esta implementación se definió aplicar el reinicio cuando se encuentra una mejor solución al evaluar la función secundaria o un decremento en caso contrario, esto con el fin de mantener en la población soluciones con columnas que cubran el menor número de filas partiendo de la idea de que si una columna está presente en una buena solución, es muy probable que esté presente en otras buenas soluciones [32].

En el Paso 2 y Paso 6 – Memorizar la mejor solución global en cada iteración (ver Tabla 9, tercera fila), no se especifica si al evaluar la función objetivo principal y encontrar soluciones con igual valor de aptitud se tenga que evaluar la función secundaria. Basado en la idea expuesta en el párrafo anterior, se consideró adicionar la evaluación de la función objetivo secundaria.

3.2.1.2. Definición de parámetros

En la ejecución del algoritmo se aplicaron las definiciones del uso de semillas generadas aleatoriamente y el uso de los parámetros de [32], con los que se obtuvieron los mejores resultados (ver Tabla 10).

Tabla 10. Parámetros de ejecución algoritmo ABC_SCP_IMP

Nombre	Descripción	Valor
$RUNTIME$	Número de ejecuciones	10
MAX_ITER	Máximo número de iteraciones	500
$FOOD\_NUM$	Número de fuentes de alimento (Tamaño de la población)	50
$EMPLOYEE\_BEES$	Número de abejas empleadas	50
$ONLOOKER\_BEES$	Número de abejas observadoras	150
$LIMIT$	Máximo número de intentos para mejorar una solución S	50
RC_SIZE	Tamaño de la lista restringida de columnas candidatas RC	6
COL_ADD_1	Número de columnas a adicionar si $\|S\| > 35$	5
COL_ADD_2	Número de columnas a adicionar si $\|S\| \leq 35$	3
COL_DROP_1	Número de columnas a eliminar si $\|S\| > 35$	12
COL_DROP_2	Número de columnas a eliminar si $\|S\| \leq 35$	5
Pa	Probabilidad manejada en el operador de reparación	0.9

A nivel global del algoritmo se establece el número de ejecuciones ($RUNTIME$) en 10, el máximo número de iteraciones por ejecución (MAX_ITER) en 500 y el número de fuentes de alimento (tamaño de la población) ($FOOD\_NUM$) en 50.

En las fases de abejas empleadas, observadoras y exploradoras se establece el número de abejas empleadas ($EMPLOYEE\_BEES$) en 50, el número de abejas observadoras ($ONLOOKER\_BEES$) en 150 y el máximo número de intentos de mejora de una solución S ($LIMIT$) en 50.

En la fase de abejas empleadas y observadoras se definen los parámetros para la adición (COL_ADD_1 y COL_ADD_2) y la eliminación de columnas (COL_DROP_1 y COL_DROP_2) considerando el tamaño de los problemas de OR-Library.

En el procedimiento de inicialización de soluciones, y en las fases de abejas empleadas y exploradoras se establece el tamaño de la lista restringida de columnas RCL (RC_SIZE) en 6 y el parámetro de probabilidad utilizado en la reparación de soluciones (Pa) en 0,9.

3.2.1.3. Resultados ejecución algoritmo base

La ejecución del algoritmo ABC_SCP_IMP se realiza con los problemas de prueba de gran escala NRE, NRF, NRG y NRH del repositorio OR-Library [9]. La Tabla 3 muestra los resultados de este algoritmo y se comparan con los reportados por el algoritmo base original (ABC_SCP base). En esta tabla se presenta los resultados de los 20 problemas de prueba con respecto al mejor valor de aptitud conocido de cada problema (*BKS*), el mejor valor obtenido en las ejecuciones (*Best*) y el valor promedio (*Avg*, Average por su abreviatura en inglés).

Tabla 11. Resultados obtenidos algoritmo ABC_SCP_IMP.

Problema	**BKS**	**ABC_SCP base**		**ABC_SCP_IMP**	
		Best	**Avg**	**Best**	**Avg**
NRE1	29	29	29	29	29
NRE2	30	30	30	30	30
NRE3	27	27	27	27	27
NRE4	28	28	28	28	28
NRE5	28	28	28	28	28
NRF1	14	14	14	14	14
NRF2	15	15	15	15	15
NRF3	14	14	14	14	14
NRF4	14	14	14	14	14
NRF5	13	13	13.7	**13**	**13.6**
NRG1	176	176	176	176	176
NRG2	154	155	155	155	155
NRG3	166	**166**	**166**	166	166.2
NRG4	168	168	168	168	168
NRG5	168	168	168	168	168
NRH1	63	63	63.9	63	63.9
NRH2	63	63	63.9	63	63.9
NRH3	59	59	59	59	59
NRH4	58	58	58	58	58
NRH5	55	55	55	55	55

Como se observa en esta tabla, los resultados son iguales en 18 de los 20 problemas de prueba (grupos NRE y NRH, problemas NRF1 a NRF4, NRG1, NRG2, NRG4, NRG5). En el problema NRF5 se logra un resultado menor al reportando en 0.1 unidades y en el problema NRG3 se obtuvo un resultado 0.2 unidades mayor. En general se puede observar que los resultados son similares a los reportados por el estudio primario.

3.2.2. Modificación del algoritmo base implementado

Para la modificación del algoritmo ABC_SCP_IMP se realizó el reemplazo de los métodos de inicialización y de búsqueda local que define por los métodos seleccionados en la sección 3.1. Se llevaron a cabo adaptaciones a varios de los métodos de inicialización y búsqueda local seleccionados para ajustar parámetros adicionales o especificar detalles de los métodos.

3.2.2.1. Adaptaciones métodos de inicialización

Las adaptaciones realizadas a los métodos de inicialización se muestran en la Tabla 12. En la modificación del algoritmo ABC_SCP_IMP con el método Random se detectó un error al resolver el problema de prueba NRE3 causado por colisiones entre soluciones de la población. Al adicionar una nueva solución generada por las abejas exploradoras a la población, en el momento en que esta contiene soluciones con valores de aptitud cercanos al *BKS* del problema (el algoritmo converge), esta es modificada adicionando columnas de otras soluciones hasta el punto de no ser posible encontrar columnas distintas, generando un error por ciclo infinito en esta búsqueda.

Este caso de colisión generado por el método Random puede verse en la Figura 6 en la fuente de alimento de la posición 15. Por el error identificado, se define cambiar el valor del parámetro R sugerido en [29] en 0.5 a 0.1 para que las soluciones generadas contenga un menor número de columnas (ver Figura 3, línea 5).

Tabla 12. Modificaciones realizadas a los métodos de inicialización

Método	Problema	Modificación
Random Method	Error en el algoritmo al intentar buscar soluciones con columnas distintas y no encontrarlas.	Se ajusta el parámetro $R = 0.1$ para generar soluciones con menos columnas
IterConstruct	La condición $rand \leq 1/\|cols\|$ retorna siempre falso dado que el resultado de $1/\|cols\|$ es siempre igual a 0,003.	Se modifica la condición por $rand \leq (1/\|cols\|) * 100$ buscando ejecutar casos en que sea verdadera
RHeuristic	No se presentaron problemas	Ninguna

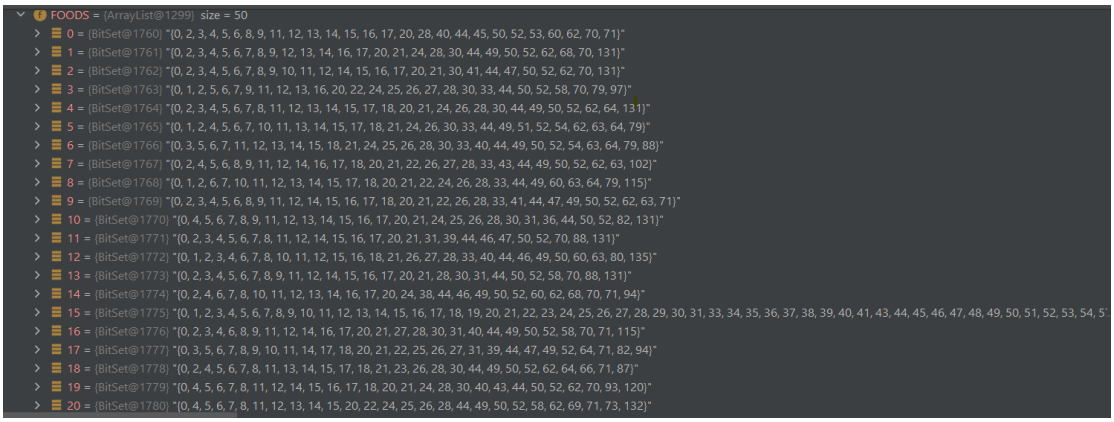


Figura 15. Problema de colisión en el problema NRE3 con el método Random

Por otra parte, al modificar el algoritmo base con el método de inicialización IterConstruct se pudo identificar un error al evaluar la operación $random * 1/|cols|$ (ver Figura 6, línea 6) dado que esta retorna valores menores a 0.003 (ver Figura 11), causando que únicamente se adicionen columnas que minimicen el valor de la función de evaluación f seleccionada descartando el caso de adicionarlas por la probabilidad proporcional al resultado de esta función. Por lo anterior, se define aplicar el producto $(random * 1/|cols|) * 100$ con el fin de ejecutar casos que correspondan a la adición de columnas por la probabilidad proporcional a f.

```
0.0022371364653243847
0.002232142857142857
0.002232142857142857
0.0022271714922048997
0.0022172949002217295
0.002207505518763797
```

Figura 16. Resultados de la operación aplicada en el método IterLS

Por último, en la implementación del método RHeuristic no se encontraron problemas por lo cual se mantiene su especificación original.

3.2.2.2. Adaptaciones métodos de búsqueda local

Las adaptaciones realizadas a los métodos de búsqueda local se muestran en la Tabla 13. Con respecto a la implementación del método IterLS basada en su especificación original (ver sección 2.1.4.1) se logró identificar casos en que se selecciona un porcentaje mayor al 50% de columnas de una solución que son eliminadas para ser reparadas obteniendo una nueva solución, lo cual hace que se pierda gran parte de las columnas adicionadas previamente por el algoritmo base y a la vez queden una gran cantidad de filas no cubiertas, incrementando los tiempos de ejecución al reparar soluciones. Por lo anterior se define eliminar un máximo de COL_DROP_1 o COL_DROP_2 columnas (aproximadamente un 10% de las columnas de una solución) según el tamaño del problema para reducir la cantidad de filas no cubiertas y el costo de procesamiento. Por lo anterior, la unión de los grupos L_1 y L_2 contendrán el número de columnas a eliminar definido.

Tabla 13. Modificaciones realizadas a los métodos de búsqueda local

Método	Problema	Modificación
IterLS	Tiempos de ejecución altos al reparar soluciones con un 50% de columnas eliminadas	Eliminar un máximo de COL_DROP_1 o COL_DROP_2 columnas de una solución agrupadas de forma aleatoria, siendo $\|L_1\| + \|L_2\|$ igual al valor seleccionado.
RWLS	Ciclo principal define un parámetro de terminación que corresponde a un alto número de iteraciones	Cambio del parámetro de terminación por intentos de mejorar una solución solo cuando esta es mejorada.

