UNIVERSIDAD DE CONCEPCIÓN
Facultad de Ingeniería
Departamento de Ingeniería
Informática y Ciencias de la Computación

Profesor Patrocinante:
María Angélica Pinninghoff J.

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

JAIME IGNACIO VÁSQUEZ FEIJOO

Tesis
para optar al titulo de

Magíster en Ciencias de la Computación

Diciembre 2016
Concepción, Chile.

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 tradicional, como Canny, 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 la de Canny, y una localización o búsqueda eficiente, como el algoritmo ABC, pueden ser integradas para lograr una eficiente detección de bordes, con el fin de que no sea necesario analizar todos los pixeles de una imagen para obtener su estructura de bordes representativa. Esta integración, se logró en la creación del modelo ABC-ED 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. Así, ABC-ED detecta con una búsqueda eficiente las flores dentro de un ambiente, simulando a las abejas en su recolección de néctar para la colonia.

Para la creación del modelo ABC-ED, 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 usando su prototipo, resultando que el modelo es más preciso en su detección de bordes a menor cantidad de *análisis de pixeles* que realice; puesto que tiene un mejor desempeño en las imágenes con menos regiones de objetos y más focalizadas que en imágenes más complejas con gran cantidad de contornos distribuidos, sin embargo, el modelo necesita un promedio de 25.35 % de *análisis de pixeles* para detectar el 95 % de lo que detecta Canny, el cual requiere de analizar siempre toda la imagen, teniendo una diferencia promedio entre las salidas del modelo y Canny de 0.22 %. Logrando así una buena aproximación visual con una leve diferencia a la salida de Canny, en donde ya se presenta la estructura de bordes representativa de la imagen de entrada.

Tabla de contenido

Resumen I

Índice de figuras III

Índice de algoritmos IV

Índice de tablas IV

Glosario V

1. Introducción 1
- 1.1. Descripción del Problema . . . 2
- 1.2. Hipótesis . . . 2
- 1.3. Objetivos . . . 2
  - 1.3.1. Objetivo General . . . 2
  - 1.3.2. Objetivos Específicos . . . 2
- 1.4. Metodología . . . 3
- 1.5. Estructura de Informe . . . 4

2. Marco Teórico 5
- 2.1. Detección de Borde en Imágenes . . . 5
  - 2.1.1. Metodos de Gradiente . . . 5
  - 2.1.2. Metodos basados en Laplaciano . . . 7
  - 2.1.3. Detector de Bordes de Canny . . . 8
- 2.2. Inteligencia de Enjambre . . . 9
- 2.3. Algoritmo ABC . . . 10
  - 2.3.1. Comportamiento de las abejas de miel . . . 11
  - 2.3.2. Algoritmo de Colonia de Abejas Artificiales (ABC) . . . 12
  - 2.3.3. Historia del Algoritmo ABC . . . 15
  - 2.3.4. Trabajos detectando bordes en imágenes usando ABC . . . 16

3. Desarrollo 18
- 3.1. Definiciones . . . 18
- 3.2. Modelo ABC-ED . . . 19
- 3.3. Implementación . . . 26

4. Experimentación 27
- 4.1. Plan de Prueba . . . 27
- 4.2. Aplicación de Plan de Prueba . . . 30
  - 4.2.1. Experimento . . . 30

5. Discusión y Conclusiones 41

Bibliografía 43

Anexos **46**
A. Algoritmos secundarios del modelo ABC-ED 46
B. Diseño de implementación del modelo 48
C. Graficos y Tablas secundarias de Experimentación 50
D. Manual de Usuario 54

Índice de figuras

1. Máscaras de Robert. Fuente: [1]. 5
2. Máscaras de Sobel. Fuente: [2]. 6
3. Máscaras de Prewitt. Fuente: [3]. 6
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. Fuente: Elaboración propia. 7
5. Máscara de Laplaciano. Fuente: [4]. 7
6. Máscara de Laplaciano de Gauss (LoG). Fuente: [4]. 8
7. Serie de imágenes procedas por Canny. Fuente: Elaboración propia. 10
8. Vecindad de Moore de $f_k(i, j)$. Fuente: Elaboración propia. 18
9. Imágenes de dataset BSDS500. Original, segmentación (SEG) y terreno de verdad (GT). Fuente: [5]. 29
10. Análisis de dataset BSDS500. Fuente: Elaboración propia. 30
11. Umbrales cálculados de los métodos de umbralización. Fuente: Elaboración propia. . 32
12. Análisis de comportamiento y métricas agrupando los TM. Fuente: Elaboración propia. 33
13. Análisis promedio de *cálculos de fitness* para cada TM. Fuente: Elaboración propia. 34
14. Análisis promedio de % HD Canny y % HD GT por TM. Fuente: Elaboración propia. 35
15. Análisis promedio de C GT por TM. Fuente: Elaboración propia. 35
16. Análisis promedio de ciclos por TM. Fuente: Elaboración propia. 35
17. Salidas con el mejor % Fit acorde a su Rank de la Figura **??**. Fuente: Elaboración propia. 36
18. Imágenes de salida pertenecientes a {% Fit Group = 100}. Fuente: Elaboración propia. 39
19. Muestra de salidas de ABC-ED. En orden: la imagen de entrada I, la salida del modelo al obtener el 95 % y 100 % de eficacia y el terreno de verdad de I. Fuente: Elaboración propia. 40
20. Análisis de comportamiento de ABC-ED para TM Mean y Median. Fuente: Elaboración propia. 50
21. Análisis de comportamiento de ABC-ED para TM Matlab y Otsu. Fuente: Elaboración propia. 51
22. Análisis de métricas de ABC-ED para TM Mean y Median. Fuente: Elaboración propia. 52
23. Análisis de métricas de ABC-ED para TM Matlab y Otsu. Fuente: Elaboración propia. 53

Índice de Algoritmos

1. Pseudo-algoritmo ABC. Los pasos generales. Fuente: [6]. 12
2. Pseudo-algoritmo ABC detallado. Parte 1: Inicialización. Fuente: [6]. 13
3. Pseudo-algoritmo ABC detallado. Parte 2: Fase Obreras Empleadas. Fuente: [6]. . . 14
4. Pseudo-algoritmo ABC detallado.
Parte 3: Calcular Probabilidades y Fase Obreras Espectadoras. Fuente: [6]. 14
5. Pseudo-algoritmo ABC detallado. Parte 4: Fase Obreras Exploradoras. Fuente: [6]. . 15
6. Pseudo-algoritmo ABC-ED. Los pasos generales. Fuente: Elaboración propia. 20
7. Pseudo-algoritmo ABC-ED. Función: Inicialización. Fuente: Elaboración propia. . . 21
8. Pseudo-algoritmo ABC-ED. Función: Fase Obreras Empleadas. Fuente: Elaboración propia. 21
9. Pseudo-algoritmo ABC-ED. Función: Calcular Probabilidades. Fuente: Elaboración propia. 22
10. Pseudo-algoritmo ABC-ED. Función: Fase Obreras Espectadoras. Fuente: Elaboración propia. 23
11. Pseudo-algoritmo ABC-ED. Función: Fase Obrera Exploradora. Fuente: Elaboración propia. 24
12. Pseudo-algoritmo de experimentación de ABC-ED. Fuente: Elaboración propia. . . . 31
13. Pseudo-algoritmo ABC-ED. Funcion: obtener-vecino-candidato. Fuente: Elaboración propia. 46
14. Pseudo-algoritmo ABC-ED. Función: hay-mas-posibles-fuentes-para-crear. Fuente: Elaboración propia. 46
15. Pseudo-algoritmo ABC-ED.
Función: obtener-nueva-unica-fuentecomida-selecta. Fuente: Elaboración propia. . . 47
16. Pseudo-algoritmo ABC-ED. Función: Calcular-Fitness. Fuente: Elaboración propia. 47
17. Pseudo-algoritmo ABC-ED. Función: obtener-forma-de-reemplazo. Fuente: Elaboración propia. 47

Índice de tablas

1. Cálculo de umbral para cada método de umbralización. Fuente: Elaboración propia. 28
2. Parámetros usados en experimentación para el modelo. Fuente: Elaboración propia. 31
3. Las 10 salidas con 100 % de eficacia con mejor % Fit. Fuente: Elaboración propia. . 36
4. Las 10 salidas con 100 % de eficacia con el mejor % HD GT. Fuente: Elaboración propia. 37
5. Las 10 salidas con 100 % de eficacia con el mejor C GT. Fuente: Elaboración propia. 37
6. Análisis de todas las salidas con 100 % de eficacia agrupadas por el % Fit correspondiente. Fuente: Elaboración propia. 38

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 [7]. 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, 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 inspiración 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 más cercano 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 [6]. 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 [8] 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 [9–13]. 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 [14].

