UNIVERSIDAD DE CONCEPCION
Facultad de Ingeniería
Departamento de Ingeniería
Informatica y Ciencias de la Computación

Profesor Patrocinante:
María Angélica Pinninghoff

LOCALIZACIÓN EFICIENTE EN DETECCIÓN DE BORDES EN IMÁGENES ADAPTANDO EL ALGORITMO ABC

JAIME VÁSQUEZ FEIJÓO

Informe de Memoria de Titulo
Para optar al Titulo de

Ingeniero Civil Informático

Marzo 2016

Resumen

El problema de la detección de bordes en imágenes digitales en escala de grises se puede dividir en localización e identificación, en donde la localización es la búsqueda de pixeles en una imagen y la identificación es la forma de saber si un pixel es borde o no. Un método clásico realiza una detección de bordes con una identificación eficaz, pero localización poco eficiente al analizar todos los pixeles de una imagen. La detección de bordes tiene el propósito de reducir y filtrar los datos de una imagen, entregando su estructura de bordes representativa, lo que implica que en la imagen probablemente hay gran cantidad de datos o pixeles no realmente necesarios de analizar y que son solo ruido para la detección.

El algoritmo ABC es una metaheurística del área de inteligencia de enjambre introducido el año 2005, el cual trata de simular el comportamiento natural de las abejas de miel en su recolección de comida o néctar. Las abejas de miel tienen un buen balance entre explotación y exploración, y usan mecanismos de comunicación como la danza de la abeja (*waggle dance*) para localizar de forma óptima nuevas y mejores fuentes de comida.

Luego, una identificación eficaz, como el operador de un método clásico, y una localización o búsqueda eficiente, como el algoritmo ABC, pueden ser integradas para lograr una eficiente detección de bordes. Esta integración, se logró en la creación del modelo ABC-ED básico realizado en este trabajo, en donde un pixel de una imagen detectado como borde por el modelo, puede ser considerado como una flor en la naturaleza, a la cual una abeja le puede extraer su néctar. Así, ABC-ED básico detecta con una localización eficiente las flores dentro de un ambiente, simulando a las abejas en su recolección de néctar de las flores para la colonia.

Para la creación del modelo ABC-ED básico, se establecieron definiciones necesarias para explicar mediante argumentación y pseudo-algoritmos su funcionamiento, siendo el núcleo necesario para plasmar el modelo en su prototipo implementado, del cual se describe su diseño y ambiente de trabajo usado.

Se realizó una experimentación del modelo ABC-ED básico usando su prototipo, con el fin de demostrar de forma justificada las decisiones tomadas en la construcción estructural y algorítmica, junto con la calibración de parámetros de control del modelo. Los resultados demuestran, que para obtener la estructura de bordes representativa de cada imagen con que se experimentó, se necesita no más de un 16 % de *análisis de pixeles* del total de cada imagen.

Índice

Índice de figuras

Índice de Algoritmos

Índice de tablas

Glosario

- **Heurística** := Es una técnica diseñada para resolver un problema más eficientemente cuando métodos clásicos no lo son, o para encontrar una solución aproximada cuando los métodos clásicos no encuentran ninguna solución exacta. Esto se logra por un intercambio entre optimalidad, completitud, exactitud y precisión, y tiempo de ejecución.

- **Metaheurística** := Es un método heurístico para resolver un tipo de problema computacional general, usando los parámetros dados por el usuario sobre unos procedimientos genéricos y abstractos de una manera que se espera eficiente.

- **ABC** := Es una metaheurística de la rama de algoritmos de inteligencia de enjambre. Sigla en inglés por Artificial Bee Colony (Colonia Artificial de Abejas).

- **ACO** := Es una metaheurística de la rama de algoritmos de inteligencia de enjambre. Sigla en inglés por Ant Colony Optimization (Optimización de Colonia de Hormigas).

- **GA** := Es una metaheurística de búsqueda. Sigla en inglés por Genetic Algorithm (Algoritmos Genéticos).

- **PSO** := Es una metaheurística de la rama de algoritmos de inteligencia de enjambre. Sigla en inglés por Particle Swarm Optimization (Optimización de Enjambre de Partículas).

- **DE** := Es una metaheurística de la rama de algoritmos de inteligencia de enjambre. Sigla en inglés por Differential Evolution (Evolución diferencial).

- **EA** := Es una metaheurística de la rama de algoritmos de inteligencia de enjambre. Sigla en inglés por Evolutiany Algorithms (Algoritmos Evolutivos).

- **PS-EA** := Es una metaheurística de la rama de algoritmos de inteligencia de enjambre. Sigla en inglés por Particle Swarm Inspired Evolutionary Algorithms (Algoritmos evolutivos inspirados en enjambre de partículas).

- **Cohesión** := Medida de fuerza o relación funcional existente entre las sentencias de un mismo módulo. Un módulo cohesionado ejecutará una única tarea sencilla interactuando muy poco o nada con el resto de módulos del programa. Se desea que los módulos tengan una alta cohesión.

- **Acoplamiento** := Grado de interdependencia que hay entre los distintos módulos de un programa. Se desea que esta interdependencia sea lo menor posible, es decir, un bajo acoplamiento.

- **Marco de imagen** := Pixeles extremos de una imagen que conforman su perímetro.

1. Introducción

En visión artificial y procesamiento de imágenes, la detección de bordes se preocupa de la localización e identificación de significativas variaciones de niveles de grises en una imagen digital. Esta información es muy útil en aplicaciones para imágenes en reconstrucción, movimiento, reconocimiento, mejoramiento, restauración, compresión, etc. Durante la historia del procesamiento de imágenes, una gran variedad de detectores de bordes se han ideado donde se diferencian en sus propiedades matemáticas y algorítmicas [1]. La detección de bordes debe ser eficiente y eficaz, detectando solo los verdaderos bordes de la imagen con bajo costo computacional, con el fin de capturar los requerimientos de las etapas de procesamiento subsecuentes en una aplicación. Sin embargo, esto es una tarea muy difícil de hacer. Generalmente hay un *intercambio* entre eficiencia y eficacia. Un detector de bordes que sea más eficaz que otro, requiere de un mayor costo computacional, lo que hace difícil usarlo para aplicaciones en tiempo real, por lo que se necesita *"quebrar"* de alguna forma este *intercambio* para poder usar una detección de bordes más eficaz en aplicaciones de tiempo real.

Hay una tendencia en la comunidad científica de modelar y resolver problemas de optimización complejos usando como concepto la vida natural. Esto es principalmente debido a la baja eficiencia de algoritmos clásicos de optimización para resolver a gran escala problemas altamente no lineales o combinatorios. Una de las principales características de las soluciones clásicas es la baja adaptación que tienen de poder ser modelados para obtener una solución de un problema de optimización. Con el fin de superar esta limitación, se requiere de algoritmos con mayor flexibilidad y adaptabilidad para poder modelar un problema tan cerca como a la realidad.

Basado en esta motivación, muchos algoritmos han sido desarrollados inspirándose en la naturaleza, e.g.: Algoritmos Genéticos (GA), Optimización de Colonia de Hormigas (ACO), Optimización de Enjambre de Partículas (PSO), Algoritmos Evolutivos (EA), Algoritmos Evolutivos inspirados en Enjambre de Partículas (PS-EA), Evolución Diferencial (DE), etc. Estos algoritmos son metaheurísticas que imitan o simulan la habilidad de seres o fenómenos en la naturaleza para solucionar problemas de optimización. Hay una rama conocida como inteligencia de enjambre, enfocada en el comportamiento colectivo descentralizado y auto-organizado de sistemas naturales o artificiales. Colonia de Abejas Artificiales (ABC) es un algoritmo relativamente nuevo en inteligencia de enjambre, introducido el año 2005 [2]. El algoritmo ABC trata de modelar el comportamiento natural de las abejas de miel en su recolección de comida. Las abejas de miel tienen un buen balance entre explotación y exploración [3] y usan mecanismos de comunicación como la danza de la abeja *(waggle dance)* para localizar de forma óptima nuevas y mejores fuentes de comida. Se ha comparado el algoritmo ABC con varios de los algoritmos mencionados anteriormente [4–8]. Los experimentos indican que el ABC es mejor o al menos similar que los otros algoritmos. Es flexible, adaptable y tiene la ventaja de usar menos parámetros de control. Además, el algoritmo ABC se ha adaptado exitosamente para solucionar varios problemas en el área de procesamiento de imágenes digitales [9].

Por lo anterior, el algoritmo ABC, que simula el comportamiento de recolección de comida de las abejas de miel, lo hace un buen candidato para adaptarlo y generar soluciones eficientes a problemas que requieran nuevos algoritmos inteligentes de búsqueda. En particular, para la detección de bordes en imágenes digitales.

1.1. Descripción del Problema

Los bordes son una discontinuidad en los valores de intensidad de una imagen. Estos caracterizan límites, y por ende, son un problema de fundamental importancia en procesamiento de imágenes. La detección de bordes reduce significativamente la cantidad de datos filtrando información inútil, mientras que preserva importantes propiedades estructurales de una imagen [10].

La detección de bordes se preocupa de la localización e identificación de significativas variaciones de niveles de grises en una imagen digital, siendo un paso fundamental y de los primeros en procesamiento de imágenes [11], por lo que es crucial que la detección de bordes sea eficaz y eficiente. Un método eficaz de detección de bordes, como un método clásico, tiene la desventaja de tener un alto costo computacional, siendo no eficiente al realizar cálculos para todos los pixeles de la imagen, puesto que la detección de bordes en un método clásico se realiza mediante una localización e identificación en todos los pixeles de la imagen; lo que hace difícil poder usar un método clásico para aplicaciones de tiempo real.

La detección de bordes tiene el propósito de reducir y filtrar los datos a procesar en pasos subsiguientes, entregando solo la información relevante de una imagen. Lo anterior implica, que en una imagen probablemente hay una gran cantidad de datos o pixeles que no son necesarios de analizar en la detección. **Por ende, se busca una forma de poder hacer más eficiente la detección de bordes de un método clásico, realizando una localización o búsqueda de los pixeles que son bordes, sin la necesidad de analizar todos los pixeles de la imagen.**

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar un modelo y prototipo que de solución eficiente al problema de localización en detección de bordes en imágenes digitales en escala de grises, integrando un método clásico para la identificación con el algoritmo ABC para la localización.

1.2.2. Objetivos Específicos

1. Revisar el estado del arte de métodos clásicos de detección de bordes en imágenes digitales en escala de grises.
2. Revisar el estado del arte del algoritmo ABC.
3. Desarrollar un modelo que de solución eficiente al problema de localización en detección de bordes en imágenes digitales en escala de grises, integrando un método clásico para la identificación con el algoritmo ABC para la localización.
4. Desarrollar prototipo acorde al modelo mencionado en punto 3.
5. Evaluar modelo usando su prototipo mencionado en punto 4.

1.3. Metodología

Para llevar a cabo este trabajo, se utiliza una metodología iterativa, en donde en cada iteración, si es que corresponde, se aplica:

- Investigación:
  1. Revisión y discusión bibliográfica sobre:
     a) Detección de bordes en imágenes digitales. Los métodos clásicos y principales.
     b) Algoritmo ABC, desde su inicios, qué es, cómo funciona, en qué fue inspirado y basado. Usos en la trayectoria del tiempo y trabajos relacionados al problema.
- Desarrollo del Modelo:
  1. Desarrollar un modelo que de solución eficiente al problema de localización en detección de bordes en imágenes digitales en escala de grises, integrando un método clásico para la identificación con el algoritmo ABC para la localización.
  2. Implementación del modelo desarrollado, resultando en un prototipo. Para esto:
     a) Definir herramientas necesarias para implementación y ambiente de trabajo.
     b) Implementar los métodos clásicos, generando con sus máscaras de convolución respectivas un operador de identificación de borde.
     c) Implementar modelo desarrollado mencionado en punto 1, integrando el algoritmo ABC para la localización y usando para la identificación el $fitness$ dado por un operador de método clásico mencionado en punto *b*).
- Evaluación del Modelo:
  1. Diseñar plan de prueba para el modelo:
     a) Declarar y definir métricas de evaluación para el modelo.
     b) Declarar experimentos con sus propósitos.
  2. Aplicar plan de prueba diseñado. Por cada experimento declarado:
     a) Definir experimento y llevarlo a cabo.
     b) Presentar resultados obtenidos.
     c) Realizar análisis de resultados obtenidos.
- Discusión y Conclusiones:
  1. Síntesis general del trabajo de memoria.
  2. Modelo desarrollado versus algoritmo ABC.
  3. Síntesis de modelo desarrollado.
  4. Trabajo a futuro.
- Elaboración informe de memoria: durante todo el transcurso de la memoria.

1.4. Estructura de Informe

En capitulo 2 se describe la detección de bordes y el algoritmo ABC. En capitulo 3 se presenta el modelo desarrollado que da solución al problema descrito. En capitulo 4 se muestra la experimentación hecha al modelo desarrollado. Por último, en capitulo 5 se realiza una discusión de síntesis del trabajo y se presentan las conclusiones.

2. Marco Teórico

En este capitulo se describe el estado del arte de los métodos clásicos y principales de detección de bordes, las propiedades fundamentales de la inteligencia de enjambre, el algoritmo ABC y los trabajos relacionados al problema usando ABC.

2.1. Detección de Borde en Imágenes

La detección de bordes es el primer paso en muchas aplicaciones de visión artificial. Esta reduce significativamente la cantidad de información a procesar filtrando datos no deseados o insignificantes; mientras mantiene las propiedades estructurales importantes de una imagen [11].

El principal objetivo de la detección de bordes es localizar e identificar fuertes discontinuidades de los valores de los pixeles en una imagen. Estas discontinuidades son debidas a cambios abruptos en la intensidad desde un pixel a otro, lo que caracteriza potenciales límites entre objetos o regiones en una imagen.

En el proceso de detección de bordes, se presentan problemas de falsos bordes detectados, desacierto de verdaderos bordes, localización de bordes, alto tiempo computacional, problemas debido a la presencia de ruido, etc.

Existen múltiples métodos para realizar la detección de bordes, pero la mayoría se puede agrupar en dos categorías: detección de bordes basada en la derivada de primer orden (Métodos de Gradiente) y detección de bordes basada en la derivada de segundo orden (Basados en Laplaciano). Además, está el método de detección de bordes de Canny, que es un procedimiento que usa métodos o conceptos de ambas categorías.

2.1.1. Metodos de Gradiente

Los métodos clásicos de gradiente basados en la derivada de primer orden son: Cruz de Robert [12], Sobel [13,14] y Prewitt [15]. En estos métodos, se hace convolución entre la imagen y sus respectivas máscaras o kernels para generar una imagen de gradiente en donde los bordes son detectados mediante la búsqueda del valor máximo y mínimo de intensidad en la vecindad de un pixel. Y se determina si el pixel es un borde mediante el parámetro umbral μ. Esto se hace para cada pixel de la imagen.

- **Cruz de Robert:** El operador consiste en un par de máscaras de convolución de $2x2$ (Figura 1), para detectar el gradiente en la dirección horizontal "x" y vertical "y". Este operador es más rápido que Sobel y Prewitt debido a que el tamaño de sus máscaras es menor, pero es más sensible al ruido. Una máscara es simplemente la otra rotada 90°.

$$x: \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \; y: \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$$

Figura 1: Máscaras de Robert.

- **Sobel:** Fue desarrollado para obtener una estimación del gradiente más eficaz que la Cruz de Robert. El operador consiste en un par de máscaras de convolución de $3x3$ (Figura 2), para detectar el gradiente en la dirección horizontal "x" y vertical "y". Una máscara es simplemente la otra rotada 90°. Este operador tiene una vecindad más uniforme que la vecindad de $2x2$ de Robert, lo que lo hace más isotrópico y robusto, pero con un mayor costo computacional [14].

$$x: \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \; y: \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Figura 2: Máscaras de Sobel.

- **Prewitt:** Es muy similar al operador de Sobel. Prewitt le da la misma relevancia a la vecindad diagonal, horizontal y vertical de un pixel cuando combina sus dos componentes de gradiente. En cambio, Sobel le da mayor peso a la vecindad horizontal y vertical. El operador Prewitt consiste en un par de máscaras de convolución de $3x3$ (Figura 3), para detectar el gradiente en la dirección horizontal "x" y vertical "y". Una máscara es simplemente la otra rotada 90°.

$$x: \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} \; y: \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}$$

Figura 3: Máscaras de Prewitt.

Para cada método, sus máscaras pueden ser aplicadas separadamente a la imagen de entrada para producir medidas separadas del gradiente en cada orientación, ya sea horizontal o vertical.

Sea $I(x, y)$ la intensidad de un pixel de la imagen de entrada en la posición (x, y); $G_x(x, y)$ la magnitud del gradiente en la dirección horizontal de $I(x, y)$, formado por la convolución entre la vecindad de $I(x, y)$ y la máscara "x"; $G_y(x, y)$ la magnitud del gradiente en la dirección vertical de $I(x, y)$ formado por la convolución entre la vecindad de $I(x, y)$ y la máscara "y".

Ambos componentes del gradiente $G_x(x, y)$ y $G_y(x, y)$ pueden ser combinados para calcular la magnitud del gradiente (usando expresión (1)) y orientación en cada pixel de la imagen.

$$\nabla I(x, y) = G(x, y) = \sqrt{G_x(x, y)^2 + G_y(x, y)^2} \tag{1}$$

Está la opción de calcular la magnitud del gradiente de una forma más rápida computacionalmente, pero menos precisa con la expresión (2).

$$|G(x, y)| = |G_x(x, y)| + |G_y(x, y)| \tag{2}$$

El ángulo de orientación o dirección del gradiente para Sobel y Prewitt está dado por la expresión (3). Para la Cruz de Robert, la expresión (3) tiene una modificación presentada por la expresión (4).

$$\Theta(x,y) = \arctan(\frac{G_y(x,y)}{G_x(x,y)}) \tag{3}$$

$$\Theta(x,y) = \arctan(\frac{G_y(x,y)}{G_x(x,y)}) - \frac{3\pi}{4} \tag{4}$$

El operador convolución "$*$" define a $G_x(x,y)$ y $G_y(x,y)$ para cada método, en base a sus máscaras respectivas y la vecindad de un pixel de la imagen de entrada definida en la Figura 4, con p_i el valor de intensidad del pixel i. Para cada método, los componentes del gradiente se definen como sigue:

Cruz de Robert:

$$G_x(x,y) = x * v_1 = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} * \begin{bmatrix} p_1 & p_2 \\ p_3 & p_4 \end{bmatrix} = p_1 - p4$$

$$G_y(x,y) = y * v_1 = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} * \begin{bmatrix} p_1 & p_2 \\ p_3 & p_4 \end{bmatrix} = p_2 - p3$$

Sobel y Prewitt: con $a = 2$ para Sobel y $a = 1$ para Prewitt.

$$G_x(x,y) = x * v_2 = \begin{bmatrix} -1 & 0 & +1 \\ -a & 0 & +a \\ -1 & 0 & +1 \end{bmatrix} * \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{bmatrix} = p_3 + a \times p_6 + p_9 - p_1 - a \times p_4 - p_7$$

$$G_y(x,y) = y * v_2 = \begin{bmatrix} -1 & -a & -1 \\ 0 & 0 & 0 \\ +1 & +a & +1 \end{bmatrix} * \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{bmatrix} = p_7 + a \times p_8 + p_9 - p_1 - a \times p_2 - p_3$$

$$v_1 : \begin{bmatrix} p_1 & p_2 \\ p_3 & p_4 \end{bmatrix} \; v_2 : \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{bmatrix}$$

Figura 4: Vecindades para un pixel en una imagen. v_1 es la vecindad para el operador de Cruz de Robert centrado en p_1. v_2 es la vecindad para el operador de Sobel y Prewitt centrado en p_5.

2.1.2. Metodos basados en Laplaciano

El Laplaciano es una medida bidimensional isotrópica basado en la derivada de segundo orden de una imagen. Este operador se puede aproximar en forma discreta por el kernel en la Figura 5. El Laplaciano para $I(x,y)$ con vecindad v_2 de la Figura 4 se define por la convolución dada en la expresión (5).

$$Laplaciano[I(x,y)] = 4 \times p_5 - p_2 - p_4 - p_6 - p_8 \tag{5}$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Figura 5: Máscara de Laplaciano.

El Laplaciano de Gauss (LoG) [16] combina un filtro de suavizado gaussiano con el Laplaciano. LoG se puede aproximar en forma discreta por la máscara de la Figura 6. Luego, LoG para $I(x, y)$ con vecindad v_2 de la Figura 4 se define por la convolución presentada en la expresión (6).