La Tabla 14 muestra resultados de experimentos preliminares para el grupo de problemas NRG, observando el mejor valor de aptitud (*Best*), el valor de aptitud promedio de las ejecuciones (*Avg*) y los tiempos de ejecución promedio en segundos (Time) de dos variantes de algoritmo base con el método Random y el método IterLS: (i) La primera implementa la especificación original del método de búsqueda local (ABC_SCP_IMP_RD_ILS1) y (ii) La segunda implementa el método adaptado (ABC_SCP_IMP_RD_ILS2), logrando observar la diferencia considerable de tiempo y la similitud en los resultados obtenidos.

Tabla 14. Tiempos de ejecución al aplicar método IterLS en los problemas NRG

Problema	ABC_SCP_IMP_RM_ILS1			ABC_SCP_IMP_RM_ILS2		
	Best	Avg	Time (seg.)	Best	Avg	Time (seg.)
NRG1	176	176.1	1202.7	176	176	**412.9**
NRG2	155	155	9201.2	155	155	**3620.2**
NRG3	166	167.3	11533	166	167.5	**3592.8**
NRG4	168	169.3	9134.6	169	169.4	**1150.1**
NRG5	168	168	865.25	168	168	**170.32**

Con respecto a la implementación del método RWLS según su especificación original (ver sección 2.1.4.2), este define un ciclo principal que culmina hasta cumplir con 5000 iteraciones (ver Figura 8, línea 6); al modificar el algoritmo ABC_SCP_IMP con este método, se obtuvo un incremento considerable en el consumo de recursos y tiempo de ejecución de la implementación realizada que incluso no permitió obtener resultados. Con el objetivo de optimizar el método y reducir el costo computacional, fue modificado el ciclo principal para que tuviese una condición que evalúa si una solución fue mejorada y realice intentos adicionales para mejorarla, esto puede verse en la Figura 12.

```
1   inicio busquedaLocalRWMLS(S)
2     weight_j, priority, score_j ← calcularPuntajesRWMLS(S)
3     S' ← S
4     iter ← 0
5     t_j ← 0, ∀j ∈ J
6     improved ← true        // variable que identifica si una solución fue mejorada
7     Mientras improved hacer
8       improved ← false
      // eliminación de columnas
8       Mientras I_u ≠ {∅} hacer
          // Se reubica comparación de soluciones S y S'' a las líneas 32-35
9         j' ← buscarColumnaConMayorPuntaje(S', score_j)
10        S' ← S' − j'
11        t_j' ← t_j' + 1
12        I_u ← actualizarFilasNoCubiertas(S')
13        score_j' ← actualizarPuntajes(j', I_u, weights_i)
14      fin Mientras
15      j'' ← buscarColumnaConMayorPuntaje(S', score_j)
16      S' ← S' ∪ j''
17      t_j'' ← t_j'' + 1
18      I_u ← actualizarFilasNoCubiertas(S')
19      score_j'' ← actualizarPuntajes(j'', I_u, weights_j)
        // procedimiento de adición de columnas
20-31   ...
32      Si aptitud(S') < aptitud(S) hacer  // evaluación de mejor solución
33        S ← S'
34        improved ← true
35      fin Si
36    fin Mientras
37  retornar S
```

Figura 17. Método de búsqueda local RWLS adaptado.

3.2.2.3. Definición de parámetros y variantes del algoritmo base

En la Tabla 15 muestra los parámetros de ejecución definidos para los métodos de inicialización y búsqueda local: (i) en el método de inicialización Random se define la probabilidad de adicionar una columna a una solución (R) en 0.1; (ii) en el método RHeuristic se define encontrar 10 mejores filas (M) y 5 mejores columnas (N) para seleccionar una columna aleatoria; (iii) el método IterConstruct no define el uso de parámetros por lo cual no requiere esta definición.

En cuanto a los métodos de búsqueda local: (i) para el método IterILS se define eliminar un máximo de 12 columnas (COL_DROP_1) si el numero de columnas de una solución es mayor a 35, de lo contrario se reinician máximo 5 columnas (COL_DROP_2); (ii) se define el parámetro de probabilidad para incluir una columna en los grupos L_1 y L_2 de reinicio de columnas a eliminar (Pb).

Tabla 15. Parámetros de los métodos seleccionados

Parámetros Métodos de inicialización			
Método	**Parámetro**	**Descripción**	**Valor**
Random	R	Probabilidad de adicionar una columna a una nueva solución	0.1
RHeuristic	M	Numero de mejores filas	10
	N	Numero de mejores columnas	5
IterConstruct	Sin parámetros		
Parámetros Métodos de búsqueda local			
Método	**Parámetro**	**Descripción**	**Valor**
IterLS	COL_DROP_1	Número máximo de columnas a eliminar aleatoriamente si $N > 35$	12
IterLS	COL_DROP_2	Número máximo de columnas a eliminar aleatoriamente si $N \leq 35$	5
IterLS	Pb	Probabilidad de adicionar una columna a un grupo de reinicio de columnas L_1 y L_2	0.5
RWLS	Sin parámetros		

La Tabla 16 muestra la designación de las variantes obtenidas al modificar el algoritmo ABC_SCP_IMP con los métodos de inicialización y búsqueda local seleccionados. Por ejemplo, ABC_SCP_IMP_RM_ILS identifica al algoritmo modificado que aplica los métodos Random y RWLS, ABC_SCP_IMP_RM_RLS al que aplica los métodos Random y RWLS; de esta forma se designan las demás variantes. Estas designaciones serán consideradas en las comparaciones de los resultados obtenidos que se realizará a continuación.

Tabla 16. Variantes del algoritmo ABC_SCP modificado

Método de inicialización	**Método de búsqueda local**	
	IterLS	RWLS
Random	ABC_SCP_IMP_RM_ILS	ABC_SCP_IMP_RM_RLS
RHeuristic	ABC_SCP_IMP_RH_ILS	ABC_SCP_IMP_RH_RLS
IterConstruct	ABC_SCP_IMP_IC_ILS	ABC_SCP_IMP_IC_RLS

3.2.2.4. Resultados de variantes que aplican el método RWLS

Los resultados de las variantes identificadas que aplican el método de búsqueda local basado en ponderación de filas RWLS se muestran en la Tabla 17, los cuales son comparados con los resultados del algoritmo ABC_SCP_IMP. En esta tabla se observa el mejor valor de aptitud conocido de cada problema (*BKS*), el mejor obtenido en las ejecuciones (*Best*) y el valor promedio de las ejecuciones (*Avg*). En estos se resaltan los resultados que fueron mejorados, y los más cercanos al valor *Best* del algoritmo implementado en los casos que no fue mejorado.

Tabla 17. Resultados variantes que integran el método de búsqueda local RWLS

Problema	BKS	ABC_SCP_IMP		ABC_SCP_IMP _RM_RLS		ABC_SCP_IMP _RH_RLS		ABC_SCP_IMP _IC_RLS	
		Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg
NRE1	29	29	29	29	29	29	29	29	29
NRE2	30	30	30	30	30	30	30	30	30
NRE3	27	27	27	27	27	27	27	27	27
NRE4	28	28	28	28	28	28	28	28	28
NRE5	28	28	28	28	28	28	28	28	28
NRF1	14	14	14	14	14	14	14	14	14
NRF2	15	15	15	15	15	15	15	15	15
NRF3	14	14	14	14	14	14	14	14	14
NRF4	14	14	14	14	14	14	14	14	14
NRF5	13	13	13.6	13	13.6	13	**13.2**	13	**13.2**
NRG1	176	176	176	176	176	176	176	176	176
NRG2	154	155	155	155	155	154	**154.8**	155	155
NRG3	166	166	166.2	**166**	**167.5**	**166**	**167.5**	167	168.7
NRG4	168	168	168	169	169.4	168	169.2	168	**168.9**
NRG5	168	168	168	168	168	168	168	168	168
NRH1	63	63	63.9	64	64	64	64	63	**63.8**
NRH2	63	63	63.9	63	**63.8**	63	**63.7**	63	**63.6**
NRH3	59	59	59	59	59	59	59	59	59
NRH4	58	58	58	58	58	58	58	58	58
NRH5	55	55	55	55	55	55	55	55	55

En los resultados de las variantes que aplican el método de búsqueda local RWLS se puede observar lo siguiente:

- En los problemas NRE1 a NR5, NRF1 a NRF4, NRG1, NRG5, y NRH3 a NRH5 las variantes lograron valores *Best* y *Avg* iguales al *BKS*.

- En el problema NRF5, las variantes ABC_SCP_IMP_RH_RLS y ABC_SCP_IMP_IC_RLS logran mejorar el valor *Avg* que obtuvo el algoritmo base (13.6 a 13.2); por otra parte, la variante ABC_SCP_IMP_RM_RLS logra igualar los valores del algoritmo ABC_SCP_IMP.

- En el problema NRG2 se destaca la variante ABC_SCP_IMP_RH_RLS, que logra mejorar el valor *Avg* del algoritmo base (155 a 154.8), las variantes ABC_SCP_IMP_RM_RLS y ABC_SCP_IMP_IC_RLS logran el valor *Best* y *Avg* igual al reportado por el algoritmo base (155).

- En los problemas NRG3 y NRG4, ninguna de las tres variantes logró mejorar el *Best* y *Avg* del algoritmo base, para el problema NRG3 el mejor valor *Best* (166) y *Avg* (167.5) fue

logrado por igual por las variantes ABC_SCP_IMP_RM_RLS y ABC_SCP_IMP_RH_RLS, y para el problema NRG4 fue logrado por la variante ABC_SCP_IMP_IC_RLS (168.9).

- En el problema NRH1, se destaca que el algoritmo ABC_SCP_IMP_IC_RLS logró mejorar el resultado (63.9 a 63.8), las otras variables no lograron superar ni igualar el resultado del algoritmo ABC_SCP_IMP (64).

- En el problema NRH2 las tres variantes logran mejorar el resultado del algoritmo ABC_SCP_IMP (63.9), siendo el mejor resultado el del algoritmo ABC_SCP_IMP_IC_RLS (63.6).

En los resultados de las variantes de aplican el método RWLS se puede observar que las tres variantes obtienen resultados óptimos para los problemas de prueba del repositorio OR-Library. Se destaca la variante ABC_SCP_IMP_RH_RLS, la cual logra resultados *Best* y *Avg* iguales a los *BKS* en 14 problemas (NRE1 a NR5, NRF1 a NRF4, NRG1, NRG5, NRH3, NRH4, NRH5), mejores resultados en 3 problemas (NRF5, NRG2 y NRH2) y resultados cercanos en tres problemas (NRG3, NRG4, NRH1).

3.2.2.5. Resultados de variantes que aplican el método IterLS

Los resultados obtenidos de las variantes que aplican el método de búsqueda local iterada IterLS se muestran en la Tabla 18, los cuales son comparados con los resultados del algoritmo ABC_SCP_IMP. En esta tabla se observa el mejor valor de aptitud conocido de cada problema (*BKS*), el mejor obtenido en las ejecuciones (*Best*) y el valor promedio de las ejecuciones (*Avg*). En estos se resaltan los resultados que fueron mejorados, y los más cercanos al valor *Best* del algoritmo implementado en los casos que no fue mejorado.