Por lo anterior, el algoritmo ABC, es 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 [15].

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 [16], por lo que es crucial que la detección de bordes sea eficaz y eficiente. Un método eficaz de detección de bordes, como Canny, tiene la desventaja de tener un alto costo computacional, siendo poco eficiente al realizar cálculos para todos los pixeles de una imagen, puesto que la detección de bordes con un método tradicional se realiza mediante una localización e identificación en todos los pixeles de la imagen; lo que hace difícil poder usar estos métodos 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 localización o búsqueda en la detección de bordes de Canny, con el fin de que no sea necesario analizar todos los pixeles de una imagen para obtener su estructura de bordes representativa.**

1.2. Hipótesis

Es posible desarrollar un modelo que integre el detector de bordes de Canny y el algoritmo ABC, con el fin de lograr una detección de bordes eficiente y eficaz en imágenes digitales en escala de grises, sin la necesidad de analizar todos los pixeles de una imagen para obtener su estructura de bordes representativa.

Esta hipótesis está fundamentada en los resultados obtenidos en el trabajo de memoria de título desarrollado [17].

1.3. Objetivos

1.3.1. Objetivo General

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

1.3.2. Objetivos Específicos

1. Revisar el estado del arte de métodos tradicionales 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 el detector de bordes de Canny 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.

1.4. Metodología

Para llevar a cabo está tesis, 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 tradicionales principales.
     b) Algoritmo ABC, desde su inicios, qué es, 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 el detector de bordes de Canny para la identificación, con el algoritmo ABC para la localización.
  2. Implementación del modelo desarrollado, resultando en un prototipo. Para lo anterior:
     a) Definir herramientas necesarias para implementación y ambiente de trabajo.
     b) Implementar el detector de bordes de Canny.
     c) Implementar modelo desarrollado mencionado en punto 1.
- Evaluación del Modelo:
  1. Diseñar plan de prueba para el modelo:
     a) Declarar y definir métricas de evaluación para el modelo.
     b) Declarar experimentación con su propósito.
  2. Aplicar plan de prueba diseñado. Para experimentación declarada:
     a) Definirla y llevarlo a cabo.
     b) Presentar resultados obtenidos.
     c) Realizar análisis de resultados obtenidos.
- Discusión y Conclusiones:
  1. Síntesis general del trabajo de tesis.
  2. Síntesis del modelo desarrollado.
  3. Trabajo a futuro.
- Elaboración informe de tesis: durante todo el transcurso de la tesis.

1.5. Estructura de Informe

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

2. Marco Teórico

En este capítulo se describe el estado del arte de los métodos tradicionales 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 [16].

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 [1], Sobel [2, 18] y Prewitt [3]. 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. 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. La máscara "x" es la máscara "y" rotada 90^o.

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

Figura 1: Máscaras de Robert. Fuente: [1].

- **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". 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 [18]. La máscara "x" es la máscara "y" rotada 90^o.

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

Figura 2: Máscaras de Sobel. Fuente: [2].

- **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". La máscara "x" es la máscara "y" rotada 90^o.

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

Figura 3: Máscaras de Prewitt. Fuente: [3].

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 I 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 usando 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.

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

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

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

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. Fuente: Elaboración propia.

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).

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

Figura 5: Máscara de Laplaciano. Fuente: [4].

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

El Laplaciano de Gauss (LoG) [4] 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). Fuente: [4].

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

2.1.3. Detector de Bordes de Canny

El detector de bordes de Canny [19] fue desarrollado con la intención de mejorar los detectores de bordes ya desarrollados en ese tiempo. Pese a su antigüedad, es considerado un detector de bordes estándar. 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 la siguiente serie de pasos:

1. **Suavizado:** usando un filtro de suavizado gaussiano se quita el ruido de la imagen. La máscara del filtro guassiano con una desviación estandar de $\sigma = 1.4$ es presentada en expresión (7). El efecto de suavizado con este filtro para la imagen de entrada Lenna (Figura 7a) es mostrado en Figura 7b.

$$\frac{1}{159} * \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \tag{7}$$

2. **Cálculo de gradiente y dirección:** se calcula la magnitud del gradiente y su dirección por cada pixel de la imagen. Generalmente es usado el operador Sobel descrito en sección 2.1.1. El efecto de cálculo de gradiente para la imagen suavizada es mostrado en Figura 7c.
3. **Adelgazamiento de bordes:** se aplica la técnica no-máxima supresión (non-maximum suppression o NMS) para adelgazar los bordes dados por la salida de gradiente magnitud. Para lo anterior, por cada pixel de la imagen:
   - Se redondea su dirección de gradiente a uno de los cuatro posibles ángulos: 0^o (horizontal), 45^o (diagonal derecho), 90^o (vertical) o 135^o (diagonal izquierdo). Los cuales corresponden a una dirección posible dentro de la vecindad de Moore de 8 vecinos alrededor de un pixel central.

- Se compara su magnitud con la de los pixeles vecinos en su dirección de gradiente positiva y negativa. Por ejemplo, si su dirección de gradiente es vertical, se hace la comparación con el pixel de arriba y abajo. Si la magnitud del pixel central es mayor que la de ambos pixeles vecinos, la magnitud se conserva. En caso contrario, se suprime.

El efecto de aplicar NMS para la salida de gradiente magnitud es mostrado en Figura 7d.

4. **Doble umbralización:** Bordes potenciales son determinados por umbralización usando dos umbrales, el umbral mínimo μ_{min} y el umbral máximo μ_{max} de la siguiente forma:

   - Si $G(x, y) > \mu_{max} : I(x, y)$ se marca como borde *fuerte*.
   - Si $\mu_{min} \leq G(x, y) \leq \mu_{max} : I(x, y)$ se marca como borde *débil*.
   - Si $G(x, y) < \mu_{min} : I(x, y)$ se suprime.

   El efecto de aplicar doble umbralización para la salida con bordes adelgazados usando como $\mu_{min} = 46$ y $\mu_{max} = 116$ es mostrado en Figura 7e, en donde los bordes *fuertes* son de color blanco y los bordes *débiles* son de color gris.

5. **Seguimiento de bordes por histéresis:** bordes finales son determinados suprimiendo todos los bordes que no estén conectados a un borde *fuerte*, ya sea directamente o indirectamente. Para lo anterior, se usa un algoritmo recursivo o iterativo que permita hacer un seguimiento para detectar todos los bordes *débiles* conectados a algún borde *fuerte*. El efecto de aplicar histéresis a la salida con doble umbralización, generando la imagen de salida final de Canny, es mostrado en Figura 7f.

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..." [20].

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.