$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Figura 6: Máscara de Laplaciano de Gauss (LoG).

$$LoG[I(x, y)] = 4 \times p_5 + (p_1 + p_3 + p_7 + p_9) - 2 \times (p_2 + p_4 + p_6 + p_8) \tag{6}$$

2.1.3. Detector de Bordes de Canny

El detector de bordes de Canny [17] fue desarrollado con la intención de mejorar los detectores de bordes ya desarrollados en ese tiempo. Es considerado el mejor detector de bordes. El procedimiento de Canny tiene el propósito de satisfacer tres objetivos principales:

1. **Baja tasa de error:** buena detección de solo bordes realmente existentes.
2. **Buena localización:** la distancia entre los pixeles de borde detectados y pixeles de borde reales tiene que ser mínima.
3. **Respuesta mínima:** solo una respuesta de detección por cada borde.

El procedimiento de Canny sigue una serie de pasos:

1. Quita el ruido de la imagen usando un filtro de suavizado gaussiano.
2. Calcula el gradiente de cada pixel en la imagen usando un método de gradiente descrito en sección 2.1.1. Generalmente es usado el operador Sobel.
3. Calcula la dirección del gradiente de cada pixel y es redondeada, dependiendo del valor, a uno de los cuatro posibles ángulos: 0°, 45°, 90° o 135°.
4. Se aplica la técnica no-máxima supresión (non-maximum suppression) para el adelgazamiento de bordes.
5. Se aplica histéresis, usando dos umbrales dados por parámetro: superior (μ_{max}) e inferior (μ_{min}), de la siguiente forma:
   - Si $G(x, y) > \mu_{max} : I(x, y)$ se acepta como borde.
   - Si $G(x, y) < \mu_{min} : I(x, y)$ se rechaza como borde.
   - Si $\mu_{min} \leq G(x, y) \leq \mu_{max} : I(x, y)$ se acepta como borde solo si está conectado a un pixel que tiene una magnitud de gradiente mayor que μ_{max}.

2.2. Inteligencia de Enjambre

En años recientes, la inteligencia de enjambre ha atraído a muchos investigadores científicos de áreas relacionadas. Boneabeu *et al.* definieron inteligencia de enjambre como "...cualquier intento de diseñar algoritmos o dispositivos para solucionar problemas distribuidos inspirados por el comportamiento colectivo de colonias de insectos sociales y otras sociedades animales..." [18].

Dos conceptos fundamentales: autoorganización y división del trabajo son propiedades necesarias y suficientes para obtener un comportamiento inteligente de enjambre.

1. La **autoorganización** es un conjunto de mecanismos dinámicos, que dan lugar a estructuras al nivel global del sistema por medio de interacciones entre sus componentes de menor nivel. Estos mecanismos establecen reglas básicas para las interacciones entre los componentes del sistema. Las reglas aseguran que estas interacciones sean ejecutadas sobre la base de sólo información local sin ninguna relación con el diseño global. La autoorganización se basa en cuatro propiedades básicas: retroalimentación positiva, retroalimentación negativa, fluctuaciones e interacciones múltiples.

   a) **Retroalimentación positiva:** es un comportamiento simple de "regla de oro"que promueve la creación de estructuras convenientes, *e.g:* colocación de rastro de feromona en hormigas y bailes en las abejas.

   b) **Retroalimentación negativa:** contrapesa la retroalimentación positiva y ayuda a estabilizar el diseño colectivo. Ayuda a salir de ciertas soluciones locales encontradas.

   c) **Fluctuaciones:** la aleatoriedad es a menudo crucial para estructuras emergentes, ya que permite el descubrimiento de nuevas soluciones.

   d) **Interacciones múltiples:** en general, se requiere de una densidad mínima de individuos mutuamente tolerantes, permitiéndoles hacer uso de los resultados de sus propias actividades así como de otros.

2. La **división del trabajo** es un fenómeno creado por la acción de diferentes tareas dentro de un sistema, que son realizadas simultáneamente mediante la cooperación de individuos especializados. Esto se cree que es más eficiente que tareas secuenciales realizadas por individuos no especializados [19].

2.3. Algoritmo ABC

Se describe a continuación el comportamiento de abejas de miel, el algoritmo ABC y sus usos en la historia. Luego, los trabajos relacionados al problema usando el algoritmo ABC.

2.3.1. Comportamiento de Abejas de Miel Reales

Tereshko considera una colonia de abejas de miel como un sistema dinámico recolectando información desde un ambiente y ajustando su comportamiento acorde a éste [19].

Tereshko *et al.* desarrollaron un modelo mínimo del comportamiento de una colonia de abejas de miel en su recolección de comida, que lleva al surgimiento de inteligencia colectiva de

estas [19–21]. Consiste de tres componentes esenciales: fuentes de comida, obreras empleadas y obreras desempleadas. Y define dos modos destacados de comportamiento: reclutamiento para una fuente y el abandono de ésta. Los componentes esenciales son explicados como sigue:

- **Fuente de comida:** el valor de la fuente de comida para un insecto depende de varios factores, entre los que se incluye su cercanía a la colmena, riqueza o concentración de energía, y la facilidad de extraer esta energía. Por simplicidad, se describe el "*beneficio*" de una fuente como un solo atributo. Se ha demostrado que las abejas prefieren una fuente con mayor riqueza que otra sin importar que tenga una mayor distancia con respecto de la colmena [22], por lo que siempre encontrarían la mejor fuente de comida dentro de un ambiente cambiante.

- **Obrera empleada:** está asociada con una fuente de comida en particular que está actualmente explotando o que está "empleada" en ésta. La empleada lleva consigo la información sobre esta fuente, tales como la distancia, dirección respecto de la colmena y el *beneficio*. La empleada comunica en el área de baile de la colmena la información de su fuente a través de la danza *(waggle dance)* con cierta probabilidad, dependiendo del beneficio de su fuente en la cual está empleada. Observar que la empleada solo sabe de la fuente que está actualmente explotando, por lo que solo posee información de forma local. Una vez que la fuente se haya "empobrecido", la empleada pasaría a ser una obrera desempleada.

- **Obrera desempleada:** está continuamente buscando fuentes de comida para explotar. Hay dos tipos de obrera desempleada: la exploradora y la espectadora. La exploradora busca aleatoriamente en el ambiente alrededor de la colmena (hasta un radio de 14 km aprox.) nuevas fuentes de comida. La espectadora espera en el área de baile de la colmena y es reclutada para una fuente a través de la información comunicada por obreras empleadas. El porcentaje de obreras que son exploradoras varía desde un 5 % hasta un 30 % (depende de la cantidad de información hacia la colmena), con respecto al número total de abejas de la colmena. El promedio es de un 10 % [22].

El intercambio de información entre abejas es el acontecimiento más importante en la formación de un conocimiento colectivo. Cada obrera empleada comparte su información de la fuente en que está empleada con una probabilidad que es proporcional al *beneficio* de su fuente, danzando en el área de baile de la colmena donde se encuentran las espectadoras con el fin de reclutarlas a la fuente que se está comunicando. Luego, la cantidad de información circulando a través de la colmena sobre una fuente será proporcional al *beneficio* de ésta, debido a que una fuente con mayor *beneficio* tendrá más espectadoras que decidieron emplearse en esta fuente, y por ende, tendrá más obreras comunicando mediante danza esta fuente. Por lo tanto, el reclutamiento para una fuente es proporcional al *beneficio* de esta [21].

Por simplicidad, el abandono de una fuente de comida es igualmente probable para todas las fuentes de comida [21].

Las propiedades básicas (mencionadas de forma general en sección 2.2) en que se basa la autoorganización de las abejas de miel son las siguientes:

1. **Retroalimentación positiva:** si el *beneficio* en fuentes de comida aumenta, el número de espectadoras visitando estas fuentes también aumentará.

2. **Retroalimentación negativa:** la explotación de fuentes de comida con pobre *beneficio* es detenido por las propias obreras.

3. **Fluctuaciones:** las exploradoras realizan procesos de búsqueda aleatorios para descubrir nuevas fuentes de comida.

4. **Interacciones múltiples:** las obreras comparten su información sobre las fuentes de comida con sus compañeras de colmena en el área de danza.

2.3.2. Algoritmo de Colonia de Abejas Artificiales (ABC)

El algoritmo ABC está basado en el comportamiento inteligente de enjambre de abejas de miel en su recolección de comida. Fue introducido por Dervis Karaboga, en base a los componentes esenciales de Tereshko *et al.*, con el propósito de resolver problemas de optimización multidimensionales y multimodales [2].

En el algoritmo ABC, se simula que la mitad de la colonia de abejas está compuesta por obreras empleadas y la otra mitad por obreras espectadoras. Una obrera empleada es la que explota o se emplea en una fuente de comida. Una obrera espectadora se emplea estocásticamente en una fuente. Por cada fuente, solo hay una obrera empleada asociada al mismo tiempo. Una fuente representa una posible solución del problema a optimizar. En el algoritmo ABC, solo se manejan fuentes de comida para simular el comportamiento de las obreras. Los pasos del algoritmo ABC están descritos en Algoritmo 1.

Algoritmo 1 Pseudo-algoritmo ABC. Los pasos.

Input: Establecer los parámetros: SN, MCN y $limit$.
Output: Fuente de comida con mayor $fitness$.

```
procedure ABC
    Inicialización;
    ciclo ← 0;
    while ciclo < MCN do
        Fase Obreras Empleadas;
        Calcular Probabilidades;
        Fase Obreras Espectadoras;
        Fase Obrera Exploradora;
        Memorizar la fuente de comida con mayor fitness;
        ciclo ← ciclo + 1;
    end while
end procedure
```

Hay tres parámetros de control en el algoritmo ABC básico. SN es el número de fuentes de comida, el cual es igual al número de obreras empleadas y al número de obreras espectadoras. MCN es el número máximo de ciclos, que es la condición de parada del ABC básico. $limit$ es el límite de *pruebas* para abandonar una fuente en caso de que ésta no haya podido ser mejorada. Observar que por cada ciclo del algoritmo ABC, se consideran las SN fuentes por cada paso.

En la **Inicialización**, los valores de los parámetros son establecidos y una población de SN fuentes es creada aleatoriamente. A cada fuente $x(s)$, con $f[x(s)]$ se evalúa en función del problema a optimizar y se calcula su $beneficio/fitness$, el cual se asigna a $fit[x(s)]$, y su contador de *pruebas* se inicializa a cero (Algoritmo 2).

Algoritmo 2 Pseudo-algoritmo ABC detallado. Parte 1: Inicialización.

Input: Establecer los parámetros de control.
SN : número de fuentes de comida.
MCN : máximo número de ciclos.
$limit$: máximo número de pruebas para abandonar una fuente.
Output: Fuente de comida con mejor posición.
procedure ABC
    // Inicialización
    for $s = 1$ to SN **do**
        $x(s) \leftarrow$ solución aleatoria;
        $fit[x(s)] \leftarrow f[x(s)]$;
        $pruebas[x(s)] \leftarrow 0$;
    end for

En la **Fase de Obreras Empleadas**, para cada fuente, se genera una fuente candidata $v(s)$ en el vecindario de la fuente actual $x(s)$ mediante expresión (7) (Algoritmo 3).

$$v(s)_j = x(s)_j + \phi[x(s)_j - x(k)_j] \tag{7}$$

Donde $k \in \{1, 2, ..., SN\}$, con $k \neq s$, y $j \in \{1, 2, ..., D\}$ son índices escogidos aleatoriamente. D es la dimensión del problema (e.g.: bidimensional, $D = 2$). $\phi \in \mathbb{R}$, es un número aleatorio entre $[-1, 1]$ que controla la producción de una fuente en el vecindario de $x(s)$. Cada fuente $x(s)$ tiene un conjunto de D elementos, en donde cada elemento $x(s)_j$ es la posición de $x(s)$ en la dimensión j respectiva. El conjunto de D elementos de $v(s)$ es igual al de $x(s)$, con la excepción de la posición calculada $v(s)_j$ para la dimensión j respectiva.

Algoritmo 3 Pseudo-algoritmo ABC detallado. Parte 2: Fase Obreras Empleadas.

    $ciclo \leftarrow 0$;
    while $ciclo < MCN$ **do**
        // Fase Obreras Empleadas
        for $s = 1$ to SN **do**
            $v(s) \leftarrow$ solución vecina de $x(s)$ calculada por exp. 7;
            $fit[v(s)] \leftarrow f[v(s)]$;
            if $fit[v(s)] > fit[x(s)]$ **then**
                $x(s) \leftarrow v(s)$;
                $fit[x(s)] \leftarrow fit[v(s)]$;
                $pruebas[x(s)] \leftarrow 0$;
            else
                $pruebas[x(s)] \leftarrow pruebas[x(s)] + 1$;
            end if
        end for

Luego, se calcula el $fitness$ de $v(s)$ y se realiza una selección *greedy* entre la fuente actual $x(s)$ y la fuente candidata $v(s)$ detallada a continuación:

- Si $fit[v(s)] > fit[x(s)]$: se reemplaza la fuente actual $x(s)$ por la nueva fuente $v(s)$.
- Si $fit[v(s)] \leq fit[x(s)]$: se mantiene la fuente actual $x(s)$, pero se incrementa en uno su contador de *pruebas*.

En **Calcular Probabilidades**, se calcula la probabilidad $p[x(s)]$ de cada fuente $x(s)$ en base a su $fitness$ mediante la expresión (8), con el fin de que cada obrera espectadora pueda elegir una fuente mediante su $fitness$ (Algoritmo 4).

$$p[x(s)] = \frac{fit[x(s)]}{\sum_{s=1}^{SN} fit[x(s)]} \tag{8}$$

De esta forma, se simula el intercambio de información entre la obrera empleada y espectadora, al codificar la información de cada fuente de comida asociada a cada obrera empleada, para ser transmitida mediante danza hacia las espectadoras.

En la **Fase de Obreras Espectadoras**, se elige estocásticamente SN fuentes de comida, dependiendo del valor de la probabilidad $p[x(s)]$. De tal forma, que es más probable que se elija una fuente con mayor $fitness$. Por cada fuente elegida se actúa igual que en la Fase de Obreras Empleadas. Se genera una fuente vecina candidata de la fuente actual para luego hacer selección *greedy* entre ambas (Algoritmo 4).

Algoritmo 4 Pseudo-algoritmo ABC detallado.
Parte 3: Calcular Probabilidades y Fase Obreras Espectadoras.

```
// Calcular Probabilidades
Se calcula la probabilidad p[x(s)] de cada fuente por exp. 8;
// Fase Obreras Espectadoras
s ← 1; t ← 0;
while t < SN do
    r ← rand(0,1); // r ∈ ℝ, un número aleatorio.
    if r < p[x(s)] then
        t ← t + 1;
        v(s) ← solución vecina de x(s) calculada por exp. 7;
        fit[v(s)] ← f[v(s)];
        if fit[v(s)] > fit[x(s)] then
            x(s) ← v(s);
            fit[x(s)] ← fit[v(s)];
            pruebas[x(s)] ← 0;
        else pruebas[x(s)] ← pruebas[x(s)] + 1;
        end if
    end if
    s ← (s mod SN) + 1
end while
```

Observar que para elegir SN fuentes, se hacen t intentos (con $t \geq SN$), en donde es más probable que se elijan estocásticamente las SN fuentes con mayores $fitness$, incluyendo las fuentes candidatas que hicieron reemplazo por selección greedy en el mismo proceso de la fase. Esto hace que el algoritmo ABC pueda converger más eficientemente hacia la solución final al simular el reclutamiento para una fuente.

En la **Fase de Obrera Exploradora**, por cada fuente, se revisa su contador de *pruebas* y se compara con el valor del parámetro *limit* (Algoritmo 5).

- Si $pruebas[x(s)] > limit$: se abandona la fuente de comida $x(s)$, reemplazándola por una solución aleatoria, a la cual se calcula su $fitness$.
- Si $pruebas[x(s)] \leq limit$: la fuente $x(s)$ se mantiene para el siguiente ciclo del algoritmo.

Algoritmo 5 Pseudo-algoritmo ABC detallado. Parte 4: Fase Obreras Exploradoras.

```
44:         // Fase Obrera Exploradora
45:         for s = 1 to SN do
46:             if pruebas[x(s)] > limit then
47:                 x(s) ← solución aleatoria;
48:                 fit[x(s)] ← f[x(s)];
49:                 pruebas[x(s)] ← 0;
50:             end if
51:         end for
52:         Memorizar la fuente de comida con mejor posición;
53:         ciclo ← ciclo + 1;
54:     end while
55: end procedure
```

Durante el transcurso del tiempo, se han hecho algunas modificaciones en el algoritmo ABC básico en las condiciones de parada del algoritmo (Algoritmo 1, punto 4). Sobre todo para poder compararlo con otros algoritmos de forma justa e imparcial, se ha introducido un nuevo parámetro de control llamado MFE que es el número máximo de evaluaciones de $fitness$ que se realizan. Se usa un contador global de *número de evaluaciones* de $fitness$ que se incrementa cada vez que se realiza una evaluación, y este es comparado con MFE en la condición de parada del algoritmo, en vez de comparar metaheurísticas por la cantidad de ciclos ejecutados [3], debido a que cada uno de estos algoritmos realiza procesos internos diferentes en cada ciclo, pero la cantidad de cálculos de $fitness$ indica cuantas evaluaciones y análisis de soluciones tuvo que hacer una metaheurística en sus procesos internos para converger a la solución final encontrada.

2.3.3. Historia del Algoritmo ABC

Karaboga *et al.* describieron formalmente el algoritmo ABC para resolver problemas de optimización de funciones numéricas. El ABC fue comparado usando un limitado número de problemas de testeo, con otros algoritmos: GA [4], PSO y PS-EA [5] y DE, PSO y EA [6]. En donde ABC superó a los otros algoritmos. Se concluye que el algoritmo ABC tiene la habilidad de salir de soluciones locales y puede ser usado eficientemente para funciones de optimización

multivariables y multimodales.

Karaboga *et al.* extendió el algoritmo ABC para resolver problemas de optimización con restricciones [7]. Con el fin de hacer frente a las restricciones, se reemplazó el mecanismo de selección de fuente de comida original del ABC. Se experimentó con trece problemas de restricción bien conocidos y se comparó con PSO y DE. Se concluyó que el algoritmo ABC puede ser eficientemente usado para éste tipo de problema.

Karaboga *et al.* comparó el desempeño del algoritmo ABC con los algoritmos GA, DE, PSO y ES en un gran conjunto de funciones multidimensionales y multimodales de testing sin restricciones para su optimización [8]. Los experimentos indican que el algoritmo ABC es mejor o similar que los algoritmos comparados, y tiene la ventaja de que usa menos parámetros de control que los otros algoritmos.

El algoritmo ABC se ha aplicado para el problema del vendedor viajero [23], entrenar redes neuronales [24, 25], para el problema de árbol de cobertura mínima limitado de hojas [26] y muchos otros problemas [27].

2.3.4. Trabajos detectando bordes en imágenes usando ABC

Desde el 2009, el algoritmo ABC ha sido exitosamente usado para varios problemas en las áreas de procesamiento de señales, imágenes y video. Akay y Karaboga realizaron un estudio de los problemas en estas áreas en que el algoritmo ABC ha sido aplicado [9]. Los métodos tradicionales del estado del arte dedicados a los problemas en estas áreas tienen desventajas, como un alto costo computacional, tener que afinar bien parámetros, baja velocidad de procesamiento, etc. Esto ha llevado a que los investigadores usen herramientas alternativas de optimización, como el algoritmo ABC para resolver eficientemente problemas de optimización difíciles en estas áreas. En [9] se estudió 133 publicaciones entre 2009 y 2014, en donde se ve que la cantidad de publicaciones crece cada año. De los 133 trabajos, 110 han sido sobre procesamiento de imágenes en una gran cantidad de sus subáreas. En detección de bordes, se han publicado 4 trabajos [28–31].

Para una mejora en la calidad de los bordes de imágenes, en [28] se usó filtros híbridos de suavizado de imagen y el algoritmo ABC. Los resultados se compararon con Algoritmos genéticos (GA). Se demuestra que ABC es la técnica de optimización más potente para muestras con un espacio de solución grande, debido a su búsqueda en la muestra de forma estocástica e imparcial; por lo que es rápidamente adaptable a procesamiento de imagen, por ende, a detección y mejoramiento de bordes. Además, para poder evaluar localmente la solución, mediante $fitness$, se midió por la suma de los valores de intensidades de bordes en una imagen mejorada, debido a que una imagen en escala de grises con un buen contraste visual incluye varios bordes intensivos. La intensidad de un borde es calculada usando el operador Prewitt.