En los resultados de las variantes que aplican el método de búsqueda local IterLS se puede observar lo siguiente:

- En los problemas NRE1 a NR5, NRF1 a NRF4, NRG1, NRG5, y NRH3 a NRH5 las variantes logran el valor *Best* y *Avg* igual al *BKS*.
- En el problema NRF5 se destaca que las tres variantes logran mejorar el resultado del algoritmo base al obtener los valores *Best* y *Avg* iguales al *BKS*.
- En el problema NRG2 se destaca la variante ABC_SCP_IMP_RH_ILS que logra mejorar el valor *Avg* (155 a 154.9), las variantes ABC_SCP_IMP_RM_ILS y ABC_SCP_IMP_IC_ILS logran igualar el *Best* y *Avg* del algoritmo ABC_SCP_IMP.
- En los problemas NRG3 y NRG4, ninguna de las tres variantes logró mejorar el *Best* y *Avg* logrado por el algoritmo ABC_SCP_IMP, para el problema NRG3 el mejor valor *Best* (166) y *Avg* (166.9) fue lograron por la variante ABC_SCP_IMP_IC_ILS y para el problema NRG4 fue logrado por la variante ABC_SCP_IMP_RM_ILS (168.9).
- En el problema NRH1, las variantes ABC_SCP_IMP_RM_ILS y ABC_SCP_IMP_RH_ILS mejoraron el valor *Avg* del algoritmo base (63.9 a 63.8) mientras que la variante IterConstr_IterLS logró igualar el valor (63.9).

- En el problema NRH2 las tres variantes logran mejorar el resultado del algoritmo base, destacándose el resultado de la variante ABC_SCP_IMP_IC_ILS al lograr el mejor valor (63.4).

Tabla 18. Resultados de las variantes que integran el método de búsqueda local IterLS

Problema	**BKS**	**ABC_SCP_IMP**		**ABC_SCP_IMP _RM_RLS**		**ABC_SCP_IMP _RH_RLS**		**ABC_SCP_IMP _IC_RLS**	
		Best	**Avg.**	**Best**	**Avg.**	**Best**	**Avg.**	**Best**	**Avg**
NRE1	29	29	29	29	29	29	29	29	29
NRE2	30	30	30	30	30	30	30	30	30
NRE3	27	27	27	27	27	27	27	27	27
NRE4	28	28	28	28	28	28	28	28	28
NRE5	28	28	28	28	28	28	28	28	28
NRF1	14	14	14	14	14	14	14	14	14
NRF2	15	15	15	15	15	15	15	15	15
NRF3	14	14	14	14	14	14	14	14	14
NRF4	14	14	14	14	14	14	14	14	14
NRF5	13	13	13.6	**13**	**13**	**13**	**13**	**13**	**13**
NRG1	176	176	176	176	176	176	176	176	176
NRG2	154	155	155	155	155	**154**	**154.9**	155	155
NRG3	166	166	166.2	166	167.1	166	167.5	**166**	**166.9**
NRG4	168	168	168	**168**	**168.3**	168	168.8	168	168.6
NRG5	168	168	168	168	168	168	168	168	168
NRH1	63	63	63.9	**63**	**63.8**	**63**	**63.8**	63	63.9
NRH2	63	63	63.9	63	63.7	63	63.6	**63**	**63.4**
NRH3	59	59	59	59	59	59	59	59	59
NRH4	58	58	58	58	58	58	58	58	58
NRH5	55	55	55	55	55	55	55	55	55

En los resultados obtenidos por las variantes que aplican el método de búsqueda local IterLS, se puede observar que las tres variantes obtienen resultados óptimos para los problemas de prueba del repositorio OR-Library. Se destaca la variante ABC_SCP_IMP_RH_ILS por haber logrado valores *Best* y *Avg* iguales al *BKS* en 15 problemas (NRE1 a NRE5, NRF1 a NRF5, NRG1, NRG5, NRH3 a NRH5) y haber mejorado los resultados del algoritmo base en tres problemas (NRG2, NRH1 y NRH2).

Con base en los resultados destacados obtenidos por la variante ABC_SCP_IMP_RH_ILS este es seleccionado para realizar un afinamiento de parámetros en busca de mejorar los resultados obtenidos.

Capítulo 4

4. ALGORITMO MODIFICADO: ABC_SCP_IMP_RH_ILS

En este capítulo se presenta el algoritmo de colonia de abejas artificiales propuesto ABC_SCP_IMP_RH_ILS que se obtuvo al modificar el algoritmo base ABC_SCP teniendo en cuenta el proceso definido en el capítulo anterior.

De igual forma se muestra la representación de las soluciones candidatas, la función objetivo utilizada en el PCC, las modificaciones y las mejoras que fueron realizadas al algoritmo ABC_SCP_IMP_RH_ILS. Por último, se muestran las características técnicas de desarrollo utilizadas en la implementación del algoritmo.

4.1. Representación de las soluciones

La representación de soluciones en algoritmos metaheurísticos se refiere a la manera en que se codifican o estructuran las posibles soluciones candidatas de un problema dentro del espacio de búsqueda. Esta representación permite al algoritmo identificar, manipular y buscar soluciones óptimas mediante procedimientos de exploración y explotación.

Teniendo en cuenta lo mencionado anteriormente, la representación de soluciones en el algoritmo ABC_SCP_IMP_RH_ILS para el PCC se realiza mediante un vector de codificación binaria de tamaño n, que es igual a la cantidad de posibles conjuntos de cobertura que cubren los elementos de cobertura (o restricciones) que define un PCC. Una solución define la siguiente estructura $S = \{x_1, x_2, \dots, x_j, \dots, x_n\}$ en donde cada posición x_j del vector es un valor binario 1 o 0 que representa si el conjunto x_j hace parte de la solución (valor 1) o no hace parte (valor 0). Por ejemplo, si la cantidad de posibles conjuntos de cobertura es diez ($n = 10$), y el número de elementos a cubrir es diez ($m = 10$), la representación de una posible solución podría ser la siguiente $S = \{1, 0, 1, 0, 0, 1, 0, 1, 0, 0\}$, lo que significa que los conjuntos $\{x_1, x_3, x_6, x_8\}$ cubren por completo los diez elementos o restricciones del problema.

4.2. Función objetivo

Dado un vector solución $S = \{x_1, x_2, \dots, x_j, \dots, x_n\}$ y un vector de costos $C = \{c_1, c_2, \dots, c_j, \dots, c_n\}$ asociados a cada conjunto de cobertura, el resultado de la función objetivo es un valor entero igual a la sumatoria de los costos de los conjuntos c_j que hacen parte de una solución S (ver ecuación (18)) en donde las restricciones del problema deben estar cubiertas por al menos un conjunto (ver ecuación (19)). El problema se resuelve en un dominio binario (ver ecuación (20)).

$$Minimizar\ f(x) = \sum_{j=1}^{n} c_j x_j \tag{18}$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, \forall i \in I \quad (19)$$

$$x_j \in \{0, 1\} \quad (20)$$

El PCC requiere minimizar el valor de la función objetivo, buscando la combinación de conjuntos x_j que obtenga el menor costo total posible.

4.3. Estructura algoritmo ABC_SCP_IMP_RH_ILS

Se realizaron modificaciones al algoritmo base ABC_SCP obteniendo el algoritmo ABC_SCP_IMP_RH_ILS, estas se describen a continuación:

- El método de inicialización de soluciones fue reemplazado por el método de inicialización aleatorio con operador heurístico RHeuristic, implementado según la especificación y parámetros definidos en los estudios primarios en los que se aplica sin requerir modificaciones.

- El método de búsqueda local ABC_SCP fue reemplazado por el método de búsqueda local iterada IterLS, implementado según la especificación del estudio primario. Se requirió adaptar el método para reducir el porcentaje de columnas que son reiniciadas (eliminadas) en una solución para la búsqueda de nuevas soluciones. Se adicionaron tres parámetros: COL_DROP_1 y COL_DROP_2 que define el máximo número de columnas que los grupos de reinicio de columnas L_1 y L_2 pueden contener y Pb que define la probabilidad de incluir una columna en el grupo L_1, siendo $(1 - Pb)$ la probabilidad de incluirla en el L_2.

El algoritmo ABC_SCP_IMP_RH_ILS (ver Figura 13) inicia con la creación de las variables que almacenan la población de fuentes de alimento FA_k, el número de intentos de mejora de una solución $trial_k$ y los valores de aptitud de las fuentes de alimento $fitness_k$, en donde k corresponde a cada fuente de alimento de la población de tamaño $FOOD\_NUM$ (líneas 1-4).

Luego, por cada fuente de alimento FA_k se inicializa una nueva solución aplicando el método de inicialización aleatoria con operador heurístico - RHeuristic (ver sección 2.1.3.2), y se calcula el valor de aptitud de la nueva solución para ser almacenada en $fitness_k$ (líneas 2-11).

Una vez creada las fuentes de alimento se memoriza el mejor valor de aptitud de la población se crean las variables $bestSol$ y $bestFitness$ para almacenar la mejor solución y el mejor valor de aptitud logrados en la ejecución, se inicializan con valores iniciales y se aplica el procedimiento de memorización de la mejor solución global buscando la mejor solución de la población creada (líneas 10-12) que se explica en la sección 4.3.1.

En seguida inicia un ciclo que se ejecuta un máximo de MAX_ITER iteraciones, cada iteración inicia con la fase de abejas empleadas recorriendo las soluciones S asociadas a las fuentes de alimento FA_k (líneas 13-15).

```
   inicio ABC_SCP()
1
2  FA_k ← {∅}; ∀k = {1,2, … , FOOD_NUM}                    // fuentes de alimento
3  trial_k ← 0; ∀k = {1,2, … , FOOD_NUM}                   // intentos de mejora
4  fitness_k ← 0; ∀k = {1,2, … , FOOD_NUM}                 // valores de aptitud
5  Desde k = 1 hasta FOOD_NUM hacer                        // inicialización
6     S ← aplicarMetodoInicializacionRHeuristic()
7     FA_k ← S
8     fitness_k ← calcularAptitud(S)
9  fin Desde
10 bestSol ← FA_0
11 bestFitness ← fitness_0
12 bestSol ← memorizarMejorSolucionGlobal(FA, fitness, bestSol, bestFitness)
13 Desde iter = 0 hasta MAX_ITER hacer
14    Desde k = 1 hasta EMPLOYEE_BEES hacer              // abejas empleadas
15       S ← FA_k
16       rS ← seleccionarSolucionDistinta(FA, S)
17       dCols ← buscarColumnasDistintas(S, rS)
18       Si |dCols| > 0 hacer
19          S' ← modificarSolucion(S, dCols)
20          S' ← aplicarMetodoBusquedaLocalIterLSADO(S')
21          FA_k ← memorizarMejorSolucionLocal(S, S', trial_k, fitness_k)
22       Sino
23          S ← aplicarMetodoInicializacionRHeuristic()
24          FA_k ← S
25          trial_k ← 0
26          fitness_k ← calcularAptitud(S)
27       fin Si
28    fin Desde
29    p_k ← calcularProbabilidades(FA_k); ∀k = {1, 2, … , FOOD_NUM}
30    Desde k = 1 hasta ONLOOKER_BEES hacer              // abejas observadoras
31       S ← seleccionarSolucionMetodoRuleta(p_k, FA)
32       dCols ← seleccionarSolucionColumnasDistintas(FA, S)
33       S' ← modificarSolucion(S, dCols)
34       S' ← aplicarMetodoBusquedaLocalIterLSADO(S')
35       FA ← memorizarMejorSolucionLocal(S, S', trial_k, fitness_k)
36    fin Desde
37    bestSol ← memorizarMejorSolucionGlobal(FA)
38    Desde k = 1 hasta FOOD_NUM hacer                   // abejas exploradoras
39       Si trial_k ≥ LIMIT hacer
40          S ← aplicarMetodoInicializacionRHeuristic()
41          FA_k ← S
42          trial_k ← 0
43          fitness_k ← calcularAptitud(S)
44       fin Si
45    fin Desde
46 fin Desde
47 fin
```