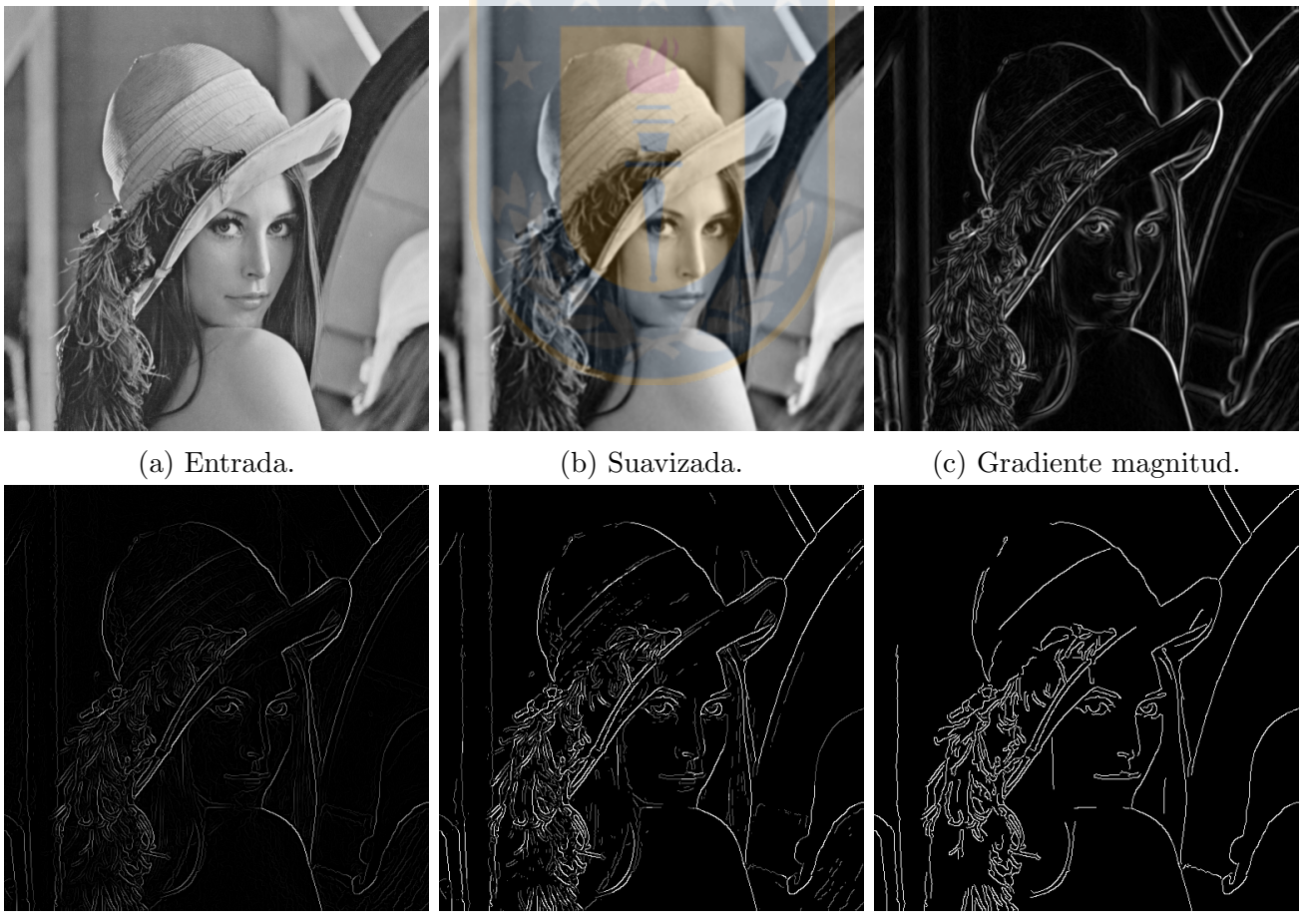
c) **Fluctuaciones:** la aleatoriedad es a menudo crucial para estructuras emergentes, ya que permite el descubrimiento de nuevas soluciones.

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 [21].

2.3. Algoritmo ABC

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



(a) Entrada. (b) Suavizada. (c) Gradiente magnitud.

(d) No-máxima supresión. (e) Doble umbralización. (f) Histéresis.

Figura 7: Serie de imágenes procedas por Canny. Fuente: Elaboración propia.

2.3.1. Comportamiento de las abejas de miel

Tereshko considera una colonia de abejas de miel como un sistema dinámico recolectando información desde un ambiente y ajustando su comportamiento acorde a éste [21].

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 [21–23]. 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 [24], 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 % [24].

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 [23].

Por simplicidad, el abandono de una fuente de comida es igualmente probable para todas las fuentes de comida [23].

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 [6].

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 generales del algoritmo ABC están descritos en Algoritmo 1.

Algoritmo 1 Pseudo-algoritmo ABC. Los pasos generales. Fuente: [6].

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

```
1:  procedure ABC
2:      Inicialización;
3:      ciclo ← 0;
4:      while ciclo < MCN do
5:          Fase Obreras Empleadas;
6:          Calcular Probabilidades;
7:          Fase Obreras Espectadoras;
8:          Fase Obrera Exploradora;
9:          Memorizar la fuente de comida con mayor fitness;
10:         ciclo ← ciclo + 1;
11:     end while
12: 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 $f[x(s)]$ la función de evaluación del problema a optimizar; 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. Fuente: [6].

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) ← solución aleatoria;
        fit[x(s)] ← f[x(s)];
        pruebas[x(s)] ← 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 (8) (Algoritmo 3).

$$v(s)_j = x(s)_j + \phi[x(s)_j - x(k)_j] \tag{8}$$

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 $x(s)$. La posición de la fuente $x(s)$ es establecida mediante 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 $v(s)_j$ mediante expresión (8).

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 (9), con el fin de que cada obrera espectadora pueda elegir una fuente mediante su $fitness$ (Algoritmo 4).

$$p[x(s)] = \frac{fit[x(s)]}{\sum_{s=1}^{SN} fit[x(s)]} \tag{9}$$

Algoritmo 3 Pseudo-algoritmo ABC detallado. Parte 2: Fase Obreras Empleadas. Fuente: [6].

```
11:  ciclo ← 0;
12:  while ciclo < MCN do
13:      // Fase Obreras Empleadas
14:      for s = 1 to SN do
15:          v(s) ← solución vecina de x(s) calculada por exp. 8;
16:          fit[v(s)] ← f[v(s)];
17:          if fit[v(s)] > fit[x(s)] then
18:              x(s) ← v(s);
19:              fit[x(s)] ← fit[v(s)];
20:              pruebas[x(s)] ← 0;
21:          else
22:              pruebas[x(s)] ← pruebas[x(s)] + 1;
23:          end if
24:      end for
```

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.

Algoritmo 4 Pseudo-algoritmo ABC detallado.
Parte 3: Calcular Probabilidades y Fase Obreras Espectadoras. Fuente: [6].

```
25:      // Calcular Probabilidades
26:      Se calcula la probabilidad p[x(s)] de cada fuente por exp. 9;
27:      // Fase Obreras Espectadoras
28:      s ← 1; t ← 0;
29:      while t < SN do
30:          r ← rand(0, 1); // r ∈ ℝ, un número aleatorio.
31:          if r < p[x(s)] then
32:              t ← t + 1;
33:              v(s) ← solución vecina de x(s) calculada por exp. 8;
34:              fit[v(s)] ← f[v(s)];
35:              if fit[v(s)] > fit[x(s)] then
36:                  x(s) ← v(s);
37:                  fit[x(s)] ← fit[v(s)];
38:                  pruebas[x(s)] ← 0;
39:              else pruebas[x(s)] ← pruebas[x(s)] + 1;
40:              end if
41:          end if
42:          s ← (s mod SN) + 1
43:      end while
```

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).

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. Fuente: [6].

```
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 [8], 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 [9], PSO y PS-EA [10] y DE, PSO y EA [11]. 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 [12]. 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 [13]. 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 [25], entrenar redes neuronales [26,27], para el problema de árbol de cobertura mínima limitado de hojas [28] y muchos otros problemas [29].

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 [14]. 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 [14] 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 [30–33].

Para una mejora en la calidad de los bordes de imágenes, en [30] 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 [31] 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 bor-

des. 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 [32] 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 [33], 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, se requiere mejorar el modelo; puesto que 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, que trata el problema de localización eficiente en detección de bordes en imágenes digitales en escala de grises, integrando el detector de bordes de Canny 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, en 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 al mismo tiempo. 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 8 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 8: Vecindad de Moore de $f_k(i, j)$. Fuente: Elaboración propia.

El cálculo de $fitness$ para un pixel, esta dado por la aplicación de un suavizado gaussiano usando la expresión (7). Filtro que se aplica al pixel central y su vecindad, para luego calcular el gradiente del pixel central usando el operador de Sobel para obtener la magnitud y valor de

$fitness$ usando expresión (1) y su dirección usando expresión (3).

Se definen cuatro conjuntos, que representan los estados posibles de una fuente. Estos conjuntos son disjuntos, 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 límite 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 $fitness < \mu_{min}$, por lo que una fuente no pertenece a RP.

El concepto usado de la cantidad de *cálculos de fitness* realizados y la cantidad de *análisis de pixeles* para una imagen es el mismo.