En [29] se presenta un mecanismo para diseñar plantillas de sensores de imágenes basados en redes celulares neurales (CNNs, sigla en inglés) usando el algoritmo ABC para la detección de bordes. Primero se realizó una evaluación estadística para inspeccionar la conveniencia de usar ABC para el problema, resultando que ABC es un algoritmo estable y robusto, por ende,

es apropiado utilizarlo. En el mecanismo, la imagen de salida dada por CNN converge a una imagen con bordes ideales ajustando las plantillas de CNN por medio del algoritmo ABC. Lo anterior es realizado comparando ambas imágenes con una función objetivo de similitud que entrega el valor del $fitness$. El algoritmo ABC evalúa mediante el $fitness$ con el fin de obtener un mejor resultado, produciendo un conjunto de plantillas que son enviadas a CNN, el cual usa este conjunto y la imagen con bordes ideales para generar la imagen de salida final. Los resultados del método son comparados con resultados de métodos bien conocidos: Canny, Robert, Sobel y Prewitt. Además de otros métodos particulares enfocados a solucionar el problema con CNNs sin el algoritmo ABC. La comparación entre los resultados dados en imágenes en escala de grises y binarizadas son evaluados mediante las métricas C (correlación), MSE (error cuadrático medio) y SSIM (similitud estructural), dando como resultado que el método CNN basado en ABC se desempeña mejor que todos los otros métodos.

En [30] se presenta un modelo híbrido entre el algoritmo de atención visual basada en la prominencia (saliency-based visual attention) y el algoritmo ABC para la detección de bordes para un vehículo aéreo de combate no tripulado (UCAV) con el fin de que pueda reconocer objetivos. Primero, con la atención visual basada en la prominencia, se hace un preprocesamiento a la imagen de entrada. Se hace extracción de características (color, intensidad y orientación) en las cuales se realizan operaciones centro-envolventes y normalización para luego hacer combinación a través de diferentes escalas con el fin de obtener regiones prominentes de la imagen y poder reducir en gran cantidad la información irrelevante a procesar de la imagen original. Luego, en estas regiones prominentes, el algoritmo ABC es usado para detectar los bordes, evitando los efectos del ambiente y ruido. Para obtener el $fitness$, se usa el valor de la intensidad de gris, donde para un pixel o fuente de comida actual, se suman las diferencias absolutas de vecinos opuestos en la vecindad de la fuente, donde la vecindad son los 8 pixeles alrededor del pixel central, incluyendo el vecino horizontal y vertical de cada pixel diagonal de la vecindad de la fuente actual, lo que forma una vecindad clique. Los resultados obtenidos demuestran la factibilidad y eficacia del método propuesto, garantizando una eficiente localización de un objetivo con una precisa detección de bordes en ambientes complejos con ruido. Los resultados son presentados solo pictográficamente.

En [31], se presenta un modelo que usa para un pixel o fuente actual una vecindad de Moore, la cual se usa para elegir aleatoriamente una fuente vecina. El valor de intensidad de gris de una fuente es usado como $fitness$. La cantidad de fuentes creadas en inicialización, depende del tamaño de la imagen, siendo igual a la raíz cuadrada de la cantidad de pixeles de la imagen. Se usa un umbral para clasificar una fuente como borde. Este umbral es definido como la desviación estándar de los niveles de gris en la imagen. Los resultados obtenidos se compararon con los resultados de Canny, Robert y Sobel. Para hacer esta comparación, se usó la métrica de la Distancia de Hamming (DH), que hace la comparación entre dos imágenes binarizadas de igual tamaño. DH analiza cada pixel de igual posición entre las imágenes, entregando la cantidad total de pixeles diferentes entre ambas imágenes de salidas. Se concluye que el algoritmo ABC puede ser una alternativa para detección de bordes en imágenes. Sin embargo, los resultados no presentan un nivel cercano de eficacia a los métodos clásicos y Canny.

3. Desarrollo

Se describe a continuación el modelo ABC-ED básico, que trata el problema de localización eficiente en detección de bordes en imágenes digitales en escala de grises, integrando un método clásico para la identificación, con el algoritmo ABC para la localización. Primero se establecen definiciones necesarias del modelo. Luego, se explica el modelo y se muestra como pseudo-algoritmo. Por último, se profundiza como fue implementando para la creación de su prototipo.

3.1. Definiciones

La entrada del problema es una imagen digital en escala de grises I. La imagen I está compuesta de pixeles con un tamaño de 8 bit cada uno, donde el valor de cada pixel es de una variedad de un conjunto de $2^8 = 256$ posibles valores dentro del rango $\{0, 1, 2, ..., 255\}$, donde el 0 es el negro, el 255 es blanco, y los valores entre $]0, 255[$ representan un nivel de gris.

La imagen I es representada por una matriz bidimensional IM de r filas y c columnas, donde:

- Cada elemento o pixel (por simplicidad) de IM es el valor de intensidad de gris de cada pixel de I en la posición respectiva.

Un pixel de IM es una **posible** fuente de comida. Cada fuente tiene una vecindad, en donde la cantidad de vecinos depende de la posición de la fuente en IM. Sea,

SN := El número de fuentes de comida.
f_k := Una fuente de comida k.
$f_k(i, j)$:= Una fuente de comida k con posición (i, j) en IM e I.
N_{f_k} := La vecindad de f_k.
Λ_{f_k} := La cantidad de vecinos de f_k.
n_{f_k} := Un vecino de f_k. Con $n_{f_k} \in N_{f_k}$.
$n^{\nu}_{f_k}$:= El vecino ν de f_k. Con $n^{\nu}_{f_k} \in N_{f_k}$.

Donde $k \in \{1, 2, ..., SN\}$ y $\nu \in \{1, 2, ..., \Lambda_{f_k}\}$. Solo una obrera empleada está asociada a una fuente de comida. Con $i, j \in \mathbb{N}$, $i \in \{1, 2, ..., r\}$ y $j \in \{1, 2, ..., c\}$ representa la posición de la fila y columna respectivamente en IM.

La vecindad de una fuente de comida $f_k(i, j)$ se define como la vecindad de Moore de 8 vecinos alrededor (horizontal, vertical y diagonal) de f_k. Se ilustra en la Figura 7 la vecindad de f_k con la posición de cada vecino $n^{\nu}_{f_k}$. El tipo de posición de f_k en IM es: No marco de I.

$$\begin{bmatrix} n^1_{f_k}(i-1, j-1) & n^2_{f_k}(i-1, j) & n^3_{f_k}(i-1, j+1) \\ n^4_{f_k}(i, j-1) & f_k(i, j) & n^5_{f_k}(i, j+1) \\ n^6_{f_k}(i+1, j-1) & n^7_{f_k}(i+1, j) & n^8_{f_k}(i+1, j+1) \end{bmatrix}$$

Figura 7: Vecindad de Moore de $f_k(i, j)$. Con tipo de posición de f_k en IM: No marco de I.

El cálculo de $fitness$ de una fuente es dado por un operador de detección de bordes de un método clásico, donde el $fitness$ de una fuente es calculado por la expresión (1). Así, se aprovecha la igual dimensión de la vecindad definida en el modelo y los operadores Sobel y

Prewitt. Para Roberts, en su vecindad de 2x2 solo se debe considerar una nueva fila superior y nueva columna a la izquierda de la vecindad de ceros.

Se definen cuatro conjuntos, que representan los estados posibles de una fuente. Estos conjuntos son exclusivos, i.e., una fuente solo pertenece al mismo tiempo a un solo conjunto.

- NFS := Son las fuentes que recién se han creado y no pertenecen a otro conjunto aún.
- AFS := Son las fuentes activas. Cada fuente de AFS está asociada a una obrera.
- IFS := Son las fuentes inactivas. Estas fuentes fueron activas anteriormente, pero cada una fue reemplazada por una fuente vecina al hacer selección greedy entre ambas. Estas fuentes aún no han sido agotadas, pero sí abandonadas.
- EFS := Son las fuentes agotadas. Debido a que la cantidad de *pruebas* de cada fuente es igual o superior al limite posible, o que todos sus vecinos han sido analizados.

Además, se define el conjunto RP que contiene las posiciones rechazadas de IM que se van encontrando en el proceso donde su $fitness \leq \mu$, por lo que una fuente no pertenece a RP.

Una fuente de comida "selecta" es una fuente cuyo $fitness > \mu$. En ABC-ED básico se crean solo fuentes de comida selectas. Para justificación, ver experimentación en sección 4, en particular experimento 2 parte 1 en sección 4.2.2.

Se define $SN = \sqrt{r \times c}$, debido a que se desea como cota superior $\{r \times c\}$, al realizar SN acciones SN veces, y también porque es usado en el estado del arte [31].

Ya teniendo lo necesario definido, se describe el modelo ABC-ED básico, o por simplicidad de expresión, ABC-ED (Artificial Bee Colony - Edge Detector o Colonia de Abejas Artificiales Detectoras de Borde).

3.2. Modelo ABC-ED básico

Los parámetros de entrada para ABC-ED básico son los del algoritmo ABC: SN, $limit$ y MCN. A los que se suman cuatro parámetros: I, m, μ y ε. I es la imagen de entrada, m es el que define con cual operador de método clásico se calculara el $fitness$ de una fuente, μ representa el valor de umbral para clasificar un pixel como borde o fuente mediante su $fitness$ y ε controla la localización por exploración entre nuevas fuentes y usar fuentes de IFS.

La salida de ABC-ED básico es una imagen binarizada con todas las posiciones de fuentes creadas las cuales su $fitness$ es mayor que el parámetro umbral μ.

Los pasos de ABC-ED básico están descritos en Algoritmo 6. Se lee la imagen de entrada y se establecen los parámetros. A continuación se explica cada paso.

En la **Inicialización**, se crea una población aleatoria de SN fuentes de comida (Algoritmo 7). Para obtener una fuente, se usa la función obtener-nueva-unica-fuentecomida-selecta (Algoritmo 14), en donde para cada posición generada, si esta no existe ya como fuente y $\notin RP$, se calcula su $fitness$ (Algoritmo 15), para el cual si es mayor que μ, se crea la fuente

Algoritmo 6 Pseudo-algoritmo ABC-ED básico. Los pasos.

Input: Imagen y establecer los parámetros.
1: IM : imagen de entrada I.
2: m : operador de método clásico.
3: μ : valor para umbralización.
4: SN : número de fuentes de comida.
5: MCN : máximo número de ciclos.
6: $limit$: máximo número de pruebas para agotamiento de una fuente.
7: ε : valor de control para localización por exploración.
Output: Imagen binarizada con cada posición de fuente de comida aceptada como borde.
8: **procedure** ABC-ED-BASICO( )
9: INICIALIZACIÓN( );
10: $ciclo \leftarrow 0$;
11: **while** $ciclo < MCN \wedge SN > 0$ **do**
12: FASE-OBRERAS-EMPLEADAS( );
13: CALCULAR-PROBABILIDADES( );
14: FASE-OBRERAS-ESPECTADORAS( );
15: FASE-OBRERA-EXPLORADORA( );
16: $ciclo \leftarrow ciclo + 1$;
17: **end while**
18: GENERAR-IMAGEN-BINARIZADA( );
19: **end procedure**

con la posición y se retorna, en caso contrario, la posición se agrega a RP. Luego, la fuente obtenida, se agrega en AFS. En el caso de que no hayan más posibles fuentes para crear (Algoritmo 13), se debe terminar la ejecución del algoritmo, ya que no existen más posiciones en IM identificadas como bordes. Para terminar el algoritmo se puede hacer *e.g.*: $ciclo \leftarrow MCN$ y/o $SN \leftarrow 0$, y luego terminar el ciclo de la Inicialización.

Algoritmo 7 Pseudo-algoritmo ABC-ED básico. Función: Inicialización.

1: **function** INICIALIZACIÓN( )
2: FuenteComida f_k;
3: **for** $f \leftarrow 1$ to SN **do**
4: **if** HAY-MAS-POSIBLES-FUENTES-PARA-CREAR( ) **then**
5: $f_k \leftarrow$ OBTENER-NUEVA-UNICA-FUENTECOMIDA-SELECTA( );
6: AGREGAR-EN-AFS(f_k); // ahora $f_k \in AFS$.
7: **else** terminar ejecución algoritmo ABC-ED básico;
8: **end if**
9: **end for**
10: **end function**

En la **Fase de Obreras Empleadas**, por cada fuente f_k en AFS (Algoritmo 8), se usa función obtener-vecino-candidato (Algoritmo 12) para obtener aleatoriamente un vecino n_{f_k} que no se haya escogido de la vecindad N_{f_k} usando la expresión (9) y se deja n_{f_k} como escogido. Luego, en esta función se actúa dependiendo de la posición de n_{f_k} como sigue:

$$n^{\nu}_{f_k} = N_{f_k}[rand(1, \Lambda_{f_k})] \tag{9}$$

- Si la posición de n_{f_k} existe como fuente de comida f_n, no es necesario calcular el $fitness$, ya que este fue calculado en un proceso anterior. Luego, se actúa dependiendo en qué conjunto está f_n:

  - Si $f_n \in IFS$: Se asocia al vecino n_{f_k} la fuente f_n con su $fitness$. Este vecino reemplazará a f_k, ya que f_n aún le quedan vecinos para generar posibles fuentes o soluciones.
  - Si $f_n \notin IFS$: Se asocia al vecino n_{f_k} $fitness$ 0 (el mínimo posible), debido a que $f_n \in (AFS \vee EFS)$, por lo que no se desea que f_n reemplace a f_k. Si $f_n \in AFS$, implica que f_n está asociado a otra obrera en la actualidad, y desde el algoritmo ABC está definido que solo una obrera está asociada por cada fuente, por lo que se actuará con f_n en su turno que corresponda en la misma fase del ciclo. Si $f_n \in EFS$, no se debe volver a trabajar con una fuente ya agotada, no se obtendrán nuevas soluciones de ella.

- Si la posición de n_{f_k} no existe como fuente ni existe como posición rechazada ($\notin RP$), se calcula el $fitness$ a la posición de n_{f_k}. Si este $fitness \leq \mu$, esta posición se agrega a RP y se asigna $fitness$ 0 a n_{f_k}. En caso contrario, se asigna a n_{f_k} el $fitness$ calculado, en donde este vecino reemplazará a f_k.

- Si la posición de $n_{f_k} \in RP$, se asocia su $fitness$ a 0, debido a que no es de interés la posición y no se desea que se cumpla la selección greedy de la fase.

Luego, en la fase y con el vecino n_{f_k} obtenido, se realiza selección greedy entre $fit(n_{f_k})$ y μ. A diferencia del algoritmo ABC, en ABC-ED básico, se realiza selección greedy para reemplazo de una fuente entre el $fitness$ del vecino candidato y μ, y no por $fitness$ entre el candidato y la fuente actual, debido a que hace el modelo más eficiente. Para completa justificación y demostración, ver experimentación (sección 4), en particular experimento 2 parte 2 (sección 4.2.2).

La selección greedy se realiza como sigue:

- Si $fit(n_{f_k}) > \mu$: se obtiene la fuente asociada de n_{f_k} y se le asigna a f_n. Luego, en caso de que f_n no exista, la posición de n_{f_k} se crea como fuente y se le asigna a f_n junto con su $fitness$ ya calculado. Por último, se realiza el reemplazo de la fuente $f_k \in AFS$ por la fuente f_n usando función reemplazar-fuentecomida. En esta función, se realizan dos acciones a destacar. La primera acción es: para asignar en cual conjunto quedará f_k se verifica si f_k está agotada, debido a que es posible que f_k se agote justo antes de ser reemplazada. Se debe usar la misma condición de agotamiento que en la Fase de Obrera Exploradora. Luego, si f_k está agotada, $f_k \rightarrow EFS$, en caso contrario, $f_k \rightarrow IFS$, y $f_n \rightarrow AFS$. La segunda acción es: la probabilidad y rangos de ruleta de f_k se heredan a f_n, los cuales son calculados en función Calcular-Probabilidades (Algoritmo 9), debido a que no es una mala aproximación los valores heredados y se evita el costo de realizar el cálculo de probabilidades para todas las fuentes en AFS cada vez que se hace un reemplazo de fuente en la misma fase y ciclo del algoritmo.

- Si $fit(n_{f_k}) \leq \mu$: se aumenta el contador de *pruebas* de f_k y se mantiene en AFS.

Algoritmo 8 Pseudo-algoritmo ABC-ED básico. Función: Fase Obreras Empleadas.

```
function FASE-OBRERAS-EMPLEADAS( )
    FuenteComida f_k ← AFS[0];
    FuenteComida f_n;
    Vecino n_{f_k};
    for each f_k ∈ AFS do
        n_{f_k} ← OBTENER-VECINO-CANDIDATO(f_k);
        if fit(n_{f_k}) > μ then
            f_n ← n_{f_k}.fs;
            if EXISTE(f_n) == false then  // f_n es null
                f_n ← CREAR-NUEVA-FUENTECOMIDA(n_{f_k}.i, n_{f_k}.j);  // ahora fn ∈ NFS
                fit(f_n) ← fit(n_{f_k});  // fit(n_{f_k}) ya está calculado.
            end if
            REEMPLAZAR-FUENTECOMIDA(f_k, f_n);  // f_k ← f_n
        else pruebas(f_k) ← pruebas(f_k) + 1;
        end if
    end for
end function
```

En **Calcular Probabilidades**, por cada fuente f_k en AFS, se calcula su probabilidad $p(f_k)$ en base a su $fitness$ usando la expresión (8), para así construir una ruleta de selección (Algoritmo 9). Luego, se puede elegir estocásticamente una fuente por medio de la ruleta de selección en la Fase de Obreras Espectadoras.

Algoritmo 9 Pseudo-algoritmo ABC-ED básico. Función: Calcular Probabilidades.

```
function CALCULAR-PROBABILIDADES( )
    FuenteComida f_k ← AFS[0];
    rango ← 0;
    suma_fit ← OBTENER-SUMA-TOTAL-DE-FITNESS-DE-FUENTES-EN-AFS( );
    for each f_k ∈ AFS do
        p(f_k) ← fit(f_k)/suma_fit;  // acorde a exp. (8).
        f_k → RANGO-RULETA(rango, rango + p(f_k));
        rango ← rango + p(f_k);
    end for
end function
```

En la **Fase de Obreras Espectadoras**, se eligen SN fuentes de comida f_k estocásticamente. En donde, por cada fuente f_k se debe revisar siempre si su vecindad N_{f_k} está agotada (no le quedan vecinos por analizar), debido a que es posible que la fuente venga agotada desde la Fase de Obreras Empleadas o que la fuente pueda ser escogida más de una vez en la misma fase del ciclo y aun no se llega a la fase de exploración. A mayor $fitness$ de f_k y a mayor cantidad de vecinos de f_k que sean bordes, es más probable que se elija la fuente f_k. Luego, ya teniendo el vecino escogido de f_k, se actúa de la misma forma que en la Fase de Empleadas ya descrita.

Observar que hay una diferencia entre el algoritmo ABC y ABC-ED básico en cómo se obtiene una fuente estocásticamente. ABC está diseñado para converger a una sola solución final, la cual va incrementando su $fitness$, y por ende su probabilidad, en cambio, en ABC-ED básico se requiere encontrar múltiples soluciones finales, por lo que las probabilidades de las fuentes están más distribuidas durante la mayoría de su ejecución. Esto hace que por cada número aleatorio generado r, se busque una fuente $f_k \in AFS$ con rango de ruleta $[a, b]$, de tal forma que $r \in [a, b]$. Esto se realiza t veces, con $t \geq SN$, solo por la razón de que es probable que algunas fuentes vengan agotadas o se agoten en este proceso.

Algoritmo 10 Pseudo-algoritmo ABC-ED básico. Función: Fase Obreras Espectadoras.

```
 1: function FASE-OBRERAS-ESPECTADORAS( )
 2:     FuenteComida f_k ← AFS[0];
 3:     FuenteComida f_n;
 4:     Vecindad N_{f_k};
 5:     Vecino n_{f_k};
 6:     t ← 0; r ← 0;
 7:     while t < SN do
 8:         r ← rand(0, 1);
 9:         f_k ← OBTENER-FUENTECOMIDA-EN-RANGO-RULETA(r);
10:         N_{f_k} ← OBTENER-VECINDAD(f_k);
11:         if HAY-VECINOS-POR-ESCOGER(N_{f_k}) then
12:             t ← t + 1;
13:             n_{f_k} ← OBTENER-VECINO-CANDIDATO(f_k);
14:             if fit(n_{f_k}) > μ then
15:                 f_n ← n_{f_k}.fs;
16:                 if EXISTE(f_n) == false then  // f_n es null
17:                     f_n ← CREAR-NUEVA-FUENTECOMIDA(n_{f_k}.i, n_{f_k}.j);  // ahora fn ∈ NFS
18:                     fit(f_n) ← fit(n_{f_k});  // fit(n_{f_k}) ya está calculado.
19:                 end if
20:                 REEMPLAZAR-FUENTECOMIDA(f_k, f_n);  // f_k ← f_n
21:             else pruebas(f_k) ← pruebas(f_k) + 1;
22:             end if
23:         end if
24:     end while
25: end function
```

En la **Fase de Obrera Exploradora**, por cada fuente $f_k \in AFS$, se revisa si está agotada. Una fuente f_k está agotada siempre cuando la N_{f_k} no tenga más vecinos para escoger, ya que no se podrán obtener más soluciones por vecindad. Además, se mantiene la opción original de ABC respecto al parámetro de control *limit* para agotamiento de una fuente. A diferencia del ABC, se usa la comparación " $\geq$ ", ya que resulta más natural la comparación con *limit* representando la cantidad de vecinos de f_k, i.e., el máximo de pruebas de f_k sería la revisión de *limit* vecinos de f_k (Algoritmo 11).