Figura 18. Algoritmo modificado ABC_SCP_IMP_RH_ILS

Por cada S se selecciona otra solución $randS$ de forma aleatoria, se buscan columnas en $randS$ que sean distintas a las que contiene S, se almacenan en la lista $dCols$ y, si el número de $dCols$ es mayor a cero, se procede a aplicar un procedimiento de modificación de S para obtener una

nueva solución S' (ver sección 4.3.2) (líneas 16-19), la cual se intenta mejorar con la aplicación del método de búsqueda local IterLS adaptado (ver sección 4.3.3); la solución S' obtenida se compara con la solución S actual aplicando un procedimiento de memorización de soluciones local (ver sección 4.3.3).

Si en la búsqueda de columnas distintas entre soluciones no se encuentra al menos una, se identifica que existe una colisión, lo que significa que estas son iguales [32]. Con el fin de evitar esta situación y aportar a la diversidad de soluciones presentes en la población FA, en los casos de colisión la abeja empleada se convierte en exploradora abandonando la fuente de alimento actual, explora el espacio de búsqueda y selecciona una nueva fuente para convertirse de nuevo en empleada. En el algoritmo ABC_SCP_IMP_RH_ILS (ver Figura 13) esto puede verse en el caso en el que la lista $dCols$ es igual a cero (línea 18), reemplazando la solución actual S con la generación de una nueva (líneas 22-28).

Se calculan las probabilidades de selección de las fuentes de alimento FA aplicando por cada una la ecuación (4) para iniciar con la fase de abejas exploradoras seleccionando una solución mediante el método de la ruleta (ver sección 4.3.5) a partir de las probabilidades p_k calculadas (líneas 29-31). Luego se busca una solución S' que contenga columnas distintas a las de S y estas se almacenan en S' (líneas 31-32); a diferencia de la fase de abejas empleadas que detecta colisiones en la búsqueda de soluciones con columnas distintas, en esta fase se busca una solución S' hasta que esta contenga columnas distintas a las de S, descartando las colisiones que se puedan encontrar. Una vez se encuentren la lista $dCols$ de columnas distintas se aplican los procedimientos de modificación de la solución (ver sección 4.3.2), aplicación del método de búsqueda local Iterada adaptado (ver sección 4.3.3) y memorización de mejor solución local (ver sección 4.3.4) de la fase de abejas empleadas (líneas 33-36).

Luego se buscan en la población de fuentes de alimento una mejor solución global aplicando de nuevo el procedimiento de memorización de mejor solución global (ver sección 4.3.1), e inicia la fase de abejas exploradoras recorriendo las fuentes de alimento buscando las que tienen un numero de intentos mayor al parámetro $LIMIT$; las fuentes que cumple la condición son abandonadas y sustituidas por nuevas fuentes que se asocia con la inicialización de nuevas soluciones con el método RHeuristic y su almacenamiento en la fuente FA_k correspondiente, se calcula el valor de aptitud de cada solución generada y se almacena en la lista $fitness_k$ y se reinicia el número de intentos de mejora $trial_k$ a cero (líneas 37-47). El algoritmo finaliza una iteración y realiza nuevas iteraciones hasta cumplir con el número máximo de iteraciones (MAX_ITER).

4.3.1. Memorización de mejor solución global

La Figura 14 muestra en detalle el procedimiento de memorización de mejor solución global, en el que se recorre cada fuente de alimento y se comparan los valores de aptitud de cada fuente con el valor $bestFitness$ inicial; en caso de encontrar una mejor solución los valores $bestSol$ y $bestFitness$ son reemplazados por los valores de la solución encontrada (líneas 1-6). En caso de encontrar soluciones con igual valor de aptitud al actual, se procede a calcular el valor de aptitud de la función secundaria (ver ecuación (5)) de la mejor solución global actual $bestFitness$ y la

solución S, buscando la que contenga columnas que cubran menor cantidad de filas, reemplazando el valor $bestSol$ en caso de encontrarla (líneas 7-15). El procedimiento retorna la mejor solución y el mejor valor de aptitud de la ejecución (líneas 16-17).

```
1  inicio memorizarMejorSolucionGlobal(FA, fitness, bestSol, bestFitness)
2  Desde k = 1 hasta FOOD_NUM hacer
3      S ← FA_k
4      Si fitness_k < bestFitness hacer
5          bestFitness ← fitness_k
6          bestSol ← S
7      Sino Si fitness_k == bestFitness hacer
8          fitnessSec_1 ← calcularAptitudFuncionSecundaria(S)
9          fitnessSec_2 ← calcularAptitudFuncionSecundaria(bestSol)
10         Si fitnessSec_1 < fitnessSec_2 hacer
11             bestFitness ← fitness_k
12             bestSol ← S
13         fin Si
14     fin Si
15 fin Desde
16 retornar bestSol, bestFitness
17 fin
```

Figura 19. Procedimiento de memorización mejor solución global

4.3.2. Modificación de soluciones

La Figura 15 muestra el procedimiento de modificación de soluciones, este inicia creando una copia S' de la solución S actual y se valida el número de columnas que contiene; si S' contiene más de 35 columnas se procede a adicionar un máximo de COL_ADD_1 columnas de $dCols$ a la solución, de lo contrario se adicionan un máximo de COL_ADD_2 columnas (líneas 1-7); luego se procede a eliminar aleatoriamente COL_DROP_1 columnas distintas a las adicionadas si el número de columnas de S' es mayor a 35, de lo contrario se eliminan COL_DROP_2 columnas, y se valida la factibilidad de S' (si cumple con las restricciones del problema) identificando la lista de filas I_u que no están cubiertas por S' (líneas 8-13). En caso de encontrar columnas no cubiertas (tamaño de I_u mayor a cero) se aplica un sub-procedimiento para reparar la solución que recorre cada fila $i \in I_u$, identifica la lista α_i de columnas que pueden cubrir a i, genera un numero aleatorio en el rango [0, 1] y se valida si es menor que el parámetro Pa (líneas 14-19). Si la condición se cumple, por cada columna de α_i se obtienen las filas de I_u que pueden cubrir y se almacenan en I_{uj}, se calcula el valor de la relación ($ratio$) entre el costo de cada columna de α_i y el numero de filas I_{uj}, y se busca la columna j con menor valor de $ratio$ (líneas 20-25); si la condición no se cumple se selecciona aleatoriamente una columna de la lista RC de columnas candidatas de α_i (líneas 26-28). La columna j seleccionada se adiciona a la solución S' y se actualizan las filas no cubiertas I_u, esta reparación se aplica hasta que la solución sea factible ($I_u = \emptyset$) retornando la solución S' obtenida (líneas 29-34).

```
1  inicio modificarSolucion(S, dCols)
2  S' ← S
3  Si |S'| > 35 hacer
4      S' ← adicionarColumnas(S', dCols, COL_ADD_1)
5  Sino
6      S' ← adicionarColumnas(S', dCols, COL_ADD_2)
7  fin Si
8  Si |S'| > 35 hacer
9      S' ← eliminarColumnas(S', COL_DROP_1)
10 Sino
11     S' ← eliminarColumnas(S', COL_DROP_2)
12 fin Si
13 I_u ← buscarFilasNoCubiertas(S')
14 Si |I_u| > 0 hacer      // inicia reparación de la solución
15     j ← 0
16     Mientras I_u ≠ {∅} hacer
17         i ← obtenerPrimerFila(I_u)
18         α_i ← buscarColumnasQuePuedenCubrirFila(i)
19         random ← generarAleatorio(0.1)
20         Si random ≤ Pa hacer
21             ratio_j ← 0; ∀j ∈ α_i
22             Para cada j ∈ α_i hacer
23                 I_uj ← buscarFilasNoCubiertasQuePuedeCubrirColumna(I_u, j)
24                 ratio_j ← c_j/|I_uj|
25             fin Para cada
26             j ← buscarColumnaMenorRatio(α_i, ratio_j)
27         Sino
28             j ← buscarColumnaListaRC(α_i, RC_SIZE)
29         fin Si
30         S ← S ∪ j
31         I_u ← actualizarFilasNoCubiertas(S')
32     fin Mientras
33 fin Si
34 retornar S'
35 fin
```

Figura 20. Procedimiento de modificación de soluciones

4.3.3. Método de búsqueda local IterLS

Con base en los experimentos realizados con el algoritmo ABC_SCP_IMP_RH_ILS se realizan adaptaciones adicionales a las realizadas en la modificación del algoritmo ABC_SCP_IMP (ver sección 3.2.2.2) la cuales se describen a continuación:

1. En el procedimiento de agrupamiento de columnas ($agruparColumnas\ (L)$) se modifica como se muestra en la Figura 16, este inicia con la creación de las listas L_1 y L_2 en donde se almacenan las columnas que serán reiniciadas (eliminadas) en una solución, y la definición del máximo número de columnas que serán agrupadas en estas listas (líneas 1-7); se reúsan los parámetros COL_DROP_1 y COL_DROP_2 del procedimiento de modificación de soluciones (ver sección 4.3.2). Luego se inicia un ciclo hasta adicionar el número de columnas a reiniciar ($maxCols$) en las listas de reinicio, seleccionando una columna de la

solución S y validando que no haya sido adicionadas en alguna de las listas L_1 y L_2 (líneas 9-10); si esto se cumple se procede a generar un numero aleatorio en el rango [0,1] y si es menor al parámetro Pb, la columna j es adicionada a la lista L_1, de lo contrario se adiciona a la lista L_2 y se decrementa el valor de $maxCols$ en uno para iniciar con una nueva iteración (líneas 11-18). Al final el procedimiento retorna las listas de reinicio L_1 y L_2 que son procesadas por el método IterLS (líneas 19-20).

```
1  inicio agruparColumnas(S)
2  L1, L2 ← {∅}
3  Si |S| > 35 hacer
4    maxCols ← COL_DROP_1
5  Sino
6    maxCols ← COL_DROP_2
7  fin Si
8  Mientras maxCols > 0 hacer
9    j ← seleccionarColumnaAleatoria(S)
10   Si j ∉ L1 y j ∉ L2 hacer
11     random ← generarAleatorio(0,1)
12     Si random < Pb hacer
13       L1 ← L1 ∪ j
14     Sino
15       L2 ← L2 ∪ j
16     fin Si
17     maxCols = maxCols − 1
18   fin Si
19 fin Mientras
20 retornar L1, L2
21 fin
```

Figura 21. Procedimiento para agrupar columnas adaptado.