Una fuente de comida *selecta* es una fuente cuyo $fitness \geq \mu_{min}$. En ABC-ED se crean solo fuentes *selectas*, debido a que hace al modelo más eficiente en el manejo de recursos, cantidad de *análisis de pixeles* de la imagen y *tiempo de ejecución* [17].

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 [33].

El modelo ABC-ED está diseñado en dos fases. La primera fase es la integración de ABC con los dos primeros pasos de Canny para cálculo de $fitness$, aplicando el suavizado y cálculo de gradiente. La segunda fase es la aplicación de los siguientes pasos de Canny, los cuales son adegalzamiento de bordes, doble umbralización e histéresis. Sin embargo, la segunda fase no requiere recorrer todos los pixeles de la imagen; puesto que solo se analiza las fuentes creadas por el modelo.

Teniendo lo necesario definido, se describe el modelo ABC-ED (Artificial Bee Colony - Edge Detector o Colonia de Abejas Artificiales Detectoras de Borde).

3.2. Modelo ABC-ED

Los parámetros de entrada para ABC-ED son los del algoritmo ABC: SN, $limit$ y MCN. A los que se suman cuatro parámetros: I, μ_{min}, μ_{max} y ε. I es la imagen de entrada, μ_{min} es el valor de umbral mínimo para clasificar un pixel como fuente mediante su $fitness$, μ_{max} es el valor de umbral máximo para clasificar una fuente como borde *débil* y ε controla la localización por exploración entre buscar nuevas fuentes en forma aleatoria o elegir una fuente inactiva de IFS.

La salida de ABC-ED es una imagen binarizada dada por la aplicación de histéresis en la segunda fase del modelo.

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

Algoritmo 6 Pseudo-algoritmo ABC-ED. Los pasos generales. Fuente: Elaboración propia.

Input: Imagen y establecer los parámetros.
1: IM : imagen de entrada I.
2: μ_{min} : valor mínimo de umbral para clasificar un pixel como fuente.
3: μ_{max} : valor máximo de umbral para clasificar una fuente como borde débil.
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: OM: imagen binarizada dada por histéresis.
8: **procedure** ABC-ED( )
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: NO-MÁXIMA-SUPRESIÓN( );
19: DOBLE-UMBRALIZACIÓN( );
20: HISTÉRESIS( );
21: **end procedure**

En la **Inicialización**, se crea una población aleatoria de SN fuentes (Algoritmo 7). Para obtener una fuente, se usa la función obtener-nueva-unica-fuentecomida-selecta (Algoritmo 15, Anexo A), en donde para cada posición generada, si esta no existe ya como fuente y $\notin RP$, se calcula su $fitness$ (Algoritmo 16, Anexo A), para el cual si es mayor que μ_{min}, se crea la fuente 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 14, Anexo A), se debe terminar la ejecución de la primera fase del algoritmo, ya que no existen más posiciones en IM identificadas como a lo menos borde *débil*. Para terminar la fase, se puede hacer *e.g.*: $ciclo \leftarrow MCN$ y/o $SN \leftarrow 0$.

En la **Fase de Obreras Empleadas**, por cada fuente f_k en AFS (Algoritmo 8), se usa función obtener-vecino-candidato (Algoritmo 13, Anexo A) para obtener aleatoriamente un vecino n_{f_k} que no se haya escogido de la vecindad N_{f_k} usando la expresión (10) y se deja n_{f_k} como escogido. Luego, en esta función se actúa dependiendo de la posición de n_{f_k} como sigue:

$$n^{\nu}_{f_k} = N_{f_k}[rand(1, \Lambda_{f_k})] \tag{10}$$

- Si la posición de n_{f_k} existe como fuente 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:

Algoritmo 7 Pseudo-algoritmo ABC-ED. Función: Inicialización. Fuente: Elaboración propia.

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

- Si $f_n \in IFS$: al vecino n_{f_k} se asocia 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 y $\notin RP$, se calcula su $fitness$. Si este $fitness < \mu_{min}$, la posición se agrega a RP y se asigna a n_{f_k} $fitness$ 0. 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.

Algoritmo 8 Pseudo-algoritmo ABC-ED. Función: Fase Obreras Empleadas.
Fuente: Elaboración propia.

```
1: function FASE-OBRERAS-EMPLEADAS( )
2:     FuenteComida f_k ← AFS[0];
3:     FuenteComida f_n;
4:     Vecino n_{f_k};
5:     for each f_k ∈ AFS do
6:         n_{f_k} ← OBTENER-VECINO-CANDIDATO(f_k);
7:         if fit(n_{f_k}) ≥ μ_min then
8:             f_n ← n_{f_k}.fs;
9:             if EXISTE(f_n) == false then  // f_n es null
10:                f_n ← CREAR-NUEVA-FUENTECOMIDA(n_{f_k}.i, n_{f_k}.j, fit(n_{f_k}));  // fn ∈ NFS
11:            end if
12:            REEMPLAZAR-FUENTECOMIDA(f_k, f_n);  // f_k ← f_n
13:        else pruebas(f_k) ← pruebas(f_k) + 1;
14:        end if
15:    end for
16: end function
```

Luego, en la fase y con el vecino n_{f_k} obtenido, se realiza selección greedy entre $fit(n_{f_k})$ y μ_{min}. A diferencia del algoritmo ABC, en ABC-ED, se realiza selección greedy para reemplazo de una fuente entre el $fitness$ del vecino candidato y μ_{min}, y no por el $fitness$ entre el candidato y la fuente actual, debido a que hace el modelo más eficiente [17].

La selección greedy se realiza como sigue:

- Si $fit(n_{f_k}) \geq \mu_{min}$: se obtiene la fuente asociada de n_{f_k} y se le asigna a f_n. Si f_n no existe como fuente, la posición de n_{f_k} se crea como fuente, se calcula su $fitness$ y se le asigna a f_n. 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 cuál 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 $fit(n_{f_k}) < \mu_{min}$: se aumenta el contador de *pruebas* de f_k y se mantiene en AFS.

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 (9), para así construir una ruleta de selección AVL (Algoritmo 9), en base al árbol de búsqueda binario auto-balanceable AVL [34]. Luego, en la Fase de Obreras Espectadoras se puede elegir estocásticamente una fuente por medio de la ruleta de selección AVL eficientemente.

Algoritmo 9 Pseudo-algoritmo ABC-ED. Función: Calcular Probabilidades.
Fuente: Elaboración propia.

```
1: function CALCULAR-PROBABILIDADES( )
2:     FuenteComida f_k ← AFS[0];
3:     rango ← 0;
4:     suma_fit ← OBTENER-SUMA-TOTAL-DE-FITNESS-DE-FUENTES-EN-AFS( );
5:     for each f_k ∈ AFS do
6:         p(f_k) ← fit(f_k)/suma_fit;  // acorde a expresión. (9).
7:         RuletaAVL → AGREGAR(rango, rango + p(f_k), f_k);
8:         rango ← rango + p(f_k);
9:     end for
10: end function
```

En la **Fase de Obreras Espectadoras**, se eligen SN fuentes 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 aún 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 fuentes, 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 Obreras Empleadas ya descrita.

Observar que hay una diferencia entre el algoritmo ABC y ABC-ED 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 se requiere encontrar múltiples soluciones finales, por lo que las probabilidades de las fuentes están más distribuidas durante la mayoría de la ejecución. Lo anterior 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 el proceso de la fase.

Algoritmo 10 Pseudo-algoritmo ABC-ED. Función: Fase Obreras Espectadoras.
Fuente: Elaboración propia.

```
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-RULETA-AVL(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_min 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, fit(n_{f_k})); // fn ∈ NFS
18:                end if
19:                REEMPLAZAR-FUENTECOMIDA(f_k, f_n); // f_k ← f_n
20:            else pruebas(f_k) ← pruebas(f_k) + 1;
21:            end if
22:        end if
23:    end while
24: 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, debido a que no se podrán obtener más soluciones de ella. 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 el análisis de *limit* vecinos de f_k (Algoritmo 11).

Luego, si f_k está agotada, se obtiene su forma de reemplazo usando función obtener-forma-de-reemplazo (Algoritmo 17, Anexo A). Existen tres formas de reemplazo: *Nueva Exploración*, *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* (generación aleatoria); $\varepsilon = 0$ 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. Función: Fase Obrera Exploradora.
Fuente: Elaboración propia.