Luego, si f_k está agotada, se obtiene la forma de reemplazo para f_k usando función obtener-forma-de-reemplazo (Algoritmo 17). Existen tres formas de reemplazo: Nueva Explora-

ción, Fuentes Inactivas y No Más Fuentes de Reemplazo.

En Algoritmo 17, si hay más posibles fuentes para crear e $IFS \neq \varnothing$, el parámetro de control $\varepsilon \in [0, 100]$ indica el porcentaje de probabilidad para que la forma de reemplazo sea por Nueva Exploración, donde $\varepsilon = 100$ indica que siempre se hará reemplazo por Nueva Exploración; $\varepsilon = 0$ indica que nunca se hará reemplazo por Nueva Exploración, puesto que se hará por Fuentes Inactivas; y para los valores de $\varepsilon \in]0, 100[$ indica que la elección es aleatoria entre ambas formas de reemplazo, con $\varepsilon\,\%$ de probabilidad que el reemplazo de una fuente agotada sea por Nueva Exploración.

Algoritmo 11 Pseudo-algoritmo ABC-ED básico. Función: Fase Obrera Exploradora.

```
1:  function FASE-OBRERA-EXPLORADORA( )
2:      FuenteComida f_k ← AFS[0];
3:      FuenteComida f_r;
4:      Vecindad N_{f_k};
5:      forma_reemplazo ← 0;
6:      for each f_k ∈ AFS do
7:          N_{f_k} ← OBTENER-VECINDAD(f_k);
8:          if pruebas(f_k) ≥ limit ∨ HAY-VECINOS-POR-ESCOGER(N_{f_k}) == false then
9:              obtener-forma-reemplazo:
10:             forma_reemplazo ← OBTENER-FORMA-DE-REEMPLAZO( );
11:             if forma_reemplazo == Nueva_Exploracion then
12:                 fr ← OBTENER-NUEVA-UNICA-FUENTECOMIDA-SELECTA( );
13:                 if EXISTE(f_r) then  // f_r ≠ null
14:                     REEMPLAZAR-FUENTECOMIDA-ABANDONADA(f_k, f_r);  // f_k ← f_r
15:                 else goto obtener-forma-reemplazo;
16:                 end if
17:             else if forma_reemplazo == Fuentes_Inactivas then
18:                 f_r ← OBTENER-FUENTECOMIDA-DE-IFS( );
19:                 REEMPLAZAR-FUENTECOMIDA-ABANDONADA(f_k, f_r);  // f_k ← f_r
20:             else if forma_reemplazo == No_Mas_Fuentes_de_Reemplazo then
21:                 REMOVER-DE-AFS(f_k);
22:                 AGREGAR-EN-EFS(f_k);
23:                 SN ← SN − 1;
24:             end if
25:         end if
26:     end for
27: end function
```

En la forma de reemplazo "Nueva Exploración" se obtiene una nueva fuente de comida $f_r \in NFS$ (con $fit(f_r) > \mu$) y se reemplaza $f_k \in AFS$ por f_r. Luego, $f_k \to EFS$ y $f_r \to AFS$. Como último caso, se puede dar que antes de llamar a la función obtener-nueva-unica-fuentecomida-selecta habían posibles posiciones para crear como fuentes de comida, pero en la llamada a la función, se analizó todas las posiciones faltantes, pero solo se encontraron posiciones rechazadas $\in RP$, por lo que se debe generar nuevamente una forma de reemplazo, la cual será Fuentes Inactivas o No Más Fuentes de Reemplazo.

En la forma de reemplazo "Fuentes Inactivas" se aprovechan fuentes de comida $f_r \in IFS$ ya creadas anteriormente, pero fueron reemplazadas por otras fuentes en el proceso. Sin embargo, está la posibilidad de que en la N_{f_r} hayan soluciones no analizadas aún. Se obtiene una fuente f_r removiéndola de IFS. Luego, $f_k \in AFS$ es reemplazada por f_r, quedando $f_k \rightarrow EFS$ y $f_r \rightarrow AFS$. Para remover una fuente f_r de IFS se usa una política $FIFO$ (primero en entrar, primero en salir), debido a su bajo costo y a que una fuente más tempranamente agregada a IFS tiene mayor probabilidad de que tenga más soluciones vecinas no analizadas, lo que puede ayudar a encontrar más rápidamente soluciones.

En la forma de reemplazo "No Mas Fuentes de Reemplazo" se llega al último caso, que significa que no existen más fuentes selectas por crear. Se introduce un mecanismo interno de término para el modelo, el cual no existe en ABC. Se remueve f_k de AFS, se agrega a EFS sin reemplazarla y se disminuye en uno SN. Si ABC-ED básico llega a que $SN = 0$, el algoritmo termina, ya que no tiene más fuentes de comida con que trabajar. Este mecanismo tiene la utilidad de poder saber, e.g.: cuantos ciclos y evaluaciones de $fitness$ máximas necesita el modelo para la imagen de entrada, haciendo valioso el mecanismo para experimentación. Sin embargo, es muy probable que este escenario de ultimo caso no sea ideal.

Por lo anterior, la condición que se debe cumplir para que ABC-ED básico se siga ejecutando (Algoritmo 6, punto 11) es: $ciclo < MCN \wedge SN > 0$.

3.3. Implementación

Para realizar la implementación del modelo, se definió como herramientas y ambiente de trabajo lo siguiente:

El lenguaje de programación C++ orientado a objeto, debido a que es eficiente y permite un gran control sobre la implementación. Sin embargo, carece de opciones rápidas de visualización gráfica, por lo que se requiere integrar con una herramienta para poder leer imágenes, procesar estas y visualizar datos de forma gráfica. MATLAB tiene estas características, y se puede integrar[1] con C/C++. Para la implementación se usó MATLAB Engine[2] Interface. Lo cual es una API que permite realizar comandos de MATLAB directamente desde C/C++. Esto requiere de una versión de MATLAB instalada para ejecutar la implementación. Con el motor de MATLAB se puede usar C/C++ Matrix Library[3] que trabaja con una estructura de datos $mxArray$ para poder recibir y enviar datos entre C/C++ y MATLAB. La biblioteca se puede encontrar en un directorio donde está instalado MATLAB llamado 'extern', la cual está disponible para las plataformas Linux, Windows x32 y x64 y Mac OS X. Debido a que actualmente todas las capacidades que entrega MATLAB Engine están disponibles solo para Windows y se tiene acceso a MATLAB x64, se utilizó Windows x64 como plataforma de trabajo. Para compilación[4] se usó Microsoft Visual C++ 2015 Profesional el que viene incluido en el IDE

[1] `http://www.mathworks.com/solutions/matlab-and-c/`

[2] `http://www.mathworks.com/help/matlab/calling-matlab-engine-from-c-c-and-fortran-programs.html`

[3] `http://www.mathworks.com/help/matlab/cc-mx-matrix-library.html`

[4] `http://www.mathworks.com/support/compilers`

Visual Studio[5] 2015 de forma gratuita. De todas formas, solo se requiere configuración para poder generar el ejecutable en Windows x32 junto con la biblioteca dada por MATLAB x32 y así ejecutarlo correctamente y poder usar MATLAB Engine.

Para poder realizar el cálculo de $fitness$ usando un operador de método clásico en ABC-ED básico, se implementaron estos operadores en el prototipo, los cuales dan resultados idénticos a los dados utilizando estos operadores directamente en MATLAB. Al igual que MATLAB, para el cálculo de gradiente de los pixeles que son marco de la imagen de entrada, se considera que los valores de intensidad afuera del marco de la imagen son asumidos como igual al valor más cercano de ese punto.

Se describe de forma breve y simple el diseño de implementación del modelo en Anexo B.

[5] `https://www.visualstudio.com`

4. Experimentación

En este capitulo, se muestra justificadamente las decisiones tomadas en la construcción estructural y algorítmica del modelo ABC-ED básico, junto con su calibración de parámetros de control para evaluar su comportamiento de eficiencia. Para lo anterior, se realiza un plan de prueba, en donde se definen las métricas de evaluación y se declaran los experimentos a realizar, para luego definirlos y realizarlos.

4.1. Plan de Prueba

Primero, se definen las métricas de evaluación de eficacia y eficiencia para el modelo. El modelo trata con el problema de localización eficiente en la detección de bordes, y la identificación o precisión si un pixel es borde, se calcula al obtener el $fitness$ de una fuente, usando un operador de método clásico. Luego, para obtener la eficacia del modelo, se debe comparar la matriz binarizada de salida $OM_{\text{ABC-ED}}$ dada por el modelo ABC-ED básico, con la matriz binarizada de salida $OM_{\text{C-ED}}$ dada por sólo utilizar un método clásico (C-ED) bajo el mismo ambiente (I y μ). Para lo anterior, se usa la métrica de la Distancia de Hamming (DH), orientada a evaluar entre dos matrices binarizadas de igual tamaño $OM_{\text{ABC-ED}}$ y $OM_{\text{C-ED}}$, en donde DH analiza cada pixel de igual posición entre ambas matrices entregando el número de pixeles que tienen valores diferentes. Luego, con $ciclo \in [0, MCN[$ y $\oplus$ el operador lógico XOR, se tiene y se define:

- $DH_{\text{ABC-ED}} = OM_{\text{ABC-ED}} \oplus OM_{\text{C-ED}} = \sum_{i=0}^{r-1}(\sum_{j=0}^{c-1} OM_{\text{ABC-ED}}[i][j] \oplus OM_{\text{C-ED}}[i][j])$.
- $|OM_{\text{C-ED}}|_1 := \sum_{i=0}^{r-1}(\sum_{j=0}^{c-1} OM_{\text{C-ED}}[i][j])$, el número total de pixeles identificados como borde por un método clásico en su matriz de salida.
- $eficacia_{\text{ABC-ED}} := |OM_{\text{C-ED}}|_1 - DH_{\text{ABC-ED}}$, el número de pixeles iguales entre las salidas $OM_{\text{ABC-ED}}$ y $OM_{\text{C-ED}}$.
- $DH_{\text{ABC-ED}}(ciclo) := OM_{\text{ABC-ED}}(ciclo) \oplus OM_{\text{C-ED}}$, el número de pixeles distintos entre la matriz de salida del $ciclo$ del modelo $OM_{\text{ABC-ED}}(ciclo)$ y $OM_{\text{C-ED}}$.
- $\Gamma_{\text{ABC-ED}} :=$ el número de fuentes de comida creadas por el modelo.
- $\Gamma_{\text{ABC-ED}}(ciclo) :=$ el número de fuentes creadas hasta el $ciclo$ del modelo.

Así, el porcentaje de eficacia para el modelo está dado por la expresión (10). Y el porcentaje de eficacia hasta cada $ciclo$ del modelo está dada por la expresión (11).

$$\%\, eficacia_{\text{ABC-ED}} := (\frac{eficiacia_{\text{ABC-ED}}}{|OM_{\text{C-ED}}|_1}) \times 100 = (1 - \frac{DH_{\text{ABC-ED}}}{|OM_{\text{C-ED}}|_1}) \times 100 \tag{10}$$

$$\%\, eficacia_{\text{ABC-ED}}(ciclo) := (1 - \frac{DH_{\text{ABC-ED}}(ciclo)}{|OM_{\text{C-ED}}|_1}) \times 100 \tag{11}$$

Además, como en el modelo las fuentes de comida f_k que se crean son todas selectas ($fit(f_k) > \mu$), basta con saber el número de todas las fuentes creadas por ABC-ED básico, el cual es igual al número de pixeles localizados e identificados como borde. Luego, el porcentaje

de eficacia dado en la expresión (10) es equivalente a la expresión (12). Por ende, el porcentaje de eficacia hasta cada *ciclo* del modelo dado en la expresión (11) es equivalente a expresión (13).

$$\%\,eficacia_{ABC-ED} = (\frac{\Gamma_{ABC-ED}}{|OM_{C-ED}|_1}) \times 100 \tag{12}$$

$$\%\,eficacia_{ABC-ED}(ciclo) = (\frac{\Gamma_{ABC-ED}(ciclo)}{|OM_{C-ED}|_1}) \times 100 \tag{13}$$

Para la evaluación de la eficiencia del modelo, en su prototipo se capturan datos en cada *ciclo* de ejecución. Los datos capturados representan aspectos de comportamiento del modelo. Desde la definición 2, se adiciona la medida respectiva como porcentaje. Además de estas definiciones o medidas, se considera para evaluar la eficiencia del modelo el *tiempo de ejecución* (t) y el número máximo de ciclos (MCN). Luego, se define:

1. $ciclos(ciclo)$:= los ciclos que ABC-ED básico ejecuta.

2. $\Gamma(ciclo)$:= el número de fuentes creadas hasta cada *ciclo*.
   $\%\Gamma(ciclo) := 100 \times \Gamma(ciclo)/r \times c$.

3. $RP(ciclo)$:= el número de posiciones rechazadas encontradas de IM hasta cada *ciclo*.
   $\%RP(ciclo) := 100 \times RP(ciclo)/(r \times c - \Gamma(ciclo))$.

4. $PA(ciclo)$:= el número de pixeles analizados de IM hasta cada *ciclo*.
   $PA(ciclo) := \Gamma(ciclo) + RP(ciclo)$.
   $\%PA(ciclo) := 100 \times PA(ciclo)/r \times c$.

5. $Fit(ciclo)$:= el número de cálculos de $fitness$ de IM realizados hasta cada *ciclo*.
   $Fit(ciclo) := Fit_N(ciclo) + Fit_E(ciclo)$.
   $\%Fit(ciclo) := 100 \times Fit(ciclo)/r \times c$.

6. $Fit_N(ciclo)$:= el número de cálculos de $fitness$ realizados por vecindad hasta cada *ciclo*. Se obtiene usando un contador global en Algoritmo 12, entre el punto 22 y 23.
   $\%Fit_N(ciclo) := 100 \times Fit_N(ciclo)/Fit(ciclo)$.

7. $Fit_E(ciclo)$:= el número de cálculos de $fitness$ realizados por exploración hasta cada *ciclo*. Se obtiene usando un contador global en Algoritmo 14, entre el punto 11 y 12.
   $\%Fit_E(ciclo) := 100 \times Fit_E(ciclo)/Fit(ciclo)$.

Las definiciones o medidas anteriores, serán representadas por medio de gráficos y tablas resumidas. Se usan dos gráficos: el gráfico de porcentaje de completitud y el gráfico de número. En el gráfico de porcentaje de completitud, se compara cada medida como porcentaje vs los *ciclos* del modelo. La leyenda de este gráfico se define en la Figura 11a. En el gráfico de número, se compara cada medida como número vs los *ciclos* del modelo. La leyenda de este gráfico se define en la Figura 11b. En el gráfico de porcentaje, se agrega el porcentaje de eficacia hasta cada *ciclo* del modelo, en donde se visualiza la convergencia de su eficacia. Para ambos gráficos, se agregan cuatro líneas verticales que corresponden al ciclo en que el modelo alcanza ($\geq$) por primera vez el porcentaje de eficacia observada respectiva a la línea (80 %, 90 %, 95 % y 100 %). Para demostrar el estado inicial del modelo, los valores capturados para $ciclo = 0$ corresponden al estado después de la inicialización del modelo. Los valores para $ciclo = 1$ corresponden al

estado del modelo del *ciclo* $\in [0, 1[$ y así sucesivamente. Las fuentes creadas en inicialización se consideran como exploración.

Las imágenes de entrada para experimentación son las siguientes: Lenna, con dimensión de 512x512 se presentan en la Figura 8a, Cameraman, con dimensión 256x256 se presenta en la Figura 8b y Peppers, con dimensión 512x512 se presenta en la Figura 8c. Estas son imágenes bien conocidas que se usan en investigación dentro del área de procesamiento de imágenes.

(a) Lenna. (b) Cameraman. (c) Peppers.

Figura 8: imágenes de entrada para experimentación.

El *tiempo de ejecución* que se analiza del modelo es usando la unidad de medida segundos, y es calculado en el prototipo usando la función *clock*. El tiempo es tomado desde antes de la inicialización del modelo hasta la detención o fin de ejecución de éste, luego de generar la imagen binarizada de salida final. El *tiempo de ejecución* incluye la captura de las medidas de evaluación del modelo, para poder conseguir el *tiempo de ejecución* que se requirió para llegar a cada porcentaje de eficacia observada, pero no se considera que se genere y visualice la imagen binarizada de salida en cada ciclo.

Ya con todo lo necesario definido, se declaran los experimentos con sus propósitos, para ser definidos y realizados al aplicar el plan de prueba.

1. **Experimento 1:** analizar la completitud, estabilidad, convergencia, aspectos generales y crear punto de partida o comparación para futuros experimentos.

2. **Experimento 2:** analizar el uso de solo fuentes de comida selectas versus fuentes corrientes. Para la mejor opción anterior, analizar la mejor condición de selección greedy para el modelo. Se usa resultados de experimento 1.

3. **Experimento 3:** analizar la mejor calibración de parámetros de control para el modelo, acorde a resultados de experimento 1 y 2.

4.2. Aplicación de Plan de Prueba

4.2.1. Experimento 1

Para este experimento se utiliza la Figura 8a. **Primero**, se realiza una detección de bordes por cada método clásico (C-ED): Roberts, Sobel y Prewitt. Para cada C-ED se calibra el valor

del umbral μ. La calibración se realiza con un valor de μ inicial y se observa la salida del C-ED. Si es necesario, se ingresa un nuevo valor para μ y se observa su nueva salida. Este paso anterior, se realiza hasta obtener una salida aceptable. Este μ calibrado por cada C-ED es usado en ABC-ED básico para el m respectivo. **Segundo**, se realiza la detección de bordes con el modelo usando los parámetros definidos en la Tabla 1a con un valor estándar por ser el punto de partida para futura calibración. Se realizaron 10 ejecuciones para cada conjunto de parámetros. Los resultados que se presentan por cada conjunto de parámetros, corresponden a los de la ejecución que tuvo como cantidad de *ciclos* ejecutados la mayor frecuencia obtenida *(moda estadística)* para cada porcentaje de eficacia observada. **Tercero**, se presentan los resultados obtenidos. En la Tabla 1b se muestran los datos finales capturados. La primera fila de dato *ciclo* corresponde al *ciclo* en que el modelo llegó al 100 % de eficacia con respecto a cada m. Luego, en la misma tabla se presentan los datos de porcentaje definidos en función del *ciclo* de la primera fila. En las ultimas dos filas de la misma tabla, se considera el total de ejecuciones para el mismo conjunto de parámetros. En la penúltima y ultima fila, se muestra el rango mínimo y máximo de número de *ciclos* cuando el modelo llegó al 100 % de eficacia y termina su ejecución por su mecanismo interno de término (cuando $SN = 0$) respectivamente. En la Figura 9, por cada m se muestra la imagen de salida del 100 % de eficacia dada por el modelo y el C-ED respectivo, **la cual es idéntica**. En la Figura 10, se muestra la imagen de salida de los tres primeros porcentajes de eficacia observada para m = Sobel. En la Figura 12, por cada m se muestran los gráficos de porcentaje y número. En la Tabla 2, se presentan los valores de los porcentajes de eficacia observada de los gráficos de las Figuras 12c y 12d, con m = Sobel. **Cuarto** y último, se analiza los resultados obtenidos.

Parámetros de entrada ABC-ED básico			
Parámetros	Roberts	Sobel	Prewitt
SN	$\sqrt{r \times c}$	$\sqrt{r \times c}$	$\sqrt{r \times c}$
MCN	200	200	200
$limit$	8	8	8
ε	50	50	50
m	Roberts	Sobel	Prewitt
μ	40	175	130

(a) Parámetros de entrada para modelo.