2. Se adicionó el procedimiento de penalización de columnas [40] que consiste en añadir una variable de penalización $penalty_j = 0;\ \forall j \in J$ que es utilizada de la siguiente forma:

   (i) Se suma el valor $\alpha = 0.05$ a $penalty_j$ cuando una columna j es adicionada a una solución. En el caso que el valor de penalización de la columna sea igual a 1.5 ($penalty_j = 1.5$), la variable se mantiene en este valor.

   (ii) Se resta el valor $\beta = 0.01$ cuando una columna j es eliminada de una solución. En el caso que el valor de penalización de la columna sea igual a 0.5 ($penalty_j = 0.5$), la variable se mantiene en este valor.

   (iii) Se modifica la función de evaluación de columnas para adicionar la variable de penalización (ver ecuación (21))

$$W_j = \frac{c_j}{I_{uj}} * penalty_j \tag{21}$$

Las propuestas [11] y [40] destacan la importancia de ponderar las filas o columnas en el proceso de modificación y generación de nuevas soluciones, buscando explorar el espacio

de búsqueda de manera eficiente evitando ciclos en los que se adicionen o eliminen las mismas columnas en cada modificación.

4.3.4. Memorización de mejor solución local

La Figura 16 muestra el procedimiento que memoriza compara una solución actual S y una solución modificada S' y selecciona la mejor solución para almacenarla en la población de fuentes de alimento. Este procedimiento inicia con el cálculo del valor de aptitud $fitnessSol$ de la solución S' para compararse con el valor de actitud de la solución S actual; si es menor, se realiza el reemplazo del valor actual en la lista $fitness_k$ con el nuevo valor de aptitud, de la solución FA_k con S' y se reinicia el número de intentos de mejora $trial_k$ a cero (líneas 1-7), y si no es mejorada se incrementa el número de intentos de mejora $trial_k$ de la solución S (líneas 17-19). En el caso en que los valores de aptitud de S' y S sean iguales, se procede a calcular el valor de aptitud de la función secundaria (ver ecuación (5)) de estas soluciones, buscando la que contenga columnas que cubran menor cantidad de filas; si el valor de aptitud secundado de S' sea menor que S se procede a reemplazar la solución FA_k y se reinicia el numero de intentos de la solución ($trial_k$) (líneas 8-13), de lo contrario se incrementa el numero de intentos (líneas 14-16).

```
1  inicio memorizarMejorSolucion(S, S', k, fitness, FA, trial, p_k)
2  fitnessSol ← calcularAptitud(S')
3  Si fitnessSol < fitness_k hacer
4     fitness_k ← fitnessSol
5     FA_k ← S'
6     trial_k ← 0
7     p_k ← calcularProbabilidades(FA)
8  Sino Si fitnessSol == fitness_k hacer
9     fitnessSec_1 ← calcularAptitudFuncionSecundaria(S')
10    fitnessSec_2 ← calcularAptitudFuncionSecundaria(S)
11    Si fitnessSec_1 < fitnessSec_2 hacer
12       FA_k ← S'
13       trial_k ← 0
14    Sino
15       trial_k ← trial_k + 1
16    fin Si
17 Sino
18    trial_k ← trial_k + 1
19 fin Si
20 fin
```

Figura 22. Procedimiento de memorización mejor solución local

4.3.5. Selección de soluciones: Método de la ruleta

La Figura 18 muestra la selección de soluciones asociadas a las fuentes de alimento FA mediante el método de la ruleta utilizado en algoritmos genéticos [53] para seleccionar soluciones con probabilidad proporcional a la probabilidad acumulada de dichas soluciones. Este inicia con la creación de una variable auxiliar p_{aux} que almacena la sumatoria de las probabilidades de las fuentes FA (probabilidad acumulada) y una solución S' vacía (líneas 1-3). Se genera un numero

aleatorio $random$ en el rango [0,1] y se recorren las soluciones de la población sumando las probabilidades de selección p_k de cada solución k a la variable p_{aux} hasta que el valor $random$ sea menor a p_{aux}, seleccionando la solución que corresponde a la última probabilidad que fue sumada (líneas 4-11).

```
1  inicio seleccionarSolucionMetodoRuleta(p_k, FA)
2  p_aux ← 0
3  S' ← {∅}
4  random ← generarAleatorio(0, 1)
5  Desde k = 1 hasta FOOD_NUM hacer
6     p_aux ← p_aux + p_k
7     Si random ≤ p_aux hacer
8        S' ← FA_k
9        terminar e ir a Linea 12
10    fin Si
11 fin Desde
12 retornar S'
13 fin
```

Figura 23. Procedimiento selección de soluciones mediante el método de la ruleta

Un resumen de las modificaciones realizadas al algoritmo ABC_SCP_IMP y la razón de cada una de ellas se muestran en la Tabla 19.

Tabla 19. Modificaciones algoritmo ABC_SCP_IMP

Modificación realizada	Lugar del algoritmo	Motivo
Reemplazo método de inicialización Rheuristic.	Inicialización de la población. Fase de abejas empleadas al detectar colusiones.	Generar soluciones mediante un método que recorre de forma alternativa las restricciones del problema para adicionar columnas.
Reemplazo método de búsqueda local IterLS.	Fase de abejas empleadas. Fase de abejas observadoras.	Realizar explotación inteligente de las soluciones mediante un método codicioso que penaliza la adición y eliminación de columnas.

4.4. Características generales de la implementación

El algoritmo base ABC_SCP_IMP fue implementado tomando como base el proyecto codificado en el lenguaje de programación Java disponible en el sitio web del algoritmo colonia de abejas artificiales ABC publicado por su autor [54], siendo adaptado en la presente investigación a la suite de desarrollo JDK en su versión 17.0.9.

La implementación del algoritmo se estructura en un modelo de clases orientado a objetos, compuesto por los paquetes *Configuration* en donde se establece la configuración de ejecución del algoritmo, *Initialization* que contiene la implementación de los métodos de inicialización, *Localsearch* que contiene la implementación de los métodos de búsqueda local y un paquete principal *Main* que se encarga de crear las instancias de acuerdo con la variante requerida e inicial la ejecución del algoritmo. El diagrama de clases se puede ver en la Figura 24.

La ejecución del algoritmo se realizó en múltiples hilos, uno por cada ejecución definida haciendo uso de la clase *ForkJoinPool* del paquete *java.util.concurrent*. Esta clase permitió controlar aspectos como el inicio, la detención al lograr los *BKS* de los problemas y recolectar los resultados requeridos mediante una interfaz de consola que se muestra en la Figura 25.