```
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 $f_r \in NFS$ 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 hubieran posibles posiciones para crear como fuentes, pero en la llamada a la función, se analizaron todas las posiciones faltan-

tes, 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 $f_r \in IFS$ ya creadas anteriormente, pero que 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 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 llega a que $SN = 0$, la primera fase del algoritmo termina, ya que no tiene más fuentes 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 último caso no sea ideal.

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

Ya terminada la primera fase del modelo, se tiene como salida todas las fuentes creadas durante el proceso en la lista $FSP2$, la cual se construye durante la primera fase, en donde al momento de crear cada fuente, esta se inserta en $FSP2$. Por ende, $FSP2 = AFS \cup IFS \cup EFS$.

Luego, se continúa con la segunda fase, i.e., no-máxima supresión para adelgazamiento de borde, doble umbralización e histéresis; pasos que están descritos en la sección 2.1.3. Sin embargo, estos tres pasos son realizados solo recorriendo $FSP2$ y no todos los pixeles de la imagen como lo hace Canny, por lo que ABC-ED es más eficiente.

En el adelgazamiento de borde, toda fuente tal que su $fitness$ no sea mayor que la de ambos vecinos en la dirección del gradiente se suprime, y por ende, es eliminada de $FSP2$.

En la doble umbralización, se clasifica cada fuente f_k que haya quedado de $FSP2$ en función de su $fitness$:

- Si $fit(f_k) > \mu_{max}$: la fuente es un borde *fuerte*.
- Si $\mu_{min} \leq fit(f_k) \leq \mu_{max}$: la fuente es un borde *débil*.
- Si $fit(f_k) < \mu_{min}$: la fuente se suprime y es eliminada de $FSP2$.

Para el seguimiento de bordes por histéresis, se diseñó un algoritmo iterativo que usa la fila *edges*. Este algoritmo consiste en dos etapas:

- La primera etapa es recorrer $FSP2$ en busca de fuentes que sean bordes *fuertes*. Si una fuente es borde *fuerte*, esta es agregada a *edges* y es marcada como borde *final* en la imagen de salida OM. Esta etapa se aprovecha de realizar al momento de hacer la doble umbralización.

- La segunda etapa consiste en recorrer *edges* hasta que esté vacía. Por cada fuente f_k (borde *fuerte*) en *edges*, se analiza su vecindad. Solo si un vecino es un borde *débil* y no está ya marcado en OM, el vecino se agrega a *edges* como borde *fuerte* y se marca en OM como borde *final*. Luego, f_k es eliminada de *edges*.

Con la histéresis terminada, se tiene la imagen binarizada OM, la cual es la salida del modelo ABC-ED.

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 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 el operador de Sobel, en ABC-ED se implementó este operador en el prototipo, el cual da resultados idénticos a los dados utilizando este operador directamente en MATLAB. Al igual que MATLAB, para el cálculo de gradiente de los pixeles que están en el 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.

[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`

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

4. Experimentación

En este capítulo se presenta el diseño y aplicación del plan de prueba para el modelo, en donde se establecen definiciones para la experimentación, se presentan sus resultados y se realiza un análisis de estos.

4.1. Plan de Prueba

Se definen las métricas de evaluación de eficacia y eficiencia para el modelo. Dado que el modelo integra el detector de bordes de Canny para realizar la identificación, ABC-ED solo va identificando, a lo más, todos los pixeles que Canny detecta. Por lo anterior, la eficacia para ABC-ED es lo detectado por Canny bajo el mismo ambiente (I, μ_{min} y μ_{max}).

Para evaluar la eficacia de ABC-ED, se usa la métrica Distancia de Hamming (HD) [35], adaptada a evaluar entre dos matrices binarizadas de igual tamaño. La métrica HD analiza todos los pixeles de igual posición entre ambas matrices, dando como salida la cantidad de pixeles que tienen valores diferentes. Luego, con $ciclo \in [0, MCN[$ y $\oplus$ el operador lógico XOR, se define:

- $OM^{ciclo}_{\text{ABC-ED}}$:= la matriz binarizada de salida del *ciclo* de ABC-ED.
- OM_{Canny} := la matriz binarizada de salida final de Canny.
- $HD(OM^{ciclo}_{\text{ABC-ED}}, OM_{\text{Canny}}) := OM^{ciclo}_{\text{ABC-ED}} \oplus OM_{\text{Canny}} = \sum_{i=0}^{r-1}(\sum_{j=0}^{c-1} OM^{ciclo}_{\text{ABC-ED}}[i][j] \oplus OM_{\text{Canny}}[i][j])$.
- $Edges_{\text{Canny}} := \sum_{i=0}^{r-1}(\sum_{j=0}^{c-1} OM_{\text{Canny}}[i][j])$, la cantidad total de bordes detectados por Canny.
- $eficacia^{ciclo}_{\text{ABC-ED}} := Edges_{\text{Canny}} - HD(OM^{ciclo}_{\text{ABC-ED}}, OM_{\text{Canny}})$, la cantidad de bordes iguales entre la salida del *ciclo* de ABC-ED y Canny.

Así, el porcentaje de eficacia por cada ciclo del modelo esta dado por expresión 11.

$$\%\, eficacia^{ciclo}_{\text{ABC-ED}} := \left(\frac{eficiacia^{ciclo}_{\text{ABC-ED}}}{Edges_{\text{Canny}}}\right) * 100 \tag{11}$$

Para evaluar la eficiencia de ABC-ED, se usa la cantidad de *cálculos de fitness* o, que es lo mismo, la cantidad de *análisis de pixeles* de la imagen necesarios para llegar a cierta eficacia. Además, se tiene como referencia el número máximo de ciclos (MCN) y el *tiempo de ejecución (t)*.

La doble umbralización del modelo, es donde se clasifica una fuente de comida como borde *débil* o borde *fuerte*, usando los umbrales μ_{min} y μ_{max}. Para obtener de forma automática estos umbrales para cada imagen de entrada, se utilizaron cuatro metodos de umbralización: Mean[6] [36], Median[6], Matlab[7] y Otsu[8] [37], los cuales se definen en la Tabla 1, en donde $mean(IM)$ es el valor promedio o media aritmética de IM, $median(IM)$ es el valor de mediana estadística de IM y μ_{Otsu}

[6] `http://www.kerrywong.com/2009/05/07/canny-edge-detection-auto-thresholding/`
[7] `http://www.mathworks.com/help/images/ref/edge.html`
[8] `http://www.mathworks.com/help/images/ref/multithresh.html`

es el valor dado al aplicar el método Otsu. Todos estos métodos son ejecutados en el ambiente de Matlab y el valor de ambos umbrales por cada método es enviado al prototipo.

Método	μ_{min}	μ_{max}	Proporción
Mean	$0.66 * mean(IM)$	$1.33 * mean(IM)$	$2.5 * \mu_{min} = \mu_{max}$
Median	$0.66 * median(IM)$	$1.33 * median(IM)$	$2.5 * \mu_{min} = \mu_{max}$
Matlab	Dado por Matlab	Dado por Matlab	$2.5 * \mu_{min} = \mu_{max}$
Otsu	$0.4 * \mu_{Otsu}$	μ_{Otsu}	$2.5 * \mu_{min} = \mu_{max}$

Tabla 1: Cálculo de umbral para cada método de umbralización. Fuente: Elaboración propia.