Datos de salida ABC-ED básico por cada m			
Datos	Roberts	Sobel	Prewitt
$ciclo$	93	107	106
$\%\Gamma(ciclo)$	4.75311	5.52559	5.45082
$\%RP(ciclo)$	99.8454	99.8361	99.7987
$\%PA(ciclo)$	99.8528	99.8451	99.8096
$\%Fit(ciclo)$	99.8528	99.8451	99.8096
$\%Fit_N(ciclo)$	10.1754	10.896	10.6052
$\%Fit_E(ciclo)$	89.8246	89.104	89.3948
$\%eficacia(ciclo)$	100	100	100
$[min, max]$	[91,97]	[105,111]	[103,108]
$ciclo(SN = 0)$	[98,99]	[114,115]	[112,113]

(b) Datos de salida de ABC-ED básico por cada m.

Tabla 1: datos de entrada y salida de experimento 1 para modelo por cada m.

Se demuestra que el modelo es completo al llegar al 100 % de eficacia para cierto *ciclo*, independientemente del parámetro m para cálculo de $fitness$.

De la Tabla 1b, se demuestra que el modelo es muy estable para el mismo conjunto de parámetros al tener un rango tan pequeño de la cantidad de *ciclos* necesarios para llegar a la eficacia que se desea, e incluso de su mecanismo interno de término.

De la Figura 9a (m = Roberts), se vislumbra que la imagen presenta mayor cantidad de bordes discontinuos que las Figuras 9b (m = Sobel) y 9c (m = Prewitt), por lo que el modelo requiere de mayor exploración para localizar estos bordes, ya que no puede llegar por vecindad.

(a) m = Roberts. (b) m = Sobel. (c) m = Prewitt.

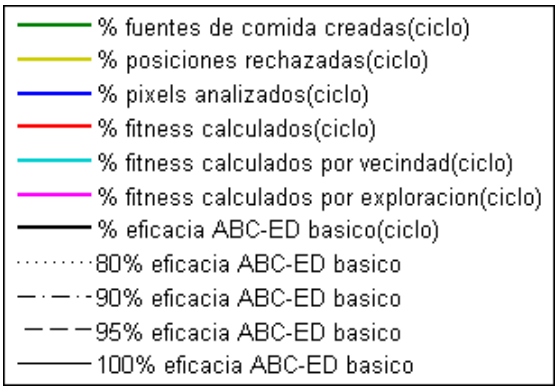
Figura 9: imágenes de salida de experimento 1 por cada m.

Lo anterior se confirma y se ve más claramente en el gráfico de la Figura 12a (m = Roberts) al compararlo con los gráficos de las Figuras 12c (m = Sobel) y 12e (m = Prewitt), en donde para el gráfico de la Figura 12a (m = Roberts) se aprecia que la distancia entre el porcentaje de exploración y de vecindad es más amplia que en los otros dos gráficos. Por lo tanto, el modelo requiere de una mayor localización por exploración en una identificación de bordes más discontinua. Luego, para obtener una identificación de bordes más continua, es suficiente con variar el valor de μ para el mismo m, con el fin de obtener diferentes niveles de identificaciones de bordes, entre discontinuos hasta más continuos. Se elige m = Sobel y $\mu = 175$ (ya evaluado) para continuar con otros experimentos.

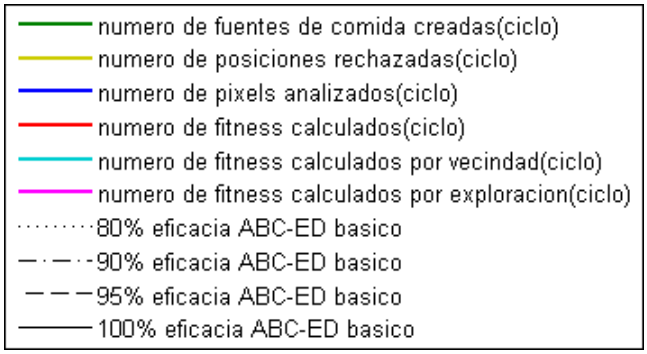
De los gráficos de porcentaje y número de la Figura 12, se aprecia visualmente el comportamiento de eficiencia del modelo con respecto a su eficacia por cada m, en donde a partir de una eficacia mayor a 80 %, la eficiencia empieza a empeorar, en términos de cantidad de cálculos de $fitness$ (Fit) de IM. Lo anterior sucede con mayor énfasis cuando el modelo requiere de una mayor exploración para localizar bordes, como se observa en los gráficos de las Figuras 12a y 12b (m = Roberts). Por lo tanto, la localización por exploración tiene un mayor costo en Fit que una localización por vecindad en una detección de bordes más discontinua, por lo que se debe experimentar con el parámetro de control ε para encontrar la mejor eficiencia del modelo versus la eficacia.

(a) %eficacia = 80, ciclo = 63. (b) %eficacia = 90, ciclo = 78. (c) %eficacia = 95, ciclo = 86.

Figura 10: imágenes de salida de experimento 1: m = Sobel, $\mu = 175$.

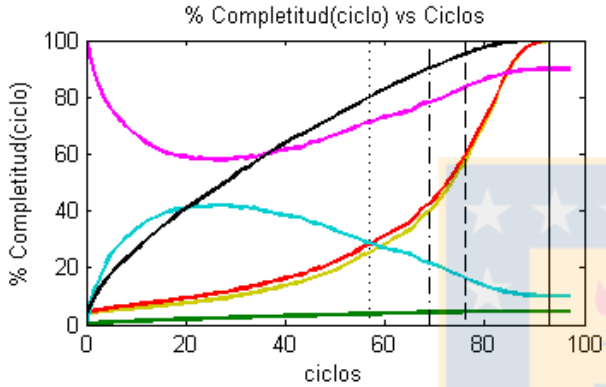


(a) Legenda para gráficos de % de completitud.

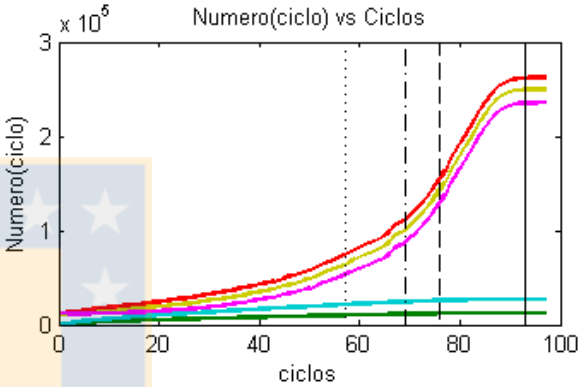


(b) Legenda para gráficos de número.

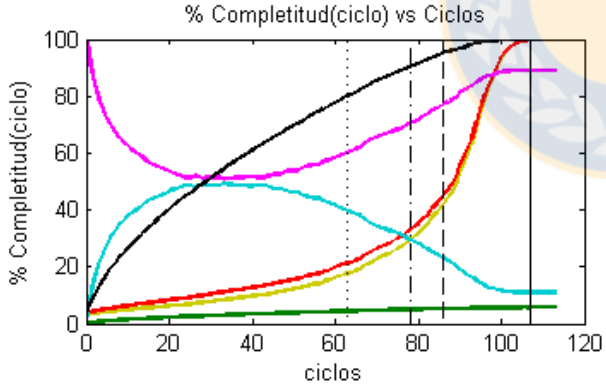
Figura 11: legendas para cada gráfico.



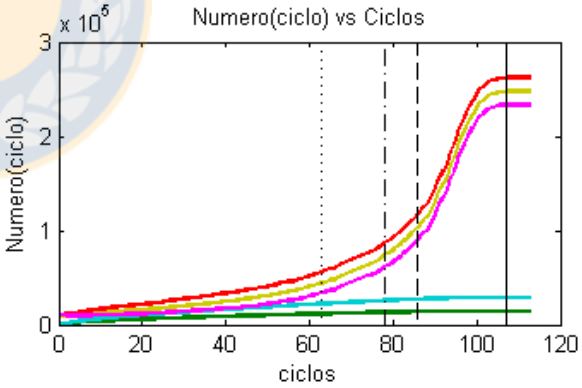
(a) Gráfico de porcentaje. m = Roberts.



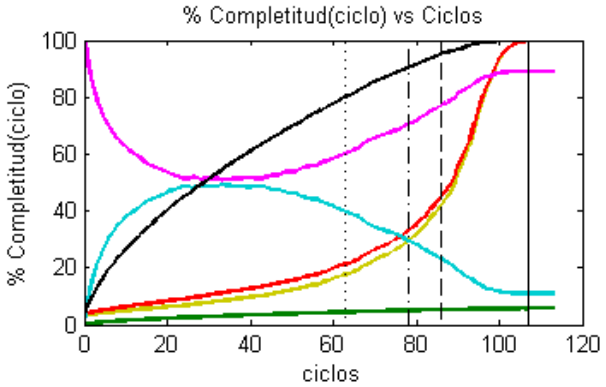
(b) Gráfico de número. m = Roberts.



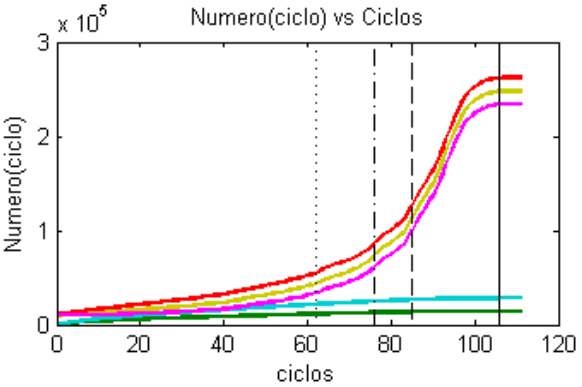
(c) Gráfico de porcentaje. m = Sobel.



(d) Gráfico de número. m = Sobel.



(e) Gráfico de porcentaje. m = Prewitt.



(f) Gráfico de número. m = Prewitt.

Figura 12: gráficos resultantes de ABC-ED básico para cada m respectivo.

En las Figuras 10a, 10b, 10c y 9b se visualizan las imágenes resultantes del modelo para los porcentajes de eficacia observada 80 %, 90 %, 95 % y 100 % respectivamente, con $m =$ Sobel y $\mu = 175$. Se demuestra que ya en el 80 % de eficacia, el modelo tiene detectada la estructura de bordes representativa de la imagen con respecto al 100 % de eficacia. Por medio de los gráficos de las Figuras 12c y 12d y cuantitativamente por la Tabla 2, se demuestra que el modelo no necesita más de un 21.3 % de cálculos de $fitness$ (Fit) y pixeles analizados (PA) del total de pixeles de IM. Estas dos medidas anteriores son idénticas, debido a que se manejan posiciones rechazadas (RP), por lo que para cada posición de pixel analizada, se calcula su $fitness$ una sola vez. Luego, el modelo sigue detectando cada vez menos por vecindad y más por exploración los bordes que están más discontinuos en la imagen. En la última fila de la Tabla 2 se presenta el *tiempo de ejecución* en segundos para cada porcentaje de eficacia observada.

Datos capturados de eficiencia				
Datos	80 %	90 %	95 %	100 %
$ciclo$	63	78	86	107
$\%\Gamma(ciclo)$	4.42123	4.99535	5.25322	5.52559
$\%RP(ciclo)$	17.6289	29.5966	41.6627	99.8361
$\%PA(ciclo)$	21.2708	33.1135	44.7273	99.8451
$\%Fit(ciclo)$	21.2708	33.1135	44.7273	99.8451
$\%Fit_N(ciclo)$	39.7113	29.4453	23.0917	10.896
$\%Fit_E(ciclo)$	60.2887	70.5547	76.9083	89.104
$t(ciclo)$	1.052	1.335	1.511	2.497

Tabla 2: datos de eficiencia de cada porcentaje de eficacia observada para $m =$ Sobel.

4.2.2. Experimento 2

Este experimento consta de dos partes. Parte 1: analizar el uso de solo fuentes de comida f_k selectas ($fit(f_k) > \mu$) versus usar fuentes corrientes (sin importar el valor de $fitness$), y comparar qué opción de creación de fuente es mejor para el modelo. Luego, para la mejor opción anterior, se continúa con la segunda parte del experimento. Parte 2: analizar la mejor condición de selección greedy para el modelo.

Para la **primera parte** del experimento, se usa la Figura 8a usada en experimento 1 y se compara sus resultados obtenidos usando solo fuentes selectas (con $m =$ Sobel y $\mu = 175$), con la experimentación de usar fuentes de comida corrientes. **Primero**, para que el modelo trabaje con fuentes corrientes, se debe verificar que la posición de IM no exista como fuente antes de crearla, para evitar duplicidad. Además, como para crear una fuente corriente no importa su valor de $fitness$, el modelo no debe manejar posiciones rechazadas. Sin embargo, solo para análisis se compara que para la fuente f_k creada, si $fit(f_k) \leq \mu$ es verdadero, se incrementa el contador global RP, pero la posición no se guarda como rechazada. Luego, se requiere pequeños cambios en el modelo para poder usar fuentes corrientes, donde la configuración es:

- En el Algoritmo 7 punto 5 y el Algoritmo 11 punto 12, se debe cambiar la función a llamar del Algoritmo 14 por el Algoritmo 16, la cual creará nuevas y únicas fuentes corrientes, en donde luego de calcular el $fitness$ a la fuente f_s creada, se debe incrementar el contador global RP si $fit(f_s) \leq \mu$.

- En el Algoritmo 12, entre puntos 20 y 29, se debe reemplazar por un "else" para el "if" de punto 14. Este else es el siguiente:
  - $n_{f_k}.fitness \leftarrow$ Calcular-Fitness($n_{f_k}.i$, $n_{f_k}.j$).
  - Incrementar contador global Fit_N para análisis.
  - Incrementar contador global RP si $n_{f_k}.fitness \leq \mu$ para análisis.
- Debido a que se crean fuentes corrientes, se debe medir el porcentaje de eficacia del modelo usando la expresión (10) y la expresión (11) para cada ciclo.

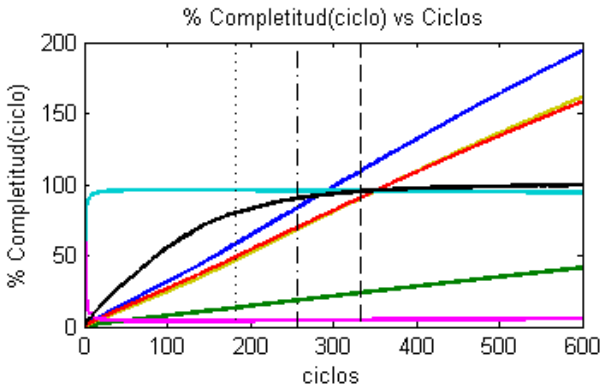
Segundo, ya con los cambios necesarios al modelo, se realiza la detección con los parámetros de entrada presentados en la Tabla 1a, con m = Sobel y $\mu = 175$, pero cambiando $MCN = 600$. Se realizaron 5 ejecuciones con el mismo conjunto de parámetros. Se eligió la ejecución que requirió menor cantidad de *ciclos* para llegar al 80 % de eficacia. **Tercero**, se presentan los resultados obtenidos. En la Tabla 3, se presentan los datos capturados para cada porcentaje de eficacia observada. En la Figura 13, se presentan las imágenes de salida para los tres primeros porcentajes de eficacia observada. En la Figura 14, se presenta el gráfico de porcentaje y el gráfico de número hasta la detención del modelo ($ciclo = MCN$). **Cuarto** y último, se analiza los resultados obtenidos.

(a) %eficacia=80, ciclo=183. (b) %eficacia=90, ciclo=256. (c) %eficacia=95, ciclo=333.

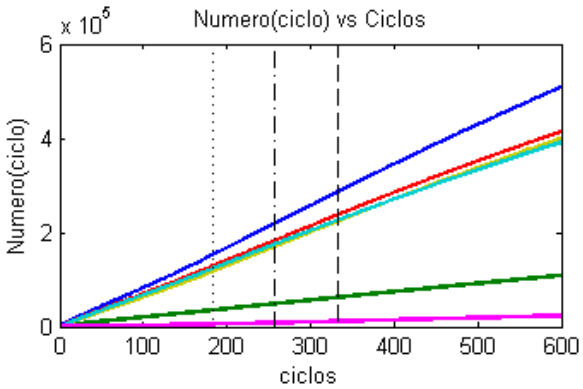
Figura 13: imágenes de salida de experimento 2, parte 1.

En las Figuras 13a, 13b, 13c, 9b, se visualizan la imágenes resultantes del modelo para los porcentajes de eficacia observada 80 %, 90 %, 95 % y 100 % respectivamente. Al comparar con las imágenes de salida de las Figuras 10a, 10b, 10c y 9b respectivamente, en donde el modelo usa solo fuentes selectas, se observa que para las imágenes antes del 100 % de eficacia presentan algunas diferencias, debido a que si bien es la misma eficacia, esto fue localizado en función de conceptos distintos de creación de fuente, lo que genera un diferente comportamiento al modelo, pero para ambos conceptos se localiza la estructura de bordes representativa de la imagen.

Del gráfico de porcentaje de la Figura 14a y la Tabla 3, se observa que al usar fuentes corrientes el modelo realiza una localización principalmente por vecindad, debido a que es más probable que se haga reemplazo de fuentes por selección greedy al trabajar con gran diversidad de valores de $fitness$, donde la mayoría de las fuentes tiene un valor de $fitness$ menor o igual a μ.



(a) Gráfico porcentaje experimento 2 parte 1.



(b) Gráfico número experimento 2 parte 1.

Figura 14: gráficos resultantes de experimentación 2 parte 1.

Datos capturados de eficiencia				
Datos	80 %	90 %	95 %	100 %
$ciclo$	183	256	333	813
$\%\Gamma(ciclo)$	13.3556	18.4551	23.7564	54.3777
$\%RP(ciclo)$	47.0087	68.1750	90.4072	210.107
$\%PA(ciclo)$	58.2844	83.2378	109.418	252.875
$\%Fit(ciclo)$	49.3534	69.7586	90.9107	204.042
$\%Fit_N(ciclo)$	95.8830	95.5438	95.2236	93.6614
$\%Fit_E(ciclo)$	4.11700	4.45620	4.77640	6.33860
$t(ciclo)$	3.193	4.497	5.987	14.942

Tabla 3: datos de eficiencia de cada porcentaje de eficacia observada de experimento 2, parte 1.

De la Tabla 3, al compararla con la Tabla 2, se demuestra que el uso de fuentes corrientes es menos eficiente, debido a que recién para llegar al 80 % de eficacia, se requiere de 183 *ciclos*, crear un 13,3 % de posiciones de IM como fuente (Γ) al analizar (PA) más de la mitad de los pixeles de IM, en donde se analizan posiciones y realizan cálculos de $fitness$ (Fit) repetidos, debido a que no se manejan las posiciones rechazadas (RP), las que fueron casi la mitad de IM. Al usar fuentes corrientes y para un 80 % de eficacia, estas cinco medidas ($ciclo$, Γ, PA, RP y Fit) son mayores que para el 95 % de eficacia usando sólo fuentes selectas. De hecho, el número de *ciclo* y el *tiempo de ejecución* (t) usando fuentes corrientes es más del doble que al usar solo fuentes selectas. Incluso, para llegar al 100 % de eficacia usando fuentes selectas, se requiere de menos ciclos y *tiempo de ejecución* que para llegar al 80 % de eficacia al usar fuentes corrientes.

Por los análisis anteriores, el uso de fuentes corrientes hace que el modelo ABC-ED básico sea menos eficiente en el manejo de recursos, cantidad de análisis de la imagen de entrada y *tiempo de ejecución* para llegar a la eficacia deseada, debido a que como los bordes en una imagen generalmente forman líneas continuas, es una mejor opción partir la búsqueda en la vecindad de una fuente selecta, ya que es más probable que en esta vecindad hayan posiciones selectas, que partir la búsqueda desde una fuente no selecta, en donde es más probable que sus vecinos sean no selectos también, lo cual no es necesario de analizar, ya que se gasta mucho tiempo analizando pixeles que no son bordes y a la vez, vecinos de estos pixeles que es probable que tampoco sean bordes.

Traduciendo los resultados a la naturaleza, una fuente selecta puede ser considerada como una flor, la cual posee de néctar, y por ende, una abeja de miel le puede extraer este néctar a la flor para que la colonia de abejas pueda trabajar y alimentarse de ella. Por otro lado, una fuente no selecta, no es una flor, y las abejas no van a trabajar ni alimentarse con algo que no le puedan extraer el néctar, lo que no tiene sentido usar este tipo de fuente. Luego, el modelo es más acorde al comportamiento de las abejas de miel.