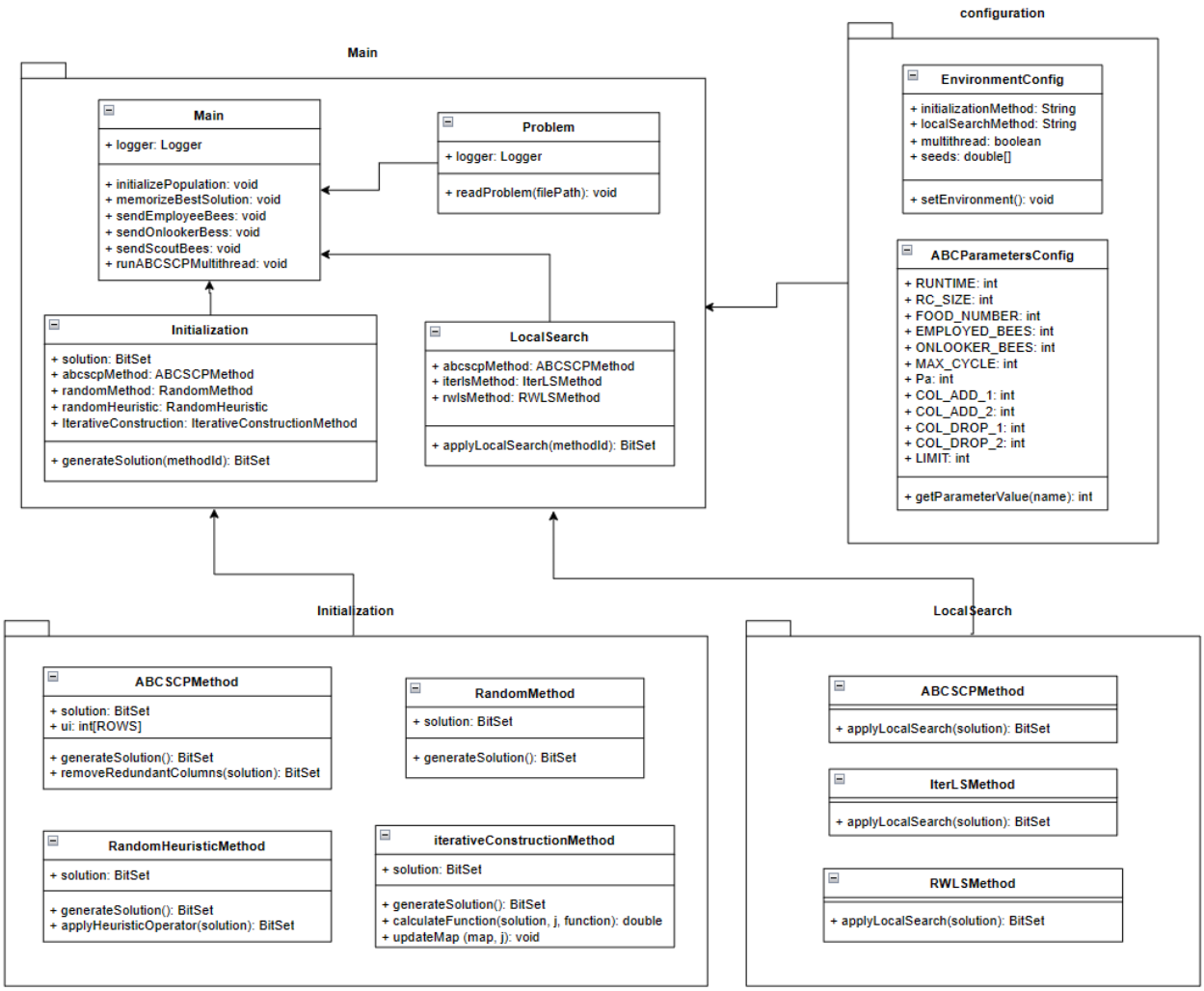


Figura 24. Diagrama de clases algoritmo ABC_SCP_IMP

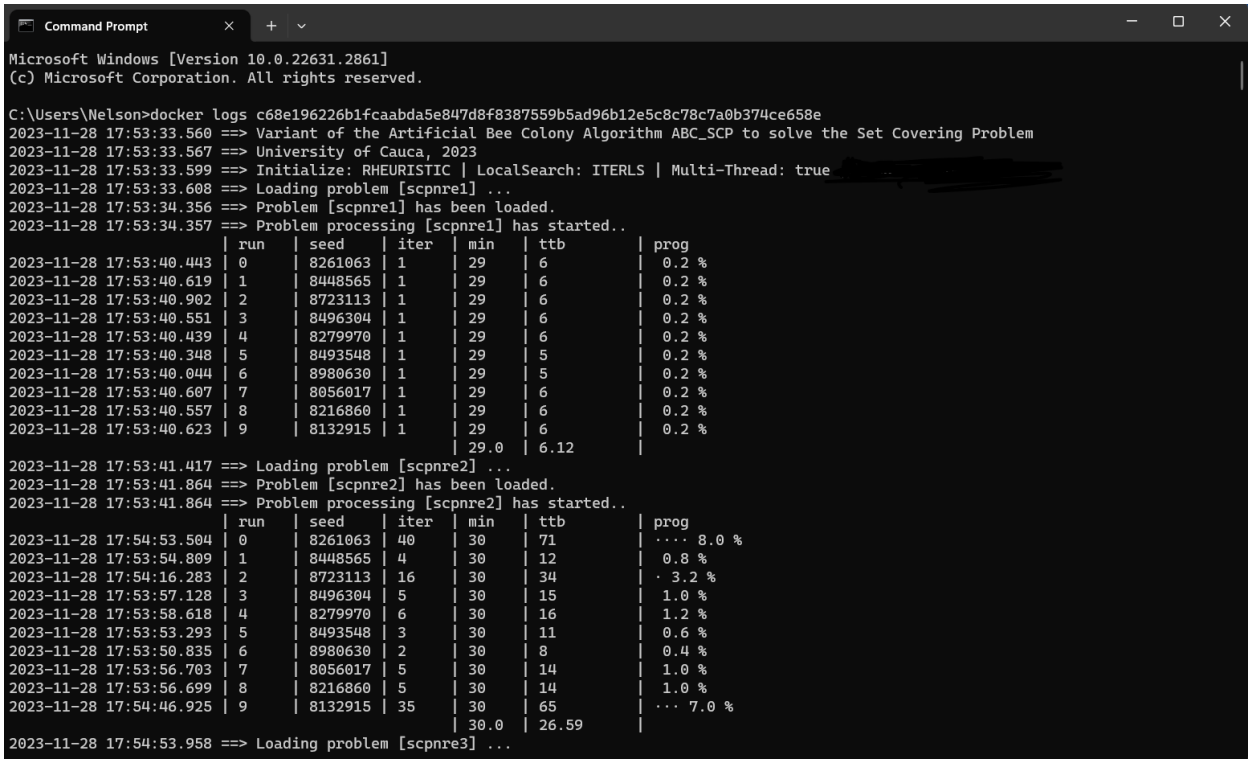


Figura 25. Interfaz de consola ejecución algoritmo ABC_SCP_IMP

Para un uso eficiente de los recursos computacionales, se utilizó la tecnología de contenedores Docker, lo cual permitió ejecutar algunos experimentos en paralelo considerando la disminución de la velocidad de procesamiento en estos. La Figura 26 muestra la configuración del archivo de despliegue *Dockerfile* en el que utiliza la imagen *eclipse-temurin:17-jre-alpine* y se configura la zona horaria a America/Bogota (líneas 1-2), se copia el contenido del proyecto java (*abcscpimp*) al contenedor (líneas 3-5) y se define el comando que se ejecuta al arrancar el contenedor (línea 7).

Dockerfile

C: > Users > Nelson > Documents > JetBrains > IntelliJ > vabcscp > deploy > Dockerfile

```
1  FROM eclipse-temurin:17-jre-alpine
2  ENV TZ="America/Bogota"
3  WORKDIR /usr/local/
4  COPY abcscpimp /usr/local/abcscpimp
5  WORKDIR /usr/local/abcscpimp
6
7  CMD ["java", "main/java/Main"]
8
```

Figura 26. Contenido archivo Dockerfile de configuración contenedor Docker

Los comandos para la creación de la imagen base y el arranque del contenedor se muestran en la Tabla 20.

Tabla 20. Comandos de ejecución contenedores Docker

Tarea	Comando
Creación de imagen	`$ docker build -t <nombre_nueva imagen> <ruta_dockerfile>`
Creación y arranque de contenedor	`$ docker run <nombre_nueva_imagen>`

Capítulo 5

5. EVALUACION

Este capítulo se describen los conjuntos de datos, los parámetros y las métricas usadas en el proceso experimental llevado a cabo para obtener los resultados del algoritmo propuesto ABC_SCP_IMP_RH_ILS y realizar su evaluación. Por último, se comparan los resultados de este algoritmo frente a otros algoritmos del estado del arte.

5.1. Conjunto de datos

OR-Library [9] es un repositorio que contiene el subconjunto de datos de prueba usado en este proyecto para problemas de investigación de operaciones (OR) entre los cuales, se encuentra el conjunto de datos scp que contiene problemas de conjuntos de cobertura que representan de forma general situaciones de la realidad. La Tabla 21 muestra el listado de problemas que conforman el conjunto scp, en donde se identifica los grupos de problemas, el número de problemas que los conforman, el número de filas (m) y de columnas (n) que los componen. En la presente investigación se resuelven los problemas denominados de gran escala, conformados por los grupos NRE, NRF, NRG y NRH que contiene la mayor cantidad de filas y columnas.

Tabla 21. Grupos de PCC del repositorio OR-Library

Grupo	Número de problemas	Filas (m)	Columnas (n)
4	10	200	1000
5	10	200	2000
6	5	200	1000
A	5	300	3000
B	5	300	3000
C	5	400	4000
D	5	400	4000
NRE	5	500	5000
NRF	5	500	5000
NRG	5	1000	10000
NRH	5	1000	10000

5.2. Descripción de las métricas

Las métricas consideradas en este proyecto miden la calidad de las soluciones obtenidas en las ejecuciones del algoritmo propuesto. Una de estas es el valor de aptitud mínimo logrado (*Best*) el cual se identifica en los resultados obtenidos al culminar el total de ejecuciones definido.

La otra métrica corresponde al promedio de los mejores valores de aptitud logrados en cada ejecución realizada (*Avg*) que se calcula con la ecuación (22):

$$Avg = \frac{\sum_{r=1}^{RUNTIME} best_r}{RUNTIME} \quad (22)$$

Donde $RUNTIME$ es el parámetro que define el número de ejecuciones del algoritmo y $best_r$ es el mejor valor de aptitud logrado en cada ejecución r. La ecuación busca el cociente entre la suma los mejores valores de aptitud logrados en cada ejecución y el número de ejecuciones realizado (valor de aptitud promedio).

5.3. Afinamiento de parámetros

En las secciones 3.2.1.2 y 3.2.2.3 se definieron los parámetros que lograron los mejores resultados en las ejecuciones de las variantes del algoritmo modificado. Para la ejecución del algoritmo propuesto modificado ABC_SCP_IMP_RH_ILS se define realizar un afinamiento de algunos parámetros basados en los experimentos preliminares, estos se muestran en la Tabla 22. En la fase de abejas exploradoras se reduce el número de abejas a 50 buscando disminuir la cantidad de ciclos y reducir el costo computacional del algoritmo. En el método de búsqueda local IterLS se incrementa el número máximo de columnas a eliminar con el fin de aumentar la diversidad en la obtención de nuevas soluciones por parte del método; para esto, el parámetro COL_DROP_1 se define en 20 columnas y COL_DROP_2 en 6.

Tabla 22. Afinamiento de parámetros algoritmo ABC_SCP_IMP_RH_ILS

Parámetros algoritmo modificado			
Paso	**Parámetro**	**Descripción**	**Valor**
Fase abejas observadoras	$ONLOOKER\_BEES$	Numero de abejas observadoras	50
Parámetros Método de búsqueda local IterLS adaptado			
Método	**Parámetro**	**Descripción**	**Valor**
IterLS	COL_DROP_1	Número de columnas a eliminar aleatoriamente si $N > 35$	20
IterLS	COL_DROP_2	Numero de columnas a eliminar aleatoriamente si $N \leq 35$	6

5.4. Evaluación de resultados

La Tabla 23 muestra los resultados del algoritmo ABC_SCP_IMP_RH_ILS y se comparan con los resultados del algoritmo ABC_SCP_IMP. En esta tabla se presenta los resultados de los 20 problemas de prueba con respecto al mejor valor de aptitud conocido de cada problema (*BKS*), el mejor valor obtenido en las ejecuciones (*Best*) y el valor promedio (*Avg*, Average por su abreviatura en inglés).

Tabla 23. Resultados algoritmo ABC_SCP_IMP_RH_ILS

Problema	BKS	ABC_SCP base		ABC_SCP_IMP		ABC_SCP_IMP_RH_ILS	
		Best	Avg	Best	Avg	Best	Avg
NRE1	29	29	29	29	29	29	29
NRE2	30	30	30	30	30	30	30
NRE3	27	27	27	27	27	27	27
NRE4	28	28	28	28	28	28	28
NRE5	28	28	28	28	28	28	28
NRF1	14	14	14	14	14	14	14
NRF2	15	15	15	15	15	15	15
NRF3	14	14	14	14	14	14	14
NRF4	14	14	14	14	14	14	14
NRF5	13	13	13.7	13	13.6	**13**	**13**
NRG1	176	176	176	176	176	176	176
NRG2	154	155	155	155	155	**154**	**154**
NRG3	166	**166**	**166**	166	166.2	166	166.1
NRG4	168	168	168	168	168	168	168
NRG5	168	168	168	168	168	168	168
NRH1	63	63	63.9	63	63.9	**63**	**63.1**
NRH2	63	63	63.9	63	63.9	**63**	**63**
NRH3	59	59	59	59	59	59	59
NRH4	58	58	58	58	58	58	58
NRH5	55	55	55	55	55	55	55

Como se observa en la Tabla 23, el algoritmo ABC_SCP_IMP_RH_ILS logra valores *Avg* iguales a los *BKS* en 18 problemas de prueba (NRE1 a NRE5, NRF1 a NRF5, NRG1, NRG2, NRG4, NRG5, NRH2 a NRH5) y resultados cercanos en dos problemas (NRG3, NRH1). Se destacan los resultados de tres problemas (NRF5, NRG2, NRH2) en los que ABC_SCP_IMP_RH_ILS mejora considerablemente los resultados del algoritmo base y el algoritmo base implementado logrando el valor *Avg* igual al *BKS*. También el resultado del problema NRH1 que es mejorado con respecto a los otros dos algoritmos con un valor *Avg* de mayor cercanía al *BKS*. Por último, el resultado del problema NRG3, el cual es mejorado con respecto a ABC_SCP_IMP, pero no supera al valor del algoritmo base.

5.5. Comparación con otros algoritmos

Con el fin de evaluar el desempeño del algoritmo ABC_SCP_IMP_RH_ILS, se realiza la comparación de resultados con otros reportados por algoritmos del estado del arte (ver Tabla 24): Optimización con enjambre de partículas saltadoras (JSPO, Jumping Particle Swarm Optimization por sus siglas en inglés) [55] y algoritmo memético codificado híbrido (HEMA, Hybrid Encoded Memetic Approach por sus siglas en inglés) [11]. Los resultados del algoritmo JSPO

solo reportan el valor de aptitud promedio obtenido (*Avg*), el cual es utilizado en la comparación. Considerando que los algoritmos del estado del arte realizaron 30 ejecuciones para obtener los resultados reportados, el algoritmo ABC_SCP_IMP_RH_ILS es ejecutado el mismo número de veces ($RUNTIME$) con el fin de realizar la comparación en condiciones de ejecución similares.