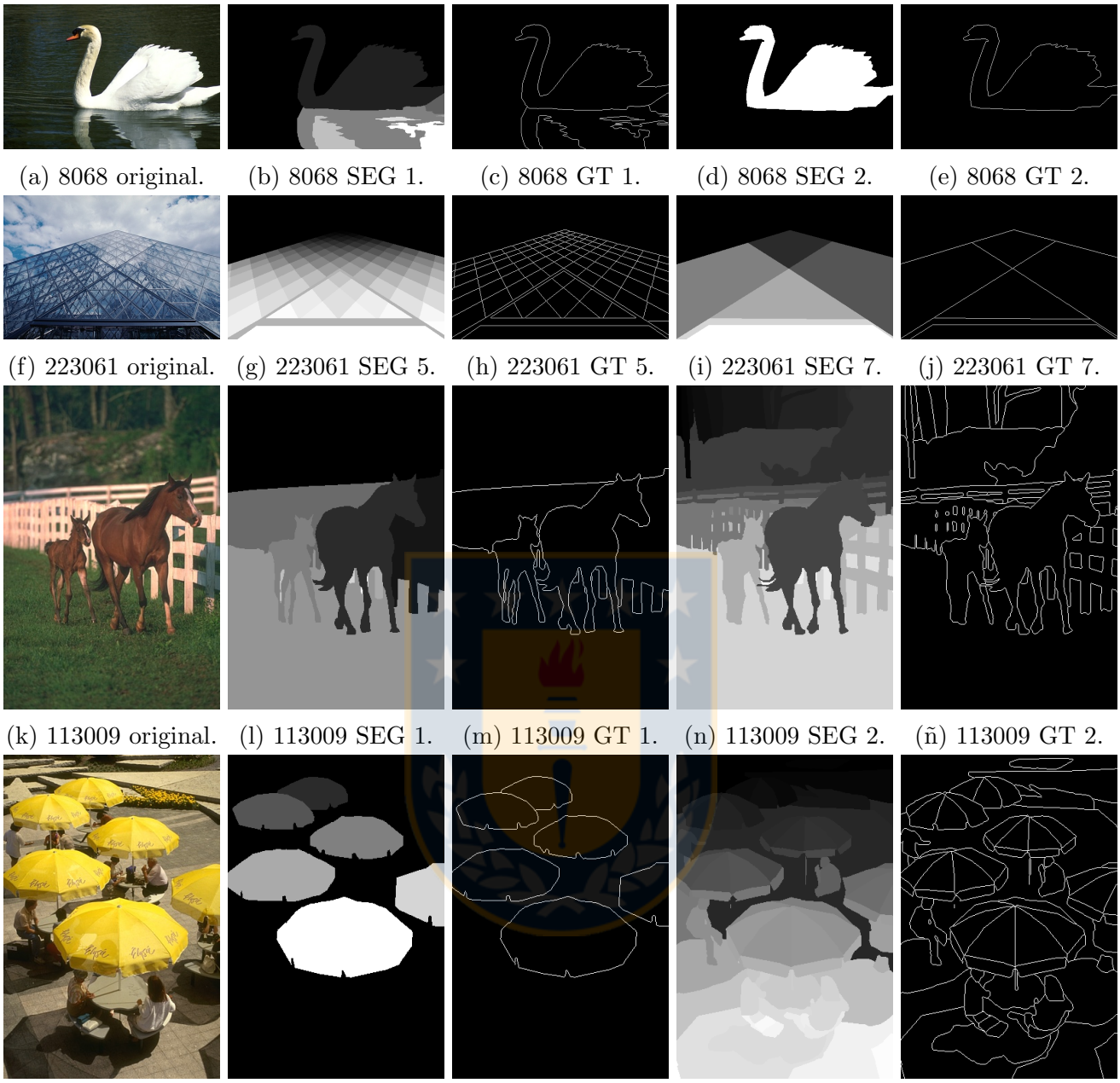
El *tiempo de ejecución (t)* se mide en segundos, y es calculado usando la función *clock*. El tiempo es tomado desde la inicialización del modelo hasta terminada la histéresis. El *tiempo de ejecución* incluye la captura de datos y métricas de evaluación requeridas en la experimentación.

Para la experimentación, se usó el dataset BSDS500[9] [5], el cual consiste de 500 imágenes naturales originales. Por cada imagen original, hay como promedio, 5 imágenes (niveles) en escala de grises manualmente segmentadas. Por cada imagen segmentada (SEG), está su imagen de terreno de verdad (GT) con sus bordes. Luego, como imágenes de entrada para la experimentación, se usaron todas las imágenes segmentadas del dataset, y las salidas del modelo se evaluaron mediante las métricas HD y C (Correlación)[10] [38], con su terreno de verdad respectivo. La métrica C es la medida de dependencia estadística entre dos matrices, con $0 \leq C \leq 1$, en donde $C = 0$ indica que las dos matrices son independientes, y $C = 1$ indica que las dos matrices son iguales. Ambas métricas son evaluadas en el ambiente Matlab y su resultado es enviado al prototipo. En la Figura 9 se presentan algunas imágenes del dataset.

Además, se usan abreviaciones y definiciones en la experimentación que se presentan mediante gráficos y tablas, con el fin de mostrar resultados de una forma ordenada. Las cuales son:

- TM := método de umbralización.
- Amount := cantidad de imágenes.
- AVG X := el promedio de alguna definición X.
- Cycle := la cantidad de ciclos ejecutados.
- uMin := el valor de umbral mínimo μ_{min}.
- uMax := el valor de umbral máximo μ_{max}.
- % EG := el grupo de porcentaje de eficacia ($\%\, eficacia$). El % EG g, corresponden a todas las salidas dadas por el modelo con un $\%\, eficacia \in\,](g-5)\,\%,\, g\,\%]$.
- % Fit := el porcentaje de *cálculos de fitness* o *análisis de pixeles* con respecto al total de pixeles de la imagen.

[9] `https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html`
[10] `http://www.mathworks.com/help/images/ref/corr2.html`



(a) 8068 original. (b) 8068 SEG 1. (c) 8068 GT 1. (d) 8068 SEG 2. (e) 8068 GT 2.

(f) 223061 original. (g) 223061 SEG 5. (h) 223061 GT 5. (i) 223061 SEG 7. (j) 223061 GT 7.

(k) 113009 original. (l) 113009 SEG 1. (m) 113009 GT 1. (n) 113009 SEG 2. (ñ) 113009 GT 2.

(o) 242078 original. (p) 242078 SEG 1. (q) 2420780 GT 1. (r) 242078 SEG 5. (s) 242078 GT 5.

Figura 9: Imágenes de dataset BSDS500. Original, segmentación (SEG) y terreno de verdad (GT). Fuente: [5].

- % Fit N := el porcentaje de *cálculos de fitness* realizados mediante una localización por vecindad con respecto al total de cálculos de *fitness* (Fit) realizados.
- % Fit E := el porcentaje de *cálculos de fitness* realizados mediante una localización por exploración con respecto al total de cálculos de *fitness* (Fit) realizados.
- % HD Canny := el porcentaje del valor de la métrica HD al comparar una salida del modelo con la salida dada por Canny bajo el mismo ambiente, con respecto al total de pixeles de la imagen.

- % HD GT := el porcentaje del valor de la métrica HD al comparar una salida del modelo con la imagen de terreno de verdad correspondiente, con respecto al total de pixeles de la imagen.
- C GT := el valor de la métrica C al comparar una salida del modelo con la imagen de terreno de verdad correspondiente.
- t := el tiempo de ejecución en segundos del modelo para obtener una imagen de salida.
- % Fit Group := el grupo de porcentaje de *cálculos de fitness* (% Fit). El % Fit Group f, corresponden a todas las salidas dadas por el modelo con un % Fit $\in](f-5)\,\%,\ f\,\%]$.

Para la experimentación, se diseñó una base de datos (bd) con el fin de poder almacenar los datos de eficacia, eficiencia y métricas de evaluación del modelo, y así de forma posterior poder analizar los resultados haciendo consultas a la bd. Para lo anterior, la bd se implementó usando el gestor MySQL, el cual se comunica con el prototipo por medio de MySQL Connector/C++, y se desarrolló en un entorno web, un despliegue de reportes de la experimentación mediante gráficos y tablas usando Google Charts [11], HTML, JavaScript, PHP y CSS.

Ya con todo lo necesario definido, se declara la experimentación con su propósito, para ser definida y realizada al aplicar el plan de prueba.

La experimentación tiene el propósito de analizar el desempeño del modelo usando un gran conjunto de entradas, de tal forma de analizar el promedio de sus resultados. El modelo se analiza en dos aspectos: el de evolución y el de completitud. El aspecto de evolución, comprende toda la ejecución del modelo y se analiza cómo este se desempeña en el tiempo en términos de cuántos pixeles van analizados y qué tan similiar va su detección comparándolo con la salida final de Canny y la imagen de terreno de verdad respectiva. El aspecto de completitud, comprende el término de ejecución del modelo, o que es lo mismo, cuándo este alcanza el 100 % de eficacia y llega a la misma salida dada por Canny, con el fin analizar cuantos pixeles el modelo necesitó como máximo para detectar lo mismo que Canny.