Ya teniendo presente que la mejor opción es usar solo fuentes selectas, se procede con la **segunda parte** del experimento, la cual es analizar qué condición de selección greedy para reemplazo de una fuente es mejor para el modelo, la cual está ubicada en la Fase de Obreras Empleadas (Algoritmo 8, punto 7) y en Fase de Obreras Espectadoras (Algoritmo 10, punto 14).

Primero, se presentan las posibles condiciones de selección greedy a analizar:

a) $[fit(n_{f_k}) > fit(f_k)]$

b) $[[fit(n_{f_k}) > fit(f_k)] \wedge [fit(n_{f_k}) > \mu]]$

c) $[[fit(n_{f_k}) > fit(f_k)] \vee [fit(n_{f_k}) > \mu]]$

d) $[fit(n_{f_k}) > \mu]$

La condición **b** y **c**, están compuestas de dos condiciones, las cuales son la condición **a** y la condición **d**. Por lógica, la condición **b** solo es verdadera cuando **sus** condiciones **a** y **d** son verdaderas. La condición **c** es verdadera cuando **su** condición **a** o **d** es verdadera.

En la función obtener-vecino-candidato(f_k) (Algoritmo 12), cuando la posición del vecino escogido (n_{f_k}) de la fuente f_k es no selecta ($fit(n_{f_k}) \leq \mu$), se asigna $fit(n_{f_k}) \leftarrow 0$ (el mínimo posible), ya que esta posición es rechazada, por lo que no es de interés y no se desea que exista la posibilidad de que se cumpla la selección greedy para reemplazo de f_k. Por ende, si la condición $fit(n_{f_k}) > \mu$ es **falsa**, implica que la condición $fit(n_{f_k}) > fit(f_k)$ **siempre** será **falsa**.

Considerando lo anterior,

- La condición **c** solo y siempre será verdadera si **su** condición **d** es verdadera, sin importar el valor de verdad de **su** condición **a**. Luego, las condiciones **c** y **d** son equivalentes para el modelo.
- La condición **b** no siempre será verdadera si **su** condición **d** es verdadera, esto depende del valor de verdad de **su** condición **a**, pero si **su** condición **d** es falsa, siempre **su** condición **a** será falsa. Luego, las condiciones **a** y **b** son equivalentes para el modelo.

Por lo tanto, solo se debe analizar, por simplicidad de expresión, las condiciones **a** y **d**.

Luego, para este experimento, se usa los resultados del experimento 1, en donde el modelo usa la condición **d** y solo fuentes selectas, y se comparan con los resultados del modelo usando el mismo conjunto de parámetros de entrada de la Tabla 1a para $m =$ Sobel, pero usando la condición **a**.

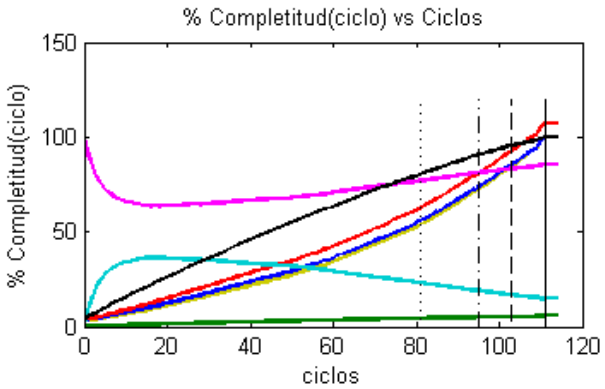
Segundo, para una comparación justa, al igual que en experimento 1, se realizaron 10 ejecuciones con la configuración ya mencionada, y los resultados que se presentan son de la ejecución que tuvo como cantidad de *ciclos* ejecutados la mayor frecuencia obtenida *(moda estadística)* para cada porcentaje de eficacia observada. **Tercero**, se presentan los resultados obtenidos. En la Figura 15, se visualizan las imágenes resultantes dadas por el modelo para los tres primeros porcentajes de eficacia observada. En la Figura 16, se presentan los gráficos de porcentaje y número dado por los datos capturados hasta cuando $SN = 0$, en donde los datos de los porcentajes de eficacia observada se presentan cuantitativamente en la Tabla 4. En las Tablas 5 y 6, se presentan de forma extendida los datos de *ciclo* y *tiempo de ejecución* (t), los cuales consideran el total de ejecuciones al usar condición **a** y **d** respectivamente. La Tabla 6 es una extensión del experimento 1. **Cuarto** y último, se analiza los resultados obtenidos.

(a) %eficacia=80, ciclo=81. (b) %eficacia=90, ciclo=95. (c) %eficacia=95, ciclo=103.

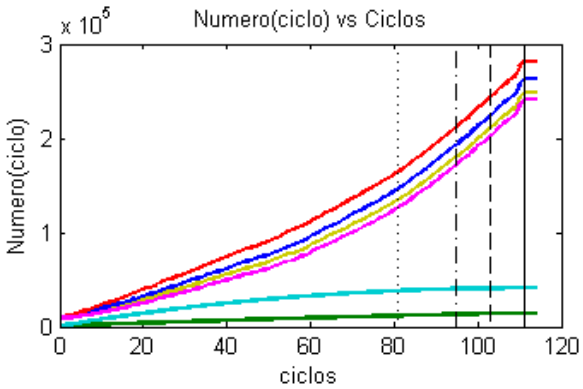
Figura 15: imágenes de salida de experimento 2, parte 2.

En las Figuras 15a, 15b, 15c y 9b, se presentan la imágenes resultantes dadas por el modelo usando la condición **a** para los porcentajes de eficacia observada 80 %, 90 %, 95 % y 100 % respectivamente. Al comparar respectivamente con las imágenes de las Figuras 10a, 10b, 10c y 9b, en donde el modelo usa la condición **d**, se observa que las imágenes anteriores al 100 % de eficacia presentan algunas diferencias, debido a que si bien presentan la misma eficacia, se usó una condición de reemplazo para una fuente distinta, por lo que el modelo genera una diferente localización en su proceso, pero para ambas condiciones se localiza la estructura de bordes representativa de la imagen. Sin embargo, cabe destacar, que al usar la condición **a** se tiene por cada fuente selecta f_k una localización de solo las mejores posiciones en la vecindad de f_k, y no todas las soluciones vecinas n_{f_k} que sean bordes. Lo anterior, hace que el modelo en etapas iniciales detecte bordes más finos en toda la estructura de bordes de la imagen, pero muy descontinuados, y posteriormente vayan siendo más gruesos y conectándose en una localización principalmente por exploración. Como lo anterior pasa muy temprano y se tiene muchos bordes discontinuos, no lo hace un escenario ideal. Luego, podría ser mejor usar una técnica de adelgazamiento de bordes al final de la ejecución del modelo, en donde solo se requiere recorrer todas las fuentes creadas, y no todo OM_{ABC-ED}; lo que queda propuesto como trabajo futuro.

En los gráficos de porcentaje y número de las Figuras 16a y 16b, se observa el comportamiento del modelo al usar la condición **a**. Al compararlos con los gráficos de las Figuras 12c y 12d, los cuales demuestran el comportamiento del modelo al usar la condición **d**, se observa que el modelo al usar la condición **a** requiere de más *ciclos* para llegar a los porcentajes de eficacia



(a) Gráfico porcentaje experimento 2 parte 2.



(b) Gráfico número experimento 2 parte 2.

Figura 16: gráficos resultantes de experimento 2 parte 2.

observada respectivamente, encuentra más posiciones rechazadas tempranamente, requiriendo una mayor cantidad de pixeles analizados y una mayor cantidad de cálculos de $fitness$. Además, el modelo realiza una mayor localización por exploración que por vecindad, en donde estas medidas tienen una diferencia mayor de amplitud que usando la condición **d** antes de llegar a los porcentajes de eficacia observada. Luego de esto, la diferencia de amplitud es menor, puesto de haber encontrado más tempranamente posiciones rechazadas, necesitando menos exploración.

El análisis visual anterior, se demuestra cuantitativamente en la Tabla 4 y en la Tabla 2, en donde se muestran los datos de los porcentajes de eficacia observada al usar la condición **a** y **d** respectivamente. Haciendo una comparación cuantitativa entre los resultados de ambas condiciones, se aprecia que al usar la condición **a** para cada porcentaje de eficacia 80 %, 90 % y 95 %, el modelo requiere alrededor del doble de análisis de pixeles (PA) de IM, por lo que necesita, en similar proporción, realizar cálculos de $fitness$ (Fit) al ir encontrando posiciones rechazadas (RP), lo que se traduce en un mayor *tiempo de ejecución* (t).

Observar que entre los mismos porcentajes de eficacia observada anteriores al 100 % entre las Tablas 4 y 2, la medida $\%\Gamma$ presenta diferentes valores, debido a que se capturan los datos al final de cada *ciclo* de ejecución, en donde el *ciclo* que se asigna, es el primero que alcance ($\geq$) el porcentaje de eficacia deseado. Sin embargo, esto no presenta diferencias en el análisis de resultados.

Datos capturados de eficiencia				
Datos	80 %	90 %	95 %	100 %
$ciclo$	81	95	103	111
$\%\Gamma(ciclo)$	4.44756	4.99840	5.27153	5.52559
$\%RP(ciclo)$	53.7473	72.1038	84.6165	100
$\%PA(ciclo)$	55.8044	73.4982	85.4275	100
$\%Fit(ciclo)$	62.6831	80.6664	92.6720	107.268
$\%Fit_N(ciclo)$	23.1500	18.9892	16.7795	14.5752
$\%Fit_E(ciclo)$	76.8500	81.0108	83.2205	85.4248
$t(ciclo)$	1.601s	1.939s	2.165s	3.168s

Tabla 4: datos de eficiencia de cada porcentaje de eficacia observada para condición **a**.

Para un análisis más robusto de *tiempo de ejecución*, en las Tablas 5 y 6, se presentan los datos de *ciclo* y *tiempo de ejecución* (t) considerando el total de ejecuciones realizadas para la condición **a** y **d** respectivamente. Ambas tablas están constituidas de 8 columnas. Las columnas 2 a 8 están en función de la primera columna. Las columnas 2 (*ciclo*) y 6 (t) representan los datos de la ejecución escogida. Las columnas 3, 4, 5, 7 y 8 representan los datos del total de ejecuciones realizados. Cada columna representa, en orden de izquierda a derecha: el porcentaje de eficacia observada, la cantidad de *ciclos* ejecutados, la *moda* estadística de los *ciclos*, el rango *mínimo* y *máximo* de *ciclos*, el *ciclo* promedio, el *tiempo de ejecución* t, el rango de tiempo *mínimo* y *máximo* y el tiempo promedio. Con estas tablas, se demuestra que el modelo al usar la condición **d**, para cada porcentaje de eficacia, es más eficiente que usando la condición **a**, puesto que usando la condición **d**, hasta un 95 % de eficacia, se tiene que como promedio y rango mínimo y máximo del total de ejecuciones, el modelo tiene un *tiempo de ejecución* menor de un 30 % aproximadamente, necesitando entre un 15 % y 22 % aproximadamente de menos *ciclos*. Y para un 100 % de eficacia, el modelo tiene un t menor de un 15 % aproximadamente, necesitando de un 5 % aproximadamente menos hasta la misma cantidad de *ciclos*. Esta disminución anterior, es debido a que para detectar el 5 % de eficacia final, el modelo necesita analizar mayor cantidad de pixeles que al usar la condición **a**, en donde la mayoría son posiciones rechazadas, en consecuencia de haber encontrado la mayoría de los bordes más tempranamente. Para llegar a $SN = 0$, el modelo solo sigue encontrando posiciones rechazadas hasta que se revisa todo IM y todas las fuentes creadas están agotadas, pero como ya se tiene el 100 % de eficacia, esta medida no es relevante. Luego, se demuestra y reafirma, que el modelo al usar la condición **a** requiere de mayor cantidad de *ciclos* y de un mayor *tiempo de ejecución* que al usar la condición **d**.

Datos de *ciclo* y *tiempo de ejecución* de condición **a**							
$\%eficacia$	*ciclo*	*moda*	$[min, max]$	$\overline{ciclo}$	t	$[t_{min},\ t_{max}]$	$\bar{t}$
80	81	81	[81, 81]	81	1.601	[1.537, 1.682]	1.5996
90	95	95	[94, 96]	95	1.939	[1.869, 2.112]	1.9416
95	103	103	[102, 104]	103	2.165	[2.089, 2.361]	2.1758
100	111	111	[111, 111]	111	3.168	[2.908, 3.489]	3.1216
$SN = 0$	115	115	[115, 116]	115.3	3.252	[2.982, 3.565]	3.1999

Tabla 5: datos extendidos de *ciclo* y *tiempo de ejecución* de cada porcentaje de eficacia observada para condición **a**.

Datos de *ciclo* y *tiempo de ejecución* de condición **d**							
$\%eficacia$	*ciclo*	*moda*	$[min, max]$	$\overline{ciclo}$	t	$[t_{min},\ t_{max}]$	$\bar{t}$
80	63	62	[63, 65]	63.6	1.052	[1.037, 1.194]	1.1047
90	78	78	[77, 78]	77.5	1.335	[1.297, 1.439]	1.3729
95	86	86	[86, 87]	86.1	1.511	[1.502, 1.633]	1.5674
100	107	107	[105, 111]	107.1	2.497	[2.346, 2.972]	2.6036
$SN = 0$	114	114	[114, 115]	114.1	3.419	[3.248, 3.419]	3.3316

Tabla 6: datos extendidos de *ciclo* y *tiempo de ejecución* de cada porcentaje de eficacia observada para condición **d**.

Los resultados obtenidos y analizados anteriormente de la segunda parte de este experimento, son solo debido a la diferencia entre el modelo usando la condición **a** y **d**. Usando la condición **d**, se hace reemplazo de una fuente f_k siempre que se detecte una solución vecina n_{f_k} que sea borde, sin importar si esta tiene mejor $fitness$ que f_k. En cambio, usando la condición **a**, si el n_{f_k} detectado es borde, pero no tiene mejor $fitness$ que f_k, n_{f_k} no reemplazará a f_k. Luego, la posición de n_{f_k} no se crea como fuente si es que no existe aún, en donde este cálculo de $fitness$ de la posición no se guarda y se debe realizar nuevamente si es que se llega a analizar la posición de n_{f_k} en un proceso futuro.

Considerando todos los resultados y análisis anteriores de este experimento, se concluye que el modelo ABC-ED básico es más eficiente, independientemente del valor de sus parámetros, usando solo fuentes de comida selectas y la condición de selección greedy **d** para reemplazo de una fuente. Lo anterior es debido a que es una mejor opción siempre reemplazar una fuente actual cuando se detecta una posición vecina que sea borde, que esperar a encontrar una posición vecina con mejor $fitness$ que la fuente actual. Esta eliminación de la espera, al usar la condición **d**, hace que el modelo tenga una convergencia más eficiente.

Traduciendo los resultados a la naturaleza, cada abeja de miel u obrera empleada, la cual le esté extrayendo el néctar a la flor que esté empleada, si encuentra una flor vecina con néctar, se va a emplear a esta última, ya que para la colonia va a ser mejor, debido a que se van a ir encontrando más flores y obteniendo una mayor cantidad de néctar recolectado en menor tiempo, que solo cambiarse de la flor actual a una flor vecina cuando esta última tenga un mayor néctar que la flor actual.

4.2.3. Experimento 3

Hasta ahora, en experimentos 1 y 2, se ha usado un valor en los parámetros de control (MCN, $limit$ y ε) estándar, por ser un punto de partida de análisis, los cuales están ubicados en la Tabla 1a. $MCN = 200$, ciclos máximos para el modelo, donde el mecanismo interno de término ha detenido antes la ejecución; $limit = 8$, revisando por cada fuente toda su vecindad y $\varepsilon = 50$, por ser el equilibrio de exploración entre localizar nuevas fuentes y usar fuentes ya creadas inactivas, pero aún no agotadas.

Luego, como MCN se puede obtener por el mecanismo interno de término del modelo, se debe analizar qué sucede con la eficiencia versus la eficacia del modelo manteniendo $limit$, pero cambiando ε, y analizar manteniendo ε, pero variando $limit$.

Para lo anterior, se hizo un análisis de extremos y medio en los valores de los parámetros $limit \in [0, 8]$ y $\varepsilon \in [0, 100]$, manteniendo igual el resto de los parámetros. Se realizaron 10 ejecuciones por cada conjunto de parámetros. Por cada $limit$, se generan 3 conjuntos de parámetros variando ε, y viceversa. Así, se genera un total de 9 conjuntos de parámetros, realizando un total de 90 ejecuciones. Lo que se varía en cada conjunto de parámetros es:

- Para $limit = 8$: $\varepsilon = 0$, $\varepsilon = 50$ y $\varepsilon = 100$.
- Para $limit = 4$: $\varepsilon = 0$, $\varepsilon = 50$ y $\varepsilon = 100$.

- Para $limit = 0$: $\varepsilon = 0$, $\varepsilon = 50$ y $\varepsilon = 100$.

La eficiencia del modelo, para llegar a la misma eficacia, se puede evaluar principalmente con tres medidas: la cantidad de cálculos de $fitness$ (Fit) o pixeles analizados (PA) necesarios de IM, el *tiempo de ejecución* (t) y la cantidad de *ciclos* (MCN) necesarios de ejecución.

En la Tabla 7b se presentan, en forma resumida, los resultados obtenidos como promedio de cada porcentaje de eficacia observada y $SN = 0$, considerando el total de ejecuciones para cada conjunto de parámetros, de las tres medidas de eficiencia principales del modelo. Los resultados para $\{limit = 8, \varepsilon = 50\}$, corresponden a los usados en experimentos 1 y 2. En la Tabla 7a, se presenta el porcentaje de fuentes creadas o bordes detectados para cada porcentaje de eficacia observada del modelo.

Analizando los resultados de la Tabla 7b, se observa lo siguiente:

- A mayor ε, mayor Fit y menor t y MCN.
- A menor $limit$, mayor Fit y menor t y MCN.
- A menor t, menor MCN.

Luego, el valor de los parámetros de control $limit$ y ε para obtener el menor cálculo de $fitness$ (Fit) necesario de IM es $\{limit = 8, \varepsilon = 0\}$. Para obtener el menor *tiempo de ejecución* (t) y MCN es $\{limit = 0, \varepsilon = 100\}$. Y para obtener un equilibrio entre estas medidas es $\{limit = 4, \varepsilon = 50\}$. El 80 % de eficacia ha mostrado ser suficiente para tener la estructura de bordes representativa que el modelo detecta de la imagen, en donde la relación con valores aproximados entre estos tres conjuntos para este porcentaje de eficacia es:

- Con $\{limit = 8, \varepsilon = 0\}$: se requiere de 0.5 veces menos Fit y 1.6 veces más t y MCN que con $\{limit = 4, \varepsilon = 50\}$.
- Con $\{limit = 0, \varepsilon = 100\}$: se requiere de 2.8 veces más Fit, 0.5 veces menos t y 4 veces menos MCN que con $\{limit = 4, \varepsilon = 50\}$.
- Con $\{limit = 0, \varepsilon = 100\}$: se requiere de 5.5 veces más Fit, 3.3 veces menos t y 6.3 veces menos MCN que con $\{limit = 8, \varepsilon = 0\}$.

Observar que para un 80 % de eficacia y $\{limit = 8, \varepsilon = 0\}$, se tiene un 4.4 % de fuentes creadas o bordes detectados de un total de 5.5 %, necesitando solo un 13.1 % de análisis de todo IM, y de forma equilibrada con $\{limit = 4, \varepsilon = 50\}$ se requiere el doble (26.1 %) de análisis de IM pero con un 60 % menos de *tiempo de ejecución* y *ciclos*. Además, tampoco se necesita analizar todo IM (72.3 %) para obtener el menor *tiempo de ejecución* posible con $\{limit = 0, \varepsilon = 100\}$.

Por lo tanto, se debe tratar de reducir los cálculos de $fitness$ (Fit) de IM si es posible, junto con disminuir su *tiempo de ejecución* (t) realizando una localización por vecindad más eficiente; lo que queda propuesto como trabajo futuro.

Dato %	**80 %**	**90 %**	**95 %**	**100 %**	SN=0
$\%\Gamma$	4.444656	4.986457	5.257836	5.525590	5.525590

(a) Tabla con porcentaje de fuentes creadas o bordes detectados para cada porcentaje de eficacia observada y mecanismo interno de término de modelo.