Tabla 24. Comparación de resultados con otros algoritmos del estado del arte

Problema	**BKS**	**JSPO**		**HEMA**		**ABC_SCP_ IMP_RH_ILS**	
		Best	**Avg**	**Best**	**Avg**	**Best**	**Avg**
NRE1	29	-	29	29	29	29	29
NRE2	30	-	30	30	30	30	30
NRE3	27	-	27	27	27	27	27
NRE4	28	-	28	28	28	28	28
NRE5	28	-	28	28	28	28	28
NRF1	14	-	14	14	14	14	14
NRF2	15	-	15	15	15	15	15
NRF3	14	-	14	14	14	14	14
NRF4	14	-	14	14	14	14	14
NRF5	13	-	13	13	13	13	13
NRG1	176	-	176	176	176	176	176
NRG2	154	-	155	154	154.2	**154**	**154**
NRG3	166	-	167.2	166	166.6	**166**	**166.13**
NRG4	168	-	168.2	168	168.2	**168**	**168**
NRG5	168	-	168	168	168	168	168
NRH1	63	-	64	**63**	**63**	63	63.13
NRH2	63	-	63	63	63	63	63
NRH3	59	-	59.2	59	59	59	59
NRH4	58	-	58.3	58	58	58	58
NRH5	55	-	55	55	55	55	55

Los resultados de la Tabla 24 muestran que los tres algoritmos logran el valor *Avg* igual al *BKS* en catorce problemas (NRE1 a NRE5, NRF1 a NRF5, NRG1, NRG5, NRH2, NRH5). También, que ABC_SCP_IMP_RH_ILS y HEMA, en dos problemas (NRH3 y NRH4) logran el valor promedio igual al *BKS*, superando a JSPO y que en tres problemas (NRG2, NRG3 y NRG4) ABC_SCP_IMP_RH_ILS supera los resultados de los otros algoritmos, alcanzando el valor de *BKS* en el valor *Avg* en dos problemas (NRG2, NRG4) y obteniendo un mejor valor Avg en uno (NRG3). Por otra parte, en el problema NRH1, el mejor resultado lo reporta el algoritmo HEMA con el valor *Avg* igual al *BKS* seguido del algoritmo ABC_SCP_IMP_RH_ILS y en último lugar el algoritmo JSPO.

Capítulo 6

6. CONCLUSIONES Y TRABAJOS FUTUROS

6.1. CONCLUSIONES

Para resolver el problema de cobertura de conjuntos (PCC), un problema NP-Completo de optimización combinatoria, esta investigación propone el algoritmo ABC_SCP_IMP_RH_ILS, el cual es una modificación del algoritmo base de colonias de abejas artificiales del estado del arte ABC_SCP con el método de inicialización aleatorio con operador heurístico (RHeuristic) para la generación de soluciones a partir de la identificación de mejores filas y columnas, y el método de búsqueda local iterada (IterLS) para aplicar el reinicio y la penalización de columnas en las soluciones e intentar mejorarlas.

En el mapeo sistemático sobre algoritmos aplicados al PCC, se identificaron dos enfoques los heurísticos y metaheurísticos, los cuales recorren los espacios de búsqueda de forma inteligente obteniendo soluciones de calidad con un uso eficiente de recursos computacionales. En el enfoque metaheurístico se identificaron algoritmos nuevos aplicados al PCC y algoritmos que ya se han aplicado pero que son modificados para observar el impacto generado en los resultados, destacándose los algoritmos que reemplazan algún componente, con parámetros que se adaptan durante las ejecuciones y los que aplican esquemas de binarización de dos pasos, buscando conservar los componentes de dominio continuo en uno binario.

Para la selección de los métodos de inicialización y búsqueda local, que serían modificados en el algoritmo base ABC_SCP, se llevó a cabo un mapeo sistemático de la literatura que permitió identificar los métodos que se han aplicado recientemente al PCC. A partir de estos resultados y la aplicación de algunos criterios, fueron seleccionados los métodos de inicialización: Aleatoria (Random), Aleatoria con Operador Heurístico (RHeuristic), Construcción Iterativa de Soluciones (IterConstruct), ya que generan soluciones utilizando componentes de aleatoriedad y redundancia de columnas lo que permite reducir el costo computacional del algoritmo implementado ABC_SCP_IMP. Por otra parte, fueron seleccionados los métodos de búsqueda local basado en ponderación de filas (RWLS) y búsqueda local iterada (IterLS), por reportar los mejores resultados en los estudios primarios en comparación con otros métodos identificados y por disponer de la especificación necesaria para su adaptación al algoritmo base.

En la evaluación del algoritmo ABC_SCP_IMP_RH_ILS con el algoritmo base (ABC_SCP base) y el implementado (ABC_SCP_IMP), se encontró una mejora en el valor de aptitud promedio (*Avg*) de tres problemas (NRF5, NRG2, NRH1, NRH2), logrando igualar el valor *Avg* a la mejor solución conocida (*BKS*) en tres problemas (NRF5, NRG2, NRH2). Esto muestra la efectividad de las modificaciones realizadas al algoritmo base ABC_SCP, adicionando en la inicialización de soluciones una estrategia de identificación de mejores filas y columnas mediante el método RHeuristic y en la búsqueda local de mejores soluciones una estrategia codiciosa con

penalización de columnas mediante el método IterLS. En conjunto, estos métodos agregan componentes que logran mayor diversificación en la evolución del algoritmo base ABC_SCP.

En la evaluación del algoritmo ABC_SCP_IMP_RH_ILS con otros algoritmos del estado del arte se observa la mejorara de los resultados del algoritmo de enjambre de partículas saltadoras (JPSO) en seis problemas (NRG2, NRG3, NRG4, NRH1, NRH3, NRH4) y mejorar los resultados del algoritmo memético codificado híbrido (HEMA) en tres problemas (NRG2, NRG3, NRG4). Con estos resultados se muestra que el ABC_SCP_IMP_RH_ILS es un algoritmo metaheurístico competitivo en el ámbito de investigaciones relacionadas con el PCC.

En la ejecución de los algoritmos, considerando la complejidad de los problemas de gran escala de OR-Library por la cantidad de posibles conjuntos (columnas) y restricciones (filas) se propuso una estrategia compuesta por la ejecución de múltiples hilos (multi-thread) y el uso de contenedores *Docker* que contribuyó al uso eficiente de recursos computacionales y por ende al alcance de los resultados buscados del procesamiento del PCC.

6.2. TRABAJO FUTURO

Incluir otros métodos de inicialización y de búsqueda local del estado del arte con el objetivo de observar el impacto en los resultados obtenidos. Además, modificar el algoritmo ABC_SCP_IMP_RH_ILS con otros componentes como esquemas de binarización, enfoques adaptativos y ajustando parámetros buscando evaluar el comportamiento del algoritmo.

Aplicar el algoritmo ABC_SCP_IMP_RH_ILS a otros problemas de optimización combinatorios y extender su aplicación a problemas de conjuntos de cobertura del mundo real, considerando restricciones y complejidades específicas.

Diseñar interfaces de usuario intuitivas que faciliten la interacción y comprensión de los resultados, promoviendo la adopción de los algoritmos en entornos prácticos.

7. REFERENCIAS

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, third edition*. in Computer science. MIT Press, 2009. [Online]. Available: https://sd.blackball.lv/library/Introduction_to_Algorithms_Third_Edition_(2009).pdf

[2] Y. Deng, Y. Zhang, and J. Pan, "Optimization for Locating Emergency Medical Service Facilities: A Case Study for Health Planning from China," *Risk Manag. Healthc. Policy*, vol. 14, pp. 1791–1802, 2021, doi: 10.2147/RMHP.S304475.

[3] D. Sumrit and K. Thongsiriruengchai, "An Optimization Model for Advanced Life Support Ambulance Facility Location Problem," *Proc. 2019, Vol. 39, Page 10*, vol. 39, no. 1, p. 10, Jan. 2020, doi: 10.3390/PROCEEDINGS2019039010.

[4] J. Purnomo, Z. Fanani, T. Domai, and A. Hariswanto, "THE MODEL FOR DETERMINING LOCATION OF NAVAL BASE USING AHP METHOD AND SET COVERING PROBLEM," *Russ. J. Agric. Socio-Economic Sci.*, vol. 101, no. 5, pp. 122–131, May 2020, doi: 10.18551/rjoas.2020-05.13.

[5] I. D. Argun, "An Overview on Set Covering Problems With a Focus on Military Applications," *https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-5225-5513-1.ch003*, pp. 54–66, Jan. 1AD, doi: 10.4018/978-1-5225-5513-1.CH003.

[6] Q. Yong, D. Liu, G. Li, W. Wu, W. Sun, and S. Liu, "Reducing exposure to COVID-19 by improving access to fever clinics: an empirical research of the Shenzhen area of China," *BMC Health Serv. Res.*, vol. 21, no. 1, pp. 1–10, Dec. 2021, doi: 10.1186/S12913-021-06831-4/TABLES/1.

[7] P. S. Muttaqin, R. A. Finata, and A. A. Masturo, "Facility Location Model for Emergency Humanitarian Logistics Using Set Covering and Analytic Network Process (ANP) Method," *IPTEK J. Proc. Ser.*, vol. 0, no. 5, pp. 49–53, Nov. 2020, doi: 10.12962/J23546026.Y2020I5.7931.

[8] J. E. Beasley, "A lagrangian heuristic for set-covering problems," *Nav. Res. Logist.*, vol. 37, no. 1, pp. 151–164, Feb. 1990, doi: 10.1002/1520-6750(199002)37:1<151::AID-NAV3220370110>3.0.CO;2-2.

[9] "OR-Library: A collection of test data sets for a variety of operations research (or) problems." [Online]. Available: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html

[10] Y. Wang, D. Ouyang, L. Zhang, and M. Yin, "A novel local search for unicost set covering problem using hyperedge configuration checking and weight diversity," *Sci. China Inf. Sci.*, vol. 60, no. 6, p. 062103, Jun. 2017, doi: 10.1007/s11432-015-5377-8.

[11] F. Xu and J. Li, "A hybrid encoded memetic algorithm for set covering problem," in *Proceedings - 2018 10th International Conference on Advanced Computational Intelligence, ICACI 2018*, IEEE, Mar. 2018, pp. 552–557. doi: 10.1109/ICACI.2018.8377519.

[12] M. Gendreau and J.-Y. Potvin, "Handbook of Metaheuristics," p. 648, 2010.

[13] L. Jorquera, P. Valenzuela, M. Valenzuela, and H. Pinto, "A Binary Ant Lion Optimisation Algorithm Applied to the Set Covering Problem," 2019, pp. 156–167. doi: 10.1007/978-3-030-19810-7_16.