4.2. Aplicación de Plan de Prueba

4.2.1. Experimento

En la Figura 10 se presenta una tabla de análisis para el dataset usado en el experimento. Se usaron 2,696 imágenes de entrada (Img SEG), las cuales tienen una dimensión promedio de 369 x 432 pixeles. Para cada entrada, la salida del modelo se comparó con la salida final de Canny y con su imagen de terreno de verdad (GT).

Dataset	Amount	AVG Rows	AVG Columns
Img Original	500	369.64	432.36
Img SEG	2,696	369.96	432.04
Img GT	2,696	369.96	432.04

Figura 10: Análisis de dataset BSDS500. Fuente: Elaboración propia.

[11] `https://developers.google.com/chart/`

El valor de los parámetros del modelo usados en la experimentación para toda imagen de entrada se presentan en la Tabla 2. Con SN la raíz cuadrada del total de pixeles de la imagen de entrada. MCN con un valor lo suficientemente alto para que el modelo siempre termine su ejecución usando su mecanismo interno de término. $\{limit = 8, \varepsilon = 0\}$, debido a que se desea que el modelo realice la menor cantidad de *cálculos de fitness* posible, al analizar toda la vecindad por cada fuente y dar como preferencia para exploración una fuente inactiva [17].

SN	MCN	$limit$	ε
$\sqrt{r * c}$	10000	8	0

Tabla 2: Parámetros usados en experimentación para el modelo. Fuente: Elaboración propia.

Para realizar la experimentación, esta se implementó en el prototipo, lo cual se presenta como pseudo-algoritmo en Algoritmo 12. Para cada imagen de entrada, por cada método de umbralización se calcula el valor de los umbrales μ_{min} y μ_{max}, los cuales son usados al ejecutar Canny y ABC-ED. Al ejecutar ABC-ED, por cada ciclo de ejecución, la salida del ciclo es comparada con la salida final de Canny y con el GT correspondiente mediante las métricas HD y C, y se calcula los datos de eficacia y eficiencia. Los resultados de Canny y ABC-ED son guardados en la base de datos. Por último, se guardan las imágenes de salidas.

Algoritmo 12 Pseudo-algoritmo de experimentación de ABC-ED. Fuente: Elaboración propia.

Input: Conjunto de imágenes de entrada y terreno de verdad.

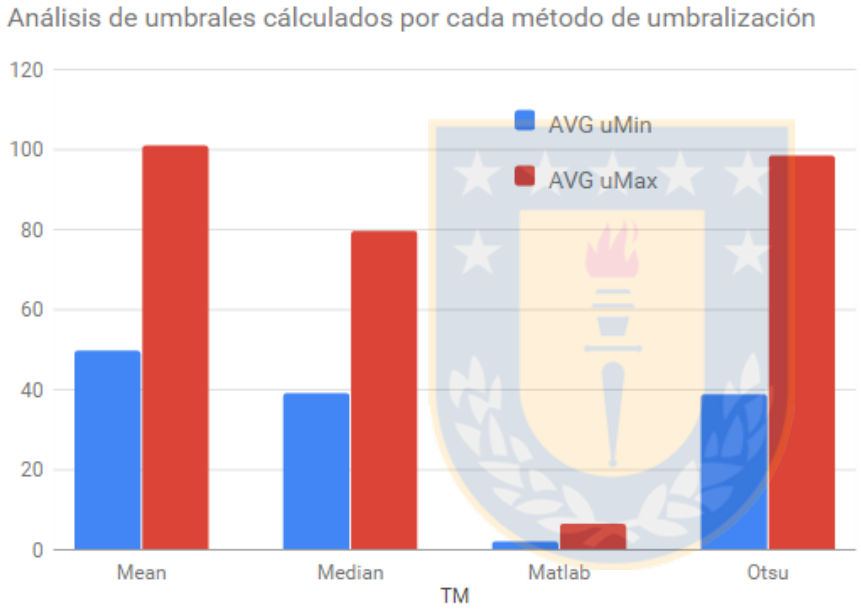
```
1: img_input: una imagen de entrada.
2: img_gt: una imagen de terreno de verdad.
3: tm: un método de umbralización.
4: procedure EXPERIMENT( )
5:     SN; MCN = 10000; limit = 8; ε = 0; umbrales;
6:     for each img_input do
7:         SN ← CALCULAR-SN(img_input);
8:         img_gt ← OBTENER-GT(img_input);
9:         for each tm do
10:            umbrales ← CALCULAR-UMBRALES(img_input, tm);
11:            EJECUTAR-CANNY(img_input, umbrales, img_gt);
12:            EJECUTAR-ABC-ED(img_input, umbrales, SN, MCN, limit, ε, img_gt);
13:            GUARDAR-IMAGENES-DE-SALIDA( );
14:        end for
15:    end for
16: end procedure
```

Para una justa comparación de la salida de cada ciclo de la fase 1 del modelo con la salida final de Canny SC, luego de terminado un ciclo de ejecución de la fase 1 del modelo, dando la salida $S1$, se le aplica la fase 2 del modelo, dando la salida $S2$, y esta salida $S2$ es la que se compara con la salida de Canny, dado que ambas salidas (SC y $S2$) presentan bordes adelgazados, doble umbralización e histéresis. Luego, se continúa con el siguiente ciclo del modelo en fase 1 con el mismo estado del modelo al haber obtenido $S1$.

La ejecución del experimento tuvo una duración de 18 días, 2 horas y 54 minutos en una maquina con un CPU Intel Core i5 2.30GHz y 4 GB RAM.

Ya con el experimento definido, se presentan los resultados obtenidos y se analizan. Primero, se realiza un análisis de los valores dados por los métodos de umbralización. Luego, se analizan los resultados del modelo en su aspecto de evolución. Por último, se analizan los resultados del modelo en su aspecto de completitud. Para todo análisis, se consideran todas las imágenes de entrada, y por ende, todas las salidas del modelo promediando sus resultados.

En la Figura 11, se presenta como promedio los valores calculados automáticamente por cada método de umbralización para todas las imágenes de entrada. Se observa que los valores dados por el método Matlab son bastante menores que el resto, implicando que con este método, el modelo acepta más frecuentemente un pixel como una fuente que el resto de los otros métodos.



(a) Gráfico de análisis de los métodos de umbralización.

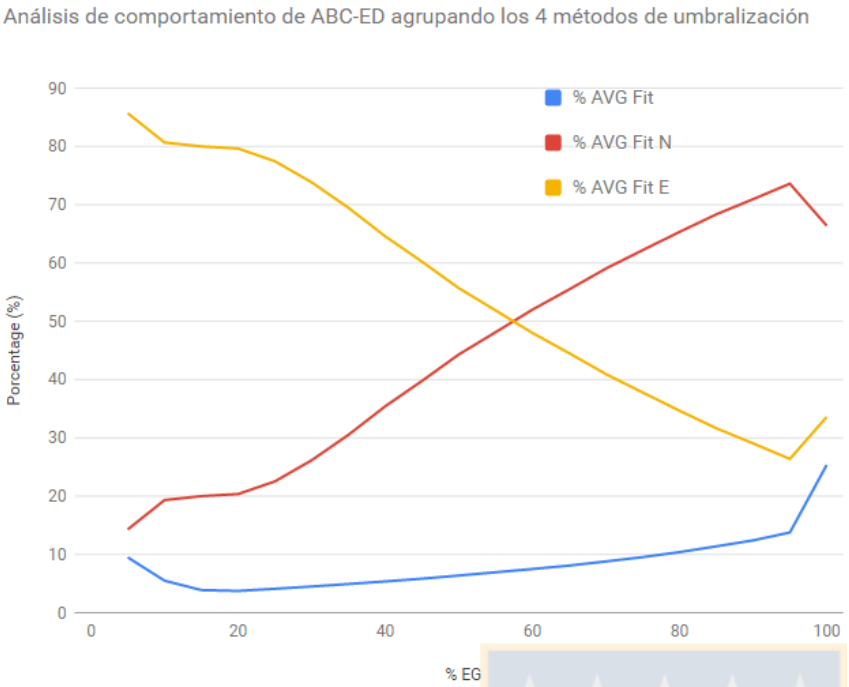
TM	AVG uMin	AVG uMax
Mean	49.99	101.25
Median	39.38	79.94
Matlab	2.36	6.74
Otsu	39.06	98.76

(b) Tabla de Figura 11a.