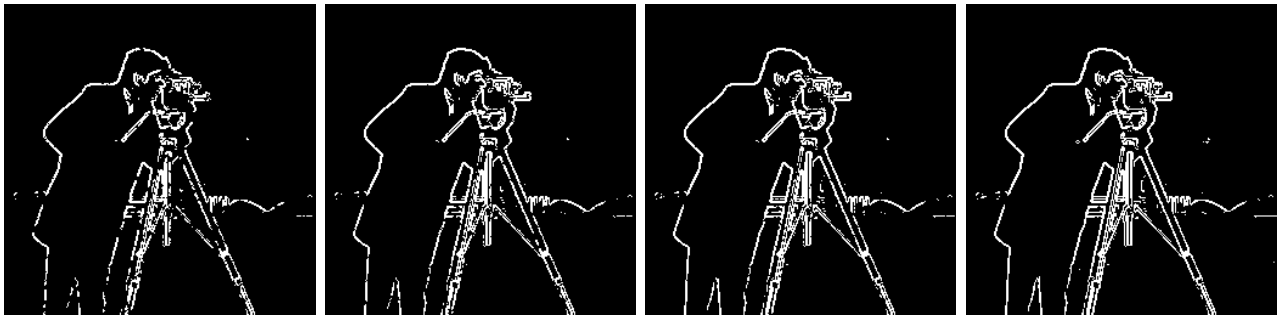
		ε														
		$\varepsilon = 0$					$\varepsilon = 50$					$\varepsilon = 100$				
$limit$	$\overline{x}$	**80 %**	**90 %**	**95 %**	**100 %**	SN=0	**80 %**	**90 %**	**95 %**	**100 %**	SN=0	**80 %**	**90 %**	**95 %**	**100 %**	SN=0
8	$\overline{\%Fit}$	13.1	23.4	38.4	100	100	21.6	33.1	47.4	99.8	100	23.6	36.8	51.9	100	100
	$\overline{t}$	1.566	1.827	1.979	2.970	3.021	1.105	1.373	1.567	2.604	3.332	1.028	1.295	1.470	2.621	3.453
	$\overline{MCN}$	88	100	106	111	115	64	78	86	108	114	59	72	80	89	114
4	$\overline{\%Fit}$	16.7	32.6	54.0	100	100	26.1	42.4	60.3	99.9	100	30.2	47.8	66.4	100	100
	$\overline{t}$	1.360	1.528	1.655	2.554	2.582	0.965	1.215	1.400	2.347	2.946	0.873	1.086	1.241	2.203	2.951
	$\overline{MCN}$	80	88	92	95	99	55	68	75	96	102	50	60	66	72	102
0	$\overline{\%Fit}$	72.3	85.9	92.7	100	100	72.3	85.9	92.7	100	100	72.3	85.9	92.7	100	100
	$\overline{t}$	0.509	0.623	0.706	1.475	1.505	0.500	0.614	0.696	1.442	1.471	0.474	0.587	0.669	1.424	1.453
	$\overline{MCN}$	14	16	17	19	20	14	16	17	19	20	14	16	17	19	20

(b) Tabla con datos promedio de $\%Fit$ o $\%PA$, t y MCN de experimento 3, comparando parámetros de control $limit$ y ε para cada porcentaje de eficacia observada y mecanismo interno de término del modelo.

Tabla 7: tablas finales de experimentación del modelo ABC-ED básico.

4.2.4. Resultados de las otras imágenes de entrada

A continuación, considerando la experimentación final, se muestran los resultados de las dos imágenes (Figuras 8b y 8c) de entrada restantes. Para cada imagen de entrada, se presenta la imagen de salida del modelo para cada porcentaje de eficacia observada usando $\{limit = 4, \varepsilon = 50\}$. Por medio de tablas, se muestran los datos de eficiencia principales para $\{limit = 8, \varepsilon = 0\}$, $\{limit = 4, \varepsilon = 50\}$ y $\{limit = 0, \varepsilon = 100\}$. Se obtiene la misma conclusión entre estos conjuntos que con la Figura 8a. A menor Fit, se requiere de mayor t y viceversa.

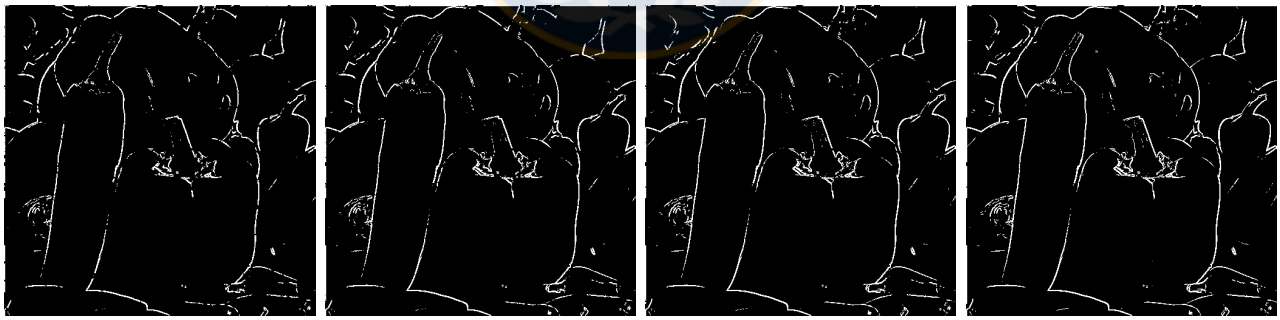


(a) 80 %, $ciclo = 30$. (b) 90 %, $ciclo = 39$. (c) 95 %, $ciclo = 44$. (d) 100 %, $ciclo = 57$.

Figura 17: imágenes de salida de la Figura 8b, con $m =$ Sobel y $\mu = 300$.

	$limit = 8$, $\varepsilon = 0$				$limit = 4$, $\varepsilon = 50$				$limit = 0$, $\varepsilon = 100$			
Datos	**80 %**	**90 %**	**95 %**	**100 %**	**80 %**	**90 %**	**95 %**	**100 %**	**80 %**	**90 %**	**95 %**	**100 %**
ciclo	44	59	63	67	30	39	44	57	8	9	10	11
$\%\Gamma(ciclo)$	5.37	6.06	6.41	6.71	5.38	6.11	6.43	6.71	5.60	6.06	6.49	6.71
$\%Fit(ciclo)$	15.9	17.8	43.5	100	27.6	43.7	57.7	99.2	71.9	82.0	93.1	100
$t(ciclo)$	0.259	0.355	0.400	0.592	0.212	0.269	0.310	0.447	0.105	0.130	0.162	0.307

Tabla 8: datos de eficiencia principales resultantes de la Figura 8b.



(a) 80 %, $ciclo = 43$. (b) 90 %, $ciclo = 53$. (c) 95 %, $ciclo = 59$. (d) 100 %, $ciclo = 74$.

Figura 18: imágenes de salida de la Figura 8c, con $m =$ Sobel y $\mu = 200$.

	$limit = 8$, $\varepsilon = 0$				$limit = 4$, $\varepsilon = 50$				$limit = 0$, $\varepsilon = 100$			
Datos	**80 %**	**90 %**	**95 %**	**100 %**	**80 %**	**90 %**	**95 %**	**100 %**	**80 %**	**90 %**	**95 %**	**100 %**
ciclo	66	76	81	86	43	53	59	74	10	12	13	14
$\%\Gamma(ciclo)$	3.52	3.95	4.18	4.38	3.54	3.98	4.19	4.38	3.54	4.01	4.23	4.38
$\%Fit(ciclo)$	11.9	17.4	36.1	100	24.1	38.4	55.8	99.6	67.5	83.7	92.9	100
$t(ciclo)$	1.174	1.379	1.515	2.281	0.787	0.989	1.156	1.756	0.378	0.480	0.565	1.026

Tabla 9: datos de eficiencia principales resultantes de la Figura 8c.

5. Discusión y Conclusiones

Se ha adaptado exitosamente el algoritmo ABC al problema de localización eficiente en detección de bordes en imágenes digitales en escala de grises, al no necesitar analizar todos los pixeles de una imagen para obtener su estructura de bordes representativa, realizando una integración para generar la detección de bordes, entre un método clásico para la identificación y el algoritmo ABC para la localización, obteniendo así el modelo ABC-ED básico desarrollado en este trabajo, del cual se implementó un prototipo usado para evaluar experimentalmente el modelo. Por ende, se solucionó exitosamente el problema propuesto, cumpliendo el objetivo general y específicos al usar correctamente la metodología definida para este trabajo.

Aunque el algoritmo o modelo ABC-ED básico haya sido adaptado del algoritmo ABC, estos tienen diferencias entre sí, como: la cantidad de soluciones finales a buscar, el concepto de qué se considera fuente de comida, la vecindad de una fuente, nuevas características introducidas, la selección greedy para reemplazo de una fuente y es más acorde al comportamiento de las abejas de miel. A continuación, se discuten estas diferencias.

El algoritmo ABC fue diseñado originalmente para resolver problemas de optimización para funciones multidimensionales y multimodales, en donde converge para obtener una sola solución óptima. En cambio, ABC-ED básico está diseñado para obtener múltiples soluciones al problema, que son los bordes de la imagen de entrada.

En el algoritmo ABC se considera como fuente de comida cualquier posición generada del dominio del problema, ya sea por vecindad o exploración. En cambio, en ABC-ED básico solo se considera como fuente, borde y solución, la posición de un pixel de la imagen que cumpla con un requisito, el cual es tener un valor de $fitness$ mayor que el valor del parámetro umbral μ. Si el requisito se cumple, la posición se crea como fuente (si es que aún no ha sido creada) denominada fuente de comida selecta. El resto de las posiciones se consideran como posiciones rechazadas y de no interés para análisis.

Al tener diferentes dominios de búsqueda, se necesita de una diferente forma de localizar una solución candidata por vecindad. Para ABC-ED básico se usa como máximo la vecindad de Moore de ocho vecinos alrededor del pixel central, la cual es la fuente de comida.

En ABC-ED básico se introdujeron nuevas características que el algoritmo ABC podría incluir, estas son: el **manejo de colisión de soluciones obtenidas** en el proceso de ejecución, lo cual ABC no realiza, aunque se tiene en cuenta que su ambiente a explorar es de dominio $\mathbb{R}$, el cual es mucho más amplio que el espacio restringido por las dimensiones de una imagen, sin embargo, a medida que se va convergiendo a la solución final, es más probable que se vayan obteniendo posibles soluciones por vecindad o exploración ya obtenidas en el pasado, debiendo calcular el $fitness$ y analizarlas nuevamente en la fase que corresponda, en cambio, ABC-ED básico hace un manejo eficiente de colisión de soluciones, no permitiendo la creación de fuentes con igual posición y calculando solo una vez el $fitness$ por cada posición localizada de la imagen de entrada; el **nuevo parámetro de control** ε que controla la localización por exploración entre nuevas fuentes no analizadas y seguir analizando fuentes abandonadas, pero no agotadas; el **mecanismo interno de término de ejecución** basado en el agotamiento de todas las fuentes

creadas, lo que hace al modelo no depender totalmente del parámetro de control de cantidad máxima de ciclos (MCN) para su detención, lo cual es muy útil para la experimentación, al entregar una cota superior del valor de ciertos parámetros y medidas de evaluación de interés, e.g.: MCN y el *tiempo de ejecución* máximo; la **elección estocástica de fuente de comida más eficiente** en la Fase de Obreras Espectadoras, la cual es la fase más importante para la convergencia del algoritmo al simular el reclutamiento para una fuente. En esta fase, el ABC debe realizar t_1 intentos (con $t_1 \geq SN$) para elegir SN fuentes, y por cada fuente, obtener una solución candidata vecina, la cual posiblemente haya sido generada anteriormente, debido a que no maneja la colisión de soluciones. En cada intento del ABC, se realiza una condición que se debe cumplir para poder elegir una fuente estocásticamente. Esta condición es que la probabilidad de la fuente debe ser mayor a un número aleatorio generado. Considerando que desde la ejecución inicial del ABC, hasta antes de la convergencia hacia la solución final, en donde las probabilidades de las fuentes están más distribuidas y por ende son menores, implica que es menos probable que se cumpla la condición de elección de fuente, necesitando muchos más intentos (t_1) que SN y recorriendo varias veces todas las fuentes que se explotan para poder elegir las SN fuentes estocásticamente. En cambio, ABC-ED básico está diseñado para buscar múltiples soluciones, por lo que siempre se tiene en consideración que las probabilidades de las fuentes están más distribuidas y son menores. Por esta razón, se construye una ruleta de selección para elegir estocásticamente una fuente, en donde cada fuente posee un rango de esta ruleta. Así, cada vez que se genere un número aleatorio r en la Fase de Obreras Espectadoras, siempre se va a poder elegir una fuente con rango de ruleta $[a, b]$, de tal forma que $r \in [a, b]$. El problema es que la fuente escogida puede estar agotada, debido a que aún no se llega a la Fase de Obrera Exploradora. Solo por esta razón anterior, se necesita $t_2 \geq SN$ intentos para elegir las SN fuentes estocásticamente. Sin embargo, $t_2 < t_1$, por la justificación anterior. Luego, basta mejorar el modelo para no elegir fuentes agotadas en la Fase de Obreras Espectadoras, garantizando así realizar $t_2 = SN$ intentos para elegir SN fuentes estocásticamente en esta fase.

El algoritmo ABC usa una condición de selección greedy para reemplazo de una fuente mediante la comparación de $fitness$ entre la fuente actual y una vecina candidata, en donde se reemplaza la fuente actual por la candidata, si esta última tiene un mayor $fitness$. En cambio, ABC-ED básico reemplaza una fuente actual por una vecina candidata, siempre que esta última sea un borde, i.e., tiene un $fitness$ mayor que el umbral μ, sin importar si la candidata tenga menor o mayor $fitness$ que la fuente actual.

Debido a que son múltiples soluciones las que se buscan y el concepto usado de qué se considera como fuente, son razones que hacen al modelo ABC-ED básico más acorde al comportamiento de las abejas de miel. Para el modelo, una solución al problema o borde o fuente de comida se puede considerar como una flor en la naturaleza, la cual posee de néctar. Luego, ABC-ED básico detecta las flores dentro de un ambiente, simulando las abejas de miel en su recolección de néctar de las flores en su ambiente para la colonia.

Por las diferencias mencionadas anteriormente entre ABC y ABC-ED básico, hacen al modelo desarrollado más eficiente en términos de manejo de recursos, convergencia y *tiempo de ejecución*.

Para la creación del modelo ABC-ED básico, se establecieron definiciones necesarias para explicar mediante argumentación y pseudo-algoritmos su funcionamiento, siendo el núcleo de su prototipo implementando, del cual se describe su diseño y ambiente de trabajo usado.

Se realizó una experimentación al modelo, con el fin de justificar las decisiones tomadas en su construcción estructural y algorítmica, junto con la calibración de los parámetros de control. La experimentación consiste de dos partes: la definición de un plan de prueba y la aplicación de este. En el plan de prueba se definieron: las métricas de evaluación de eficiencia y eficacia para el modelo; las imágenes de entrada a utilizar; dos gráficos obtenidos automáticamente del prototipo, en donde se puede visualizar el comportamiento del modelo en toda su ejecución, junto con los porcentajes de eficacia observada 80 %, 90 %, 95 % y 100 %; la declaración de tres experimentos junto con sus propósitos respectivos, los cuales son definidos y realizados al aplicar el plan de prueba.

Hay principalmente tres medidas para medir la eficiencia del modelo con respecto a su eficacia: la cantidad de *análisis de la imagen* de entrada, que es la cantidad de cálculos de $fitness$ (Fit); el *tiempo de ejecución* (t) y el número máximo de ciclos (MCN) de ejecución.

De experimento 1, se demuestra que el modelo: es completo, independientemente del operador clásico de identificación que se utilice al poder llegar al 100 % de eficacia; es muy estable entre sus ejecuciones para un mismo conjunto de valores de parámetros, al tener una muy baja diferencia entre sus resultados y medidas de evaluación; requiere de mayor localización por exploración en una detección de bordes más discontinua, debido a que mediante una localización por vecindad no se puede llegar a soluciones aisladas de la estructura de bordes representativa de la imagen, la cual se obtiene en temprana ejecución del modelo, debido a que este realiza un análisis global de la imagen en cada ciclo, y no secuencial y en orden como una detección realizada por un método clásico.

De experimento 2, se demuestran dos mejoras importantes para la eficiencia del modelo: el uso de solo fuentes selectas y que la mejor condición de selección greedy para reemplazo de una fuente sea por cualquier posición/fuente vecina que sea selecta, lo que es una mejor opción que esperar por una posición con mejor $fitness$ que la fuente actual. Esta eliminación de la espera y el uso de solo fuentes selectas, hace al modelo converger más eficientemente.

De experimento 3, se hizo un estudio de calibración de los parámetros de control *limit* y ε para la eficiencia del modelo, realizando un análisis de valores extremos de estos parámetros. A mayor *limit* posible y menor ε posible, se obtiene la más baja cantidad de *análisis de la imagen*; a menor *limit* posible y mayor ε posible, se obtiene el más bajo *tiempo de ejecución* y número máximo de ciclos posible. Lo anterior implica que, para obtener la estructura de bordes representativa de una imagen, hay un *intercambio* entre la cantidad de *análisis de la imagen* y el *tiempo de ejecución*. A menor cantidad de *análisis de la imagen*, se requiere de un mayor *tiempo de ejecución* y viceversa, puesto que en la localización por exploración, al explorar aleatoriamente los pixeles, se realiza una mayor cantidad de *análisis de la imagen* que en la localización por vecindad, ya que en la vecindad de un borde es probable que hayan más bordes, debido a que generalmente forman lineas continuas. Además, en la localización por vecindad se requiere de un mayor *tiempo de ejecución* que en la localización por exploración, debido al diseño del modelo,

en donde se gasta *tiempo de ejecución* al analizar por cada fuente todos sus vecinos. Luego, el modelo es más eficiente a mayor localización por exploración, debido a que requiere de un menor *tiempo de ejecución* en detectar bordes que una localización principalmente por vecindad.

La experimentación realizada al modelo, permitió hacer un estudio robusto del comportamiento de eficiencia del modelo. Logrando importantes mejoramientos de eficiencia y realizando un análisis de como el modelo puede tener su mejor desempeño.

Para el modelo ABC-ED básico, implementar un operador de identificación de bordes no es difícil, debido al diseño de implementación creado, en donde se consideró una alta cohesión y bajo acoplamiento para sus módulos.

Luego, y por lo tanto, se propone como **trabajo futuro** dos aspectos generales: mejorar la eficiencia del modelo y adaptarlo para el procedimiento de detección de bordes de Canny, el cual es considerado el mejor detector de bordes en la actualidad; lo que incluye usar una técnica de adelgazamiento de bordes. Para una mejor eficiencia, se debe mejorar el modelo en no tener la opción de poder elegir fuentes agotadas en la Fase de Obreras Espectadoras, usar estructuras de datos más eficientes que permitan bajar el orden de complejidad, y una localización por vecindad con un menor *tiempo de ejecución* y si es posible, de menor *análisis de la imagen*.

Referencias

[1] Djemel Ziou, Salvatore Tabbone, et al. Edge detection techniques-an overview. *Pattern Recognition and Image Analysis C/C of Raspoznavaniye Obrazov I Analiz Izobrazhenii*, 8:537–559, 1998.

[2] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report, Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, 2005.

[3] Marjan Mernik, Shih-Hsi Liu, Dervis Karaboga, and Matej Črepinšek. On clarifying misconceptions when comparing variants of the artificial bee colony algorithm by offering a new implementation. *Information Sciences*, 291:115–127, 2015.

[4] Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39(3):459–471, 2007.

[5] Bahriye Basturk and Dervis Karaboga. An artificial bee colony (abc) algorithm for numeric function optimization. In *IEEE swarm intelligence symposium*, volume 8, pages 687–697, 2006.

[6] Dervis Karaboga and Bahriye Basturk. On the performance of artificial bee colony (abc) algorithm. *Applied soft computing*, 8(1):687–697, 2008.

[7] Dervis Karaboga and Bahriye Basturk. Artificial bee colony (abc) optimization algorithm for solving constrained optimization problems. In *Foundations of Fuzzy Logic and Soft Computing*, pages 789–798. Springer, 2007.

[8] Dervis Karaboga and Bahriye Akay. A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, 214(1):108–132, 2009.

[9] Bahriye Akay and Dervis Karaboga. A survey on the applications of artificial bee colony in signal, image, and video processing. *Signal, Image and Video Processing*, 9(4):967–990, 2015.

[10] Raman Maini and Himanshu Aggarwal. Study and comparison of various image edge detection techniques. *International journal of image processing (IJIP)*, 3(1):1–11, 2009.

[11] Mamta Juneja and Parvinder Singh Sandhu. Performance evaluation of edge detection techniques for images in spatial domain. *methodology*, 1(5):614–621, 2009.

[12] Roberts L. G. Machine perception of three dimensional solids. Optical and Electro-Optical Information Processing, J, T, Tippet, Ed., Cambridge, Mass., MIT Press, 1965.