[14] B. Crawford, R. Soto, A. Peña, and G. Astorga, "A Binary Grasshopper Optimisation Algorithm Applied to the Set Covering Problem," 2019, pp. 1–12. doi: 10.1007/978-3-319-91192-2_1.

[15] A. Fernández, A. Peña, M. Valenzuela, and H. Pinto, “A Binary Percentile Sin-Cosine Optimisation Algorithm Applied to the Set Covering Problem,” 2019, pp. 285–295. doi: 10.1007/978-3-030-00211-4_25.

[16] M. Valenzuela, A. Pena, L. Lopez, and H. Pinto, “A binary multi-verse optimizer algorithm applied to the set covering problem,” in *2017 4th International Conference on Systems and Informatics (ICSAI)*, IEEE, Nov. 2017, pp. 513–518. doi: 10.1109/ICSAI.2017.8248346.

[17] B. Crawford *et al.*, “Binary Fruit Fly Swarm Algorithms for the Set Covering Problem,” *Comput. Mater. Contin.*, vol. 71, no. 2, pp. 4295–4318, 2022, doi: 10.32604/cmc.2022.023068.

[18] B. Crawford *et al.*, “A teaching-learning-based optimization algorithm for the weighted set-covering problem,” *Teh. Vjesn.*, vol. 27, no. 5, pp. 1678–1684, 2020, doi: 10.17559/TV-20180501230511.

[19] B. Crawford, R. Soto, G. Cabrera, A. Salas-Fernández, and F. Paredes, “Using a Social Media Inspired Optimization Algorithm to Solve the Set Covering Problem,” 2019, pp. 43–52. doi: 10.1007/978-3-030-21902-4_4.

[20] L. Pavez, F. Altimiras, and G. Villavicencio, “A K-means Bat Optimisation Algorithm Applied to the Set Covering Problem,” 2020, pp. 622–632. doi: 10.1007/978-3-030-63319-6_58.

[21] G. Villavicencio, M. Valenzuela, F. Altimiras, P. Moraga, and H. Pinto, “A K-Means Grasshopper Optimisation Algorithm Applied to the Set Covering Problem,” 2020, pp. 312–323. doi: 10.1007/978-3-030-51971-1_25.

[22] G. Villavicencio, M. Valenzuela, L. Causa, P. Moraga, and H. Pinto, “A Machine Learning Firefly Algorithm Applied to the Matrix Covering Problem,” 2021, pp. 316–325. doi: 10.1007/978-3-030-77445-5_29.

[23] J. García, G. Astorga, and V. Yepes, “An analysis of a KNN perturbation operator: An application to the binarization of continuous metaheuristics,” *Mathematics*, vol. 9, no. 3, pp. 1–20, Jan. 2021, doi: 10.3390/math9030225.

[24] L. Jorquera, P. Valenzuela, L. Causa, P. Moraga, and G. Villavicencio, “A Percentile Firefly Algorithm an Application to the Set Covering Problem,” in *Lecture Notes in Networks and Systems*, vol. 229, S. R., Ed., Springer Science and Business Media Deutschland GmbH, 2021, pp. 750–759. doi: 10.1007/978-3-030-77445-5_67.

[25] L. Jorquera, P. Valenzuela, F. Altimiras, P. Moraga, and G. Villavicencio, “A Percentil Bat Algorithm an Application to the Set Covering Problem,” 2020, pp. 223–233. doi: 10.1007/978-3-030-51971-1_18.

[26] L. Pavez, F. Altimiras, and G. Villavicencio, “A Percentil Gravitational Search Algorithm an Aplication to the Set Covering Problem,” 2020, pp. 663–673. doi: 10.1007/978-3-030-63319-6_62.

[27] J. M. Lanza-Gutierrez, B. Crawford, R. Soto, N. Berrios, J. A. Gomez-Pulido, and F. Paredes, “Analyzing the effects of binarization techniques when solving the set covering problem through swarm optimization,” *Expert Syst. Appl.*, vol. 70, pp. 67–82, 2017, doi: 10.1016/j.eswa.2016.10.054.

[28] B. Crawford *et al.*, “A self-adaptive biogeography-based algorithm to solve the set covering problem,” *RAIRO - Oper. Res.*, vol. 53, no. 3, pp. 1033–1059, Jul. 2019, doi: 10.1051/ro/2019039.

[29] B. Crawford *et al.*, “A binary monkey search algorithm variation for solving the set covering

problem," *Nat. Comput.*, vol. 19, no. 4, pp. 825–841, Dec. 2020, doi: 10.1007/s11047-019-09752-8.

[30] M. F. L. Schmitt, M. H. Mulati, A. A. Constantino, F. Hernandes, and T. A. Hild, "Ant-set: A subset-oriented ant colony optimization algorithm for the set covering problem," *J. Univers. Comput. Sci.*, vol. 26, no. 2, pp. 293–316, Feb. 2020, doi: 10.3897/jucs.2020.016.

[31] J. García, B. Crawford, R. Soto, and P. García, "A Multi Dynamic Binary Black Hole Algorithm Applied to Set Covering Problem," 2017, pp. 42–51. doi: 10.1007/978-981-10-3728-3_6.

[32] S. Sundar and A. Singh, "A hybrid heuristic for the set covering problem," *Oper. Res.*, vol. 12, no. 3, pp. 345–365, Nov. 2012, doi: 10.1007/s12351-010-0086-y.

[33] E. M. Castro, "Two Neighbourhood-based Approaches for the Set Covering Problem," *U.Porto J. Eng.*, vol. 5, no. 1, pp. 1–15, Feb. 2019, doi: 10.24840/2183-6493_005.001_0001.

[34] J. García, B. Crawford, R. Soto, and G. Astorga, "A clustering algorithm applied to the binarization of Swarm intelligence continuous metaheuristics," *Swarm Evol. Comput.*, vol. 44, pp. 646–664, Feb. 2019, doi: 10.1016/j.swevo.2018.08.006.

[35] R. Soto *et al.*, "Solving the non-unicost set covering problem by using cuckoo search and black hole optimization," *Nat. Comput.*, vol. 16, no. 2, pp. 213–229, Jun. 2017, doi: 10.1007/s11047-016-9609-7.

[36] D. Karaboga, "An idea based on Honey Bee Swarm for Numerical Optimization," *Tech. Rep. TR06, Erciyes Univ.*, no. TR06, p. 10, 2005.

[37] A. Singh, "An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem," *Appl. Soft Comput.*, vol. 9, no. 2, pp. 625–631, Mar. 2009, doi: 10.1016/j.asoc.2008.09.001.

[38] Z. G. Ren, Z. R. Feng, L. J. Ke, and Z. J. Zhang, "New ideas for applying ant colony optimization to the set covering problem," *Comput. Ind. Eng.*, vol. 58, no. 4, pp. 774–784, May 2010, doi: 10.1016/j.cie.2010.02.011.

[39] N.-E. Quemá-Taimbud, M.-E. Mendoza-Becerra, and O.-F. Bedoya-Leyva, "Initialization and Local Search Methods Applied to the Set Covering Problem: A Systematic Mapping," *Rev. Fac. Ing.*, vol. 32, no. 63, p. e15235, Feb. 2023, doi: 10.19053/01211129.v32.n63.2023.15235.

[40] V. Reyes and I. Araya, "A GRASP-based scheme for the set covering problem," *Oper. Res.*, vol. 21, no. 4, pp. 2391–2408, Dec. 2019, doi: 10.1007/s12351-019-00514-z.

[41] G. Lan and G. W. DePuy, "On the effectiveness of incorporating randomness and memory into a multi-start metaheuristic with application to the Set Covering Problem," *Comput. Ind. Eng.*, vol. 51, no. 3, pp. 362–374, Nov. 2006, doi: 10.1016/j.cie.2006.08.002.

[42] M. Pinard, L. Moalic, M. Brévilliers, J. Lepagnot, and L. Idoumghar, "A memetic approach for the unicost set covering problem," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12096 LNCS, 2020, pp. 233–248. doi: 10.1007/978-3-030-53552-0_23.

[43] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Inf. Softw. Technol.*, vol. 55, no. 12, pp. 2049–2075, Dec. 2013, doi: 10.1016/J.INFSOF.2013.07.010.

[44] Y. Wang, S. Pan, S. Al-Shihabi, J. Zhou, N. Yang, and M. Yin, "An improved configuration

checking-based algorithm for the unicost set covering problem," *Eur. J. Oper. Res.*, vol. 294, no. 2, pp. 476–491, Oct. 2021, doi: 10.1016/j.ejor.2021.02.015.

[45] B. Crawford *et al.*, "Balancing Exploration-Exploitation in the Set Covering Problem Resolution with a Self-adaptive Intelligent Water Drops Algorithm," *Adv. Sci. Technol. Eng. Syst. J.*, vol. 6, no. 1, pp. 134–145, Jan. 2020, doi: 10.25046/aj060115.

[46] R. Soto *et al.*, "Adaptive Black Hole Algorithm for Solving the Set Covering Problem," *Math. Probl. Eng.*, vol. 2018, pp. 1–23, Oct. 2018, doi: 10.1155/2018/2183214.

[47] G. Lin, J. Luo, X. Chen, H. Xu, and M. Xu, "An Adaptive Feasible and Infeasible Search Algorithm for Solving the Set Covering Problem," 2021, pp. 271–278. doi: 10.1007/978-3-030-70665-4_31.

[48] B. Crawford, R. Soto, E. Monfroy, G. Astorga, J. García, and E. Cortes, "A Meta-Optimization Approach for Covering Problems in Facility Location," in *Communications in Computer and Information Science*, vol. 742, F.-E. R. V.-R. J. L. F.-G. J. C. L.-S. E.R., Ed., Springer Verlag, 2017, pp. 565–578. doi: 10.1007/978-3-319-66963-2_50.

[49] H. Razip and M. N. Zakaria, "Combining approximation algorithm with genetic algorithm at the initial population for NP-complete problem," in *IEEE Student Conference on Research and Development: Inspiring Technology for Humanity, SCOReD 2017 - Proceedings*, IEEE, Dec. 2018, pp. 98–103. doi: 10.1109/SCORED.2017.8305413.

[50] H. Razip and N. Zakaria, "Genetic Algorithm with Approximation Algorithm Based Initial Population for the Set Covering Problem," Springer, Singapore, 2021, pp. 59–78. doi: 10.1007/978-981-16-3246-4_6.

[51] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge University Press, 2011. doi: 10.1017/CBO9780511921735.

[52] M. Brévilliers, J. Lepagnot, L. Idoumghar, M. Rebai, and J. Kritter, "Hybrid differential evolution algorithms for the optimal camera placement problem," *J. Syst. Inf. Technol.*, vol. 20, no. 4, pp. 446–467, Nov. 2018, doi: 10.1108/JSIT-09-2017-0081.

[53] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. in Bradford book. MIT Press, 1992.

[54] D. Karaboga, "Artificial Bee Colony (ABC) Algorithm - Home page." [Online]. Available: https://abc.erciyes.edu.tr/

[55] S. Balaji and N. Revathi, "A new approach for solving set covering problem using jumping particle swarm optimization method," *Nat. Comput.*, vol. 15, no. 3, pp. 503–517, Sep. 2016, doi: 10.1007/s11047-015-9509-2.