[13] Irwin Sobel and Gary Feldman. A 3x3 isotropic gradient operator for image processing. Presented at a talk at the Stanford Artificial Project, 1968.

[14] Irwin Sobel. History and definition of the sobel operator. *Retrieved from the World Wide Web*, 2014.

[15] Judith MS Prewitt. Object enhancement and extraction. *Picture processing and Psychopictorics*, 10(1):15–19, 1970.

[16] Vincent Torre and Tomaso A Poggio. On edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):147–163, 1986.

[17] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[18] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

[19] Valery Tereshko. Reaction-diffusion model of a honeybee colony's foraging behaviour. In *Parallel Problem Solving from Nature PPSN VI*, pages 807–816. Springer, 2000.

[20] Valery Tereshko and Troy Lee. How information-mapping patterns determine foraging behaviour of a honey bee colony. *Open Systems & Information Dynamics*, 9(02):181–193, 2002.

[21] Valery Tereshko and Andreas Loengarov. Collective decision making in honey-bee foraging dynamics. *Computing and Information Systems*, 9(3):1, 2005.

[22] Thomas D Seeley. *The wisdom of the hive: the social physiology of honey bee colonies*. Harvard University Press, 2009.

[23] Dervis Karaboga and Beyza Gorkemli. A combinatorial artificial bee colony algorithm for traveling salesman problem. In *Innovations in Intelligent Systems and Applications (INISTA), 2011 International Symposium on*, pages 50–53. IEEE, 2011.

[24] Dervis Karaboga, Bahriye Akay, and Celal Ozturk. Artificial bee colony (abc) optimization algorithm for training feed-forward neural networks. In *Modeling decisions for artificial intelligence*, pages 318–329. Springer, 2007.

[25] Dervis Karaboga and Bahriye Akay. Artificial bee colony (abc) algorithm on training artificial neural networks. In *2007 IEEE 15th Signal Processing and Communications Applications*. 2007.

[26] Alok Singh. An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem. *Applied Soft Computing*, 9(2):625–631, 2009.

[27] Dervis Karaboga, Beyza Gorkemli, Celal Ozturk, and Nurhan Karaboga. A comprehensive survey: artificial bee colony (abc) algorithm and applications. *Artificial Intelligence Review*, 42(1):21–57, 2014.

[28] Tirimula Rao Benala, Sree Durga Jampala, Sathya Harish Villa, and Bhargavi Konathala. A novel approach to image edge enhancement using artificial bee colony optimization algorithm for hybridized smoothening filters. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 1071–1076. IEEE, 2009.

[29] Selami Parmaksızoğlu and Mustafa Alçı. A novel cloning template designing method by using an artificial bee colony algorithm for edge detection of cnn based imaging sensors. *Sensors*, 11(5):5337–5359, 2011.

[30] Yimin Deng and Haibin Duan. Biological edge detection for ucav via improved artificial bee colony and visual attention. *Aircraft Engineering and Aerospace Technology: An International Journal*, 86(2):138–146, 2014.

[31] E Yigitbasi and N Baykan. Edge detection using artificial bee colony algorithm (abc). *Int. J. Inf. Electron. Eng*, 3(6):634–638, 2013.

A. Algoritmos secundarios del modelo ABC-ED básico

Algoritmo 12 Pseudo-algoritmo ABC-ED básico. Funcion: obtener-vecino-candidato.

```
1:  function OBTENER-VECINO-CANDIDATO(FuenteComida f_k)
2:      FuenteComida f_n;
3:      Vecindad N_{f_k};
4:      Vecino n_{f_k};
5:      id_vecino ← 0; n_fitness ← 0;
6:      N_{f_k} ← OBTENER-VECINDAD(f_k);
7:      if HAY-VECINOS-POR-ESCOGER(N_{f_k}) then
8:          Do
9:              id_vecino ← rand(1, CANTIDAD-VECINOS(N_{f_k}));  // [1, Λ_{f_k}]
10:             n_{f_k} ← OBTENER-VECINO(N_{f_k}, id_vecino);
11:         while YA-ESTA-ESCOGIDO(n_{f_k});
12:         ESTABLECER-COMO-ESCOGIDO(n_{f_k});
13:         f_n ← OBTENER-FUENTECOMIDA-CON-POSICION(n_{f_k}.i, n_{f_k}.j);
14:         if EXISTE(f_n) then  // f_n ≠ null ∧ f_n ∈ (AFS ∨ IFS ∨ EFS).
15:             if f_n ∈ IFS then
16:                 n_{f_k}.fitness ← fit(f_n);
17:                 n_{f_k}.fs ← f_n;
18:             else n_{f_k}.fitness ← 0;
19:             end if
20:         else if EXISTE-POSICION-COMO-RECHAZADA(n_{f_k}.i, n_{f_k}.j) == false then
21:             // f_n = null ∧ posicion de n_{f_k} ∉ RP.
22:             n_fitness ← CALCULAR-FITNESS(n_{f_k}.i, n_{f_k}.j);
23:             if n_fitness ≤ μ then
24:                 AGREGAR-COMO-POSICION-RECHAZADA(n_{f_k}.i, n_{f_k}.j);
25:                 // posicion de n_{f_k} ∈ RP.
26:                 n_fitness ← 0;
27:             end if
28:             n_{f_k}.fitness ← n_fitness;
29:         else n_{f_k}.fitness ← 0;  // f_n = null ∧ posicion de n_{f_k} ∈ RP.
30:         end if
31:     end if
32:     return n_{f_k};
33: end function
```

Algoritmo 13 Pseudo-algoritmo ABC-ED básico.
Función: hay-mas-posibles-fuentes-para-crear.

```
1: function HAY-MAS-POSIBLES-FUENTES-PARA-CREAR( )
2:     if cantidad_fuentes_creadas + tamaño(RP) < (r × c) then return true;
3:     else return false;
4:     end if
5: end function
```

Algoritmo 14 Pseudo-algoritmo ABC-ED básico.
Función: obtener-nueva-unica-fuentecomida-selecta.

```
function OBTENER-NUEVA-UNICA-FUENTECOMIDA-SELECTA( )
    FuenteComida f_s;
    rand_i ← 0; rand_j ← 0;
    intentos_maximos ← SN;
    fs_fitness ← 0;
    for intento ← 0 to intentos_maximos do
        rand_i ← rand(0, r − 1);
        rand_j ← rand(0, c − 1);
        if EXISTE-POSICION-COMO-FUENTECOMIDA(rand_i, rand_j) == false ∧
            EXISTE-POSICION-COMO-RECHAZADA(rand_i, rand_j) == false then
            fs_fitness ← CALCULAR-FITNESS(rand_i, rand_j);
            if fs_fitness > μ then
                f_s ← CREAR-NUEVA-FUENTECOMIDA(rand_i, rand_j); // ahora f_s ∈ NFS.
                fit(f_s) ← fs_fitness;
                return f_s;
            else AGREGAR-COMO-POSICION-RECHAZADA(rand_i, rand_j); // en RP.
            end if
        end if
    end for
    return null;
end function
```

Algoritmo 15 Pseudo-algoritmo ABC-ED básico. Función: Calcular-Fitness

```
function CALCULAR-FITNESS(i, j)
    return OBTENER-MAGNITUD-GRADIENTE(IM, m, i, j); // usando exp. (1) para m.
end function
```

Algoritmo 16 Pseudo-algoritmo ABC-ED básico. Función: obtener-nueva-unica-fuentecomida.

```
function OBTENER-NUEVA-UNICA-FUENTECOMIDA( )
    FuenteComida f_s;
    rand_i ← 0; rand_j ← 0;
    intentos_maximos ← SN;
    for intento ← 0 to intentos_maximos do
        rand_i ← rand(0, r − 1);  rand_j ← rand(0, c − 1);
        if EXISTE-POSICION-COMO-FUENTECOMIDA(rand_i, rand_j) == false then
            f_s ← CREAR-NUEVA-FUENTECOMIDA(rand_i, rand_j); // ahora f_s ∈ NFS.
            fit(f_s) ← CALCULAR-FITNESS(f_s.i, f_s.j);
            return f_s;
        end if
    end for
    return null;
end function
```

Algoritmo 17 Pseudo-algoritmo ABC-ED básico. Función: obtener-forma-de-reemplazo.

1: **function** OBTENER-FORMA-DE-REEMPLAZO( )
2: $forma \leftarrow 0$; $r \leftarrow 0$;
3: **if** HAY-MAS-POSIBLES-FUENTES-PARA-CREAR( ) $\wedge$ $IFS \neq \varnothing$ **then**
4: $r \leftarrow rand(1, 100)$;
5: **if** $\varepsilon \geq r$ **then** $forma \leftarrow Nueva\_Exploracion$;
6: **else** $forma \leftarrow Fuentes\_Inactivas$;
7: **end if**
8: **else if** HAY-MAS-POSIBLES-FUENTES-PARA-CREAR( ) $\wedge$ $IFS == \varnothing$ **then**
9: $forma \leftarrow Nueva\_Exploracion$;
10: **else if** HAY-MAS-POSIBLES-FUENTES-PARA-CREAR( ) == **false** $\wedge$ $IFS \neq \varnothing$ **then**
11: $forma \leftarrow Fuentes\_Inactivas$;
12: **else** $forma \leftarrow No\_Mas\_Fuentes\_de\_Reemplazo$;
13: **end if**
14: **end function**

B. Diseño de implementación del modelo

Se describe a continuación, de forma simple y breve el diseño de implementación del modelo ABC-ED básico para la generación de su prototipo.

Con el fin de cumplir un diseño *modulado*, con *alta cohesión* y bajo *acoplamiento* para trabajos futuros, se realizó la siguiente implementación:

Clases implementadas: myMatlabInterface, myDataHandler, ClassicEdgeDetection, ABC_EdgeDetection, FoodSource, Neighbourhood, Neighbour y Statistic.

Propósitos y/o funciones de cada clase:

- **myMatlabInterface:** realiza la comunicación con MATLAB Engine para la entrada y salida de datos entre C++ y MATLAB. Además de poder realizar comandos de MATLAB directamente desde C++ para ser ejecutados en MATLAB Engine.

- **myDataHandler:** convierte los datos recibidos por la clase myMatlabInterface en estructura $mxArray$ hacia la estructura de datos necesitada y viceversa. Lo anterior es válido para la imagen de entrada (IM) con clases ClassicEdgeDetection y ABC_EdgeDetection, la matriz binarizada de salida por método clásico (OM_{C-ED}) dada por clase ClassicEdgeDetection, la matriz binarizada de salida por ABC-ED básico (OM_{ABC-ED}) dada por clase ABC_EdgeDetection y los arreglos con valores estadisticos de la clase Statistic.

- **ClassicEdgeDetection:** tiene implementado los operadores clásicos de detección de bordes: Roberts, Sobel y Prewitt para la generación de la OM_{C-ED} respectiva al método. Se llama a dos funciones de esta clase desde la clase ABC_EdgeDetection: obtener-magnitud-gradiente (en Algoritmo 15) para el cálculo de $fitness$ de una fuente y la función para obtener el tipo de posición de un pixel en IM al crear cada fuente de comida en la función crear-fuente-comida de clase ABC_EdgeDetection.

- **ABC_EdgeDetection:** realiza la detección de bordes con el algoritmo ABC adaptado. Posee todo lo descrito sobre el modelo ABC-ED básico construido.

- **FoodSource:** para la construcción de cada fuente de comida. Incluye los atributos necesarios para la fuente y la referencia a su vecindad, la cual se crea usando la clase Neighbourhood, luego de obtener el tipo de posición de la fuente en IM, con el fin de saber cuántos y cuáles vecinos puede tener su vecindad.

- **Neighbourhood:** para la construcción de la vecindad de cada fuente de comida. Posee todos los vecinos posibles para una fuente y controla la información de la vecindad mediante métodos. Al crear la vecindad, se crean todos los vecinos posibles usando clase Neighbour en función del tipo de posición de la fuente en IM.

- **Neighbour:** clase padre de todas las subclases del tipo de vecino posible para una fuente. Se usa polimorfismo, e.g.: Left_Neighbour es una subclase de la clase Neighbour que representa el vecino izquierdo en la vecindad de una fuente. Cada subclase hereda los atributos y los métodos de la clase Neighbour. Solo en las subclases se redefine (*virtual/override*) el método para establecer la posición del vecino en función de la posición de la fuente central. Este método es llamado desde la clase padre Neighbour para cada subclase de vecino correspondiente. Por lo anterior, no se necesita realizar cálculos de posición cada vez que se tiene bajo análisis un vecino de una fuente durante la ejecución del modelo por medio de la clase ABC_EdgeDetection.

- **Statistic:** se integra a la clase ABC_EdgeDetection para capturar en ejecución los datos de eficiencia definidos en sección 4.1. Al final de la ejecución del modelo, se pide a la clase myDataHandler convertir cada uno de estos datos en la estructura $mxArray$, y luego son enviados a MATLAB mediante la clase myMatlabInterface. Luego, en la clase Statistic se construyen tres comandos MATLAB. Dos comandos para graficar los dos tipos de gráficos (de porcentaje y de número) y el tercer comando para guardar en un archivo .mat todas las variables con sus datos ubicados en MATLAB Workspace de la ejecución del modelo, para respaldo y propósitos futuros. Estos tres comandos MATLAB son enviados a la clase myMatlabInterface para ser ejecutados en MATLAB Engine.

Estructuras de datos utilizadas: con el fin de realizar reemplazos de fuentes a bajo costo, se usó una lista doblemente encadenada para cada conjunto definido (AFS, IFS y EFS) en sección 3.1. Con el fin de tener un bajo costo y constante, para agregar y obtener posiciones rechazadas (RP) y saber si una posición de IM existe como fuente, se usó arreglos bidimensionales estáticos. Para lo anterior, considerando un bajo uso de recursos, se usó una matriz booleana para manejar RP, y para las posiciones de IM existentes como fuente se usó una estructura (*struct*) como cada elemento de una matriz. Esta estructura (*struct*) solo posee una referencia de tipo FoodSource que es *null* por defecto, la cual se establece al momento de crear una fuente para la posición respectiva. Se usa memoria dinámica para crear cada fuente. Además, se usó un arreglo bidimensional de: enteros (*int*) para IM, dobles (*double*) para almacenar la magnitud del gradiente calculado al usar un método clásico para la clase ClassicEdgeDetection y booleano (*bool*) para las matrices binarizadas de salida OM_{C-ED} y OM_{ABC-ED}.

C. Manual de Usuario

Se describe a continuación, el manual de usuario para el prototipo implementado del modelo ABC-ED básico.

Requerimientos para ejecución de prototipo: Windows 7 x64, Matlab R2013a x64 o superior, 2GB RAM mínimo.

Directorio de trabajo por defecto: "C:/abc_ed", el cual debe estar creado antes de iniciar la ejecución del prototipo. En este directorio, debe estar la imagen de entrada a usar.

1. Hacer doble click a archivo "ABC_ED.exe"para abrir como terminal de windows el prototipo, o ir al directorio donde se encuentra el ejecutable por terminal de windows y escribir el nombre del archivo y presione enter.

2. Se iniciará MATLAB Engine automáticamente. Si aparece "error", verifique que cumpla con los requerimientos para la ejecución o intente nuevamente. Si aparece "ok", se abrió la ventana externa "MATLAB Command Window" y podrá continuar con la ejecución.

3. Ingrese nombre completo de la imagen de entrada. Por ejemplo: "nombre.png". Se abrirá una ventana externa "v1" de MATLAB mostrando la imagen en escala de grises al centro.

4. Elegir operador de método clásico (parámetro m) para la identificación en la detección de bordes. Ingresar: Roberts (r) o Sobel (s) o Prewit (p). Sobel es por defecto. Presione enter.

5. Ingresar el valor del parámetro umbral μ y presione enter. Se hará la detección de bordes con el método clásico m elegido, usando como umbral μ. Al terminar, se mostrara la imagen de salida en "v1" a la izquierda de la imagen de entrada.

6. Se pregunta si desea calibrar el valor de μ. Si (y) o no (n). Ingrese opción y presione enter. Si su opción fue "y", vaya al paso 5. Si su opción fue "n", se guardará en directorio de trabajo por defecto la imagen de salida dada por el método clásico.

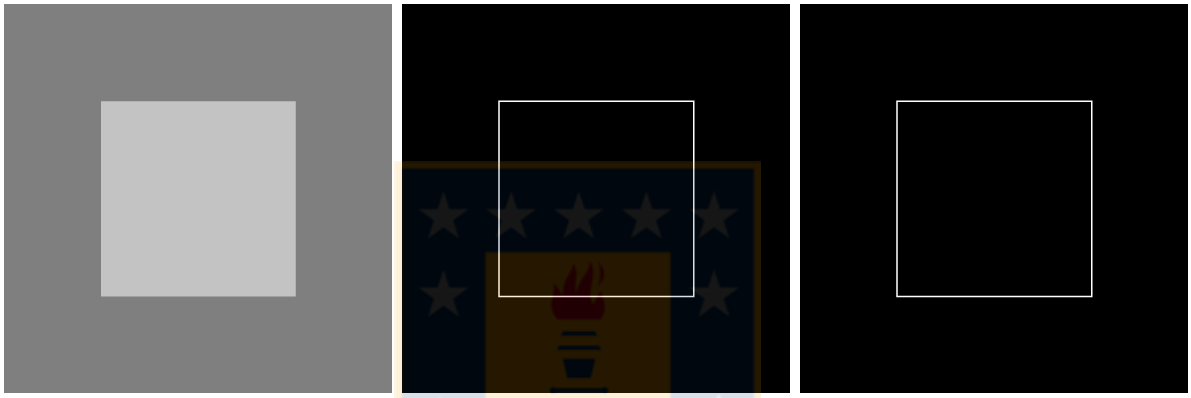
7. Se pedirán los parámetros para ejecutar ABC-ED básico. Ingrese el valor para: MCN, $limit$ y ε presionando enter para cada uno. Luego, se pregunta si desea visualizar en "v1", a la derecha de la imagen de entrada, cada imagen de salida por cada ciclo de ejecución del modelo. Si (y) o no (n). Ingrese opción y presione enter.

8. Se realiza la ejecución del modelo. Al terminar la ejecución: se muestra en "v1" la imagen de salida del modelo a la derecha de la imagen de entrada; se muestra en la terminal los resultados finales; se guarda en el directorio de trabajo por defecto la imagen de salida final del modelo, ambos gráficos de visualización de comportamiento del modelo, el archivo MATLAB .mat con todas las variables y sus datos de la ejecución del prototipo, y si su opción de visualización de imagen de salida de cada ciclo del modelo fue sí (y), se guardaran las imágenes de salida para cada porcentaje de eficacia observada, si estas son alcanzadas.

9. Insertar comando válido para prototipo. Inserte "help" para ver los comandos (help, go y quit). Si inserta "go", vaya al paso 3. Si inserta "quit", la ejecución del prototipo terminará.

D. Terreno de Verdad

Se describe a continuación, la precisión de identificación de bordes del modelo ABC-ED básico, usando terreno de verdad. Se usa una imagen construida de dimensión 512x512 (Figura 19a), con un cuadrado interno de mayor intensidad de dimensión 256x256, ubicado uniformemente al centro de la imagen. Observar que la precisión de identificación del modelo ABC-ED básico está dada por el operador de método clásico para cálculo de $fitness$, por lo que la imagen de salida usando el método clásico Sobel (Figura 19b, con $\mu = 90$) es idéntica a la salida del modelo ABC-ED básico (Figura 19c, con $m =$ Sobel, $\mu = 90$). Los pixeles detectados como borde forman un marco con el contorno del cuadrado interior de la figura original. Este marco tiene un grosor de 2 pixeles alrededor de todo el contorno, por lo que se detecta el borde del cuadrado interior de color más intenso, y el borde del color menos intenso.



(a) Imagen original. (b) Salida método Sobel. (c) Salida ABC-ED básico.

Figura 19: imágenes de terreno de verdad

$$Precision = \frac{Aciertos}{Aciertos + FP + FN} \times 100 \tag{14}$$

Se usa la expresión (14) para evaluar la precisión de identificación, en donde:

- $Aciertos$: el número de pixeles de bordes detectados correctamente.
- FP (Falsos positivos): el número de pixeles detectados como bordes, pero no lo son.
- FN (Falsos negativos): el número de pixeles de borde no detectados, pero si son bordes.

Luego, $Aciertos = 2056 = 1024$ (el marco de contorno interior detectado) $+$ 1032 (el marco de contorno exterior detectado); $FP = 0$, no hay pixeles detectados que no sean bordes; $FN = 0$, todos los pixeles que son borden fueron detectados. Implica que $Precision = 100\,\%$. Observar que este análisis es válido para $\mu = 90$.