Figura 11: Umbrales cálculados de los métodos de umbralización. Fuente: Elaboración propia.

Para analizar el aspecto de evolución del modelo, en la Figura 12 se presentan los gráficos de análisis de comportamiento (Figura 12a) y el de análisis de métricas (Figura 12c), ambos con su tabla de valores correspondiente a su derecha. Estos resultados son una agrupación promedio de los resultados de los cuatro métodos de umbralización, los cuales consideran todas las salidas del modelo de forma promedio y se presentan de forma separada en el Anexo C.

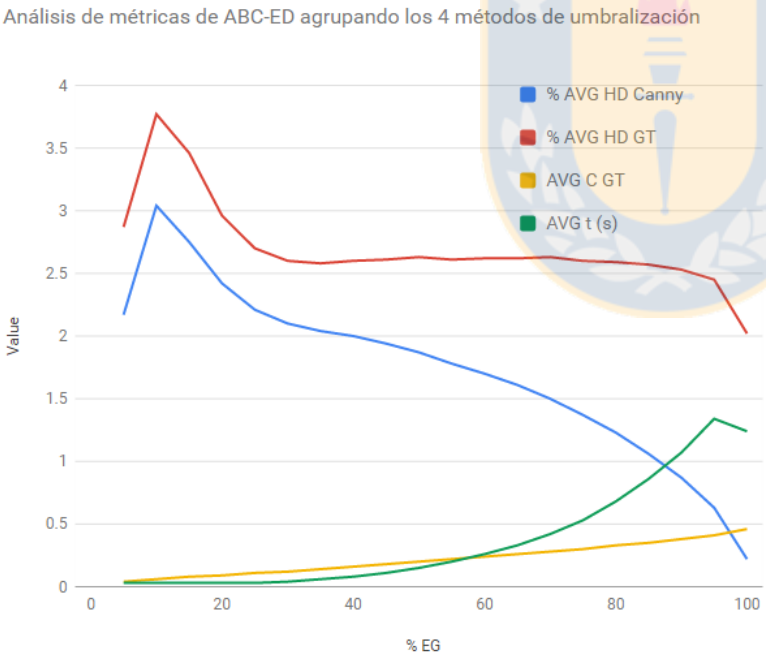
Para ambos gráficos, los resultados de todas las salidas del modelo son agrupados al grupo de porcentaje de eficacia (% EG) que corresponda, con el fin de saber, la evolución promedia, en términos de la cantidad de *cálculos de fitness* (% Fit) o *análisis de pixeles*, la diferencia con respecto a la salida de Canny (HD Canny) y a su terreno de verdad (HD GT), y su dependencia hacia este último (C GT).



(a) Gráfico de análisis de comportamiento.

% EG	% AVG Fit	% AVG Fit N	% AVG Fit E
5	9.5	14.29	85.7
10	5.5	19.32	80.68
15	3.89	20	80
20	3.74	20.36	79.64
25	4.11	22.53	77.47
30	4.5	26.16	73.84
35	4.93	30.52	69.48
40	5.37	35.42	64.58
45	5.84	39.77	60.23
50	6.38	44.31	55.69
55	6.94	48.13	51.87
60	7.5	51.99	48.01
65	8.09	55.47	44.53
70	8.8	59.05	40.95
75	9.54	62.21	37.79
80	10.4	65.34	34.66
85	11.39	68.35	31.65
90	12.41	70.95	29.05
95	13.77	73.61	26.39
100	25.35	66.4	33.6

(b) Tabla de Figura 12a.



(c) Gráfico de análisis de métricas.

% EG	% AVG HD Canny	% AVG HD GT	AVG C GT	AVG t (s)
5	2.17	2.87	0.04	0.03
10	3.04	3.77	0.06	0.03
15	2.75	3.46	0.08	0.03
20	2.42	2.96	0.09	0.03
25	2.21	2.7	0.11	0.03
30	2.1	2.6	0.12	0.04
35	2.04	2.58	0.14	0.06
40	2	2.6	0.16	0.08
45	1.94	2.61	0.18	0.11
50	1.87	2.63	0.2	0.15
55	1.78	2.61	0.22	0.2
60	1.7	2.62	0.24	0.26
65	1.61	2.62	0.26	0.33
70	1.5	2.63	0.28	0.42
75	1.37	2.6	0.3	0.53
80	1.23	2.59	0.33	0.68
85	1.06	2.57	0.35	0.86
90	0.87	2.53	0.38	1.07
95	0.63	2.45	0.41	1.34
100	0.22	2.02	0.46	1.24

(d) Tabla de Figura 12c.

Figura 12: Análisis de comportamiento y métricas agrupando los TM. Fuente: Elaboración propia.

En el gráfico de análisis de comportamiento, se observa que a pesar de haber forzado el modelo a realizar una localización por vecindad máxima por medio de los parámetros *limit* y ε, se requirió de un $\{\,\%\,EG = 60\}$ para que la búsqueda por vecindad (% Fit N) sea mayor que una búsqueda aleatoria (% Fit E) en la imagen, sin embargo, sirvió para que el modelo aumente lo menos posible la cantidad de pixeles que tuvo que analizar. Para los tres primeros grupos de eficacia (5 %, 10 % y 15 %), se observa que hay un descenso en el % Fit y luego este crece hasta el $\{\,\%\,EG = 100\}$. Lo anterior es debido a los métodos de umbralización Mean y Median, con sus altos valores de umbrales para ciertas imágenes, el modelo se inicializó con una elevada cantidad de *análisis de pixeles*; puesto que un alto valor de umbral implica una menor probabilidad de crear un pixel como fuente de comida. Lo anterior no sucede para los métodos Matlab y Otsu. Por último, para un $\{\,\%\,EG = 100\}$, en donde el $\%\,eficacia \in]95\,\%, 100\,\%]$, i.e., el último 5 % de eficacia para obtener la misma salida que Canny, el modelo necesitó una cantidad de *análisis de pixeles* promedio de un **25.35 %** con respecto al total de pixeles de la imagen.

En el gráfico de análisis de métricas, se observa la evolución del modelo con un decremento continuo en la diferencia que existe entre las salidas del modelo y la salida final de Canny respectiva, lo cual decrece de forma mayor que con respecto a la comparación con el terreno de verdad respectivo (GT), debido a la precisión de la identificación del detector de bordes de Canny, el cual está integrado en el modelo. El aumento inicial en la diferencia entre las salidas de Canny y GT es debido a los métodos de umbralización Mean y Median con sus altos valores de umbrales para ciertas imágenes, en particular un alto μ_{max}, implica una baja frecuencia en clasificar una fuente como borde *fuerte*, y para este caso, el valor de μ_{max} fue tan elevado que el modelo no pudo clasificar alguna fuente como borde *fuerte*, y por ende da como salida una imagen negra (sin bordes detectados). Lo anterior no sucede para los métodos Matlab y Otsu. La correlación entre las salidas del modelo y el GT respectivo aumenta continuamente en la evolución del modelo hasta llegar a un promedio de 0.46. Por último, para un $\{\,\%\,EG = 100\}$, las salidas del modelo tienen una diferencia promedio con la salida de Canny respectiva de un **0.22 %** con respecto al total de pixeles de la imagen.

A continuación, se analiza el aspecto de evolución del modelo haciendo una comparación entre los métodos de umbralización. Se compara la cantidad de *cálculos de fitness*, el porcentaje de diferencia entre las salidas del modelo, la salida de Canny y el GT respectivo, la correlación con respecto a GT y los ciclos de ejecución.

En la Figura 13, se presenta el análisis promedio de *cálculos de fitness* por cada método de umbralización. Se observa que los métodos Matlab y Otsu ofrecen un mejor cálculo de los valores de umbral para cada imagen, ya que no presentan un elevado *análisis de pixeles* inicial como Mean y Median. Sin embargo, los cuatro métodos hacen que el modelo termine en un valor muy cercano de % Fit promedio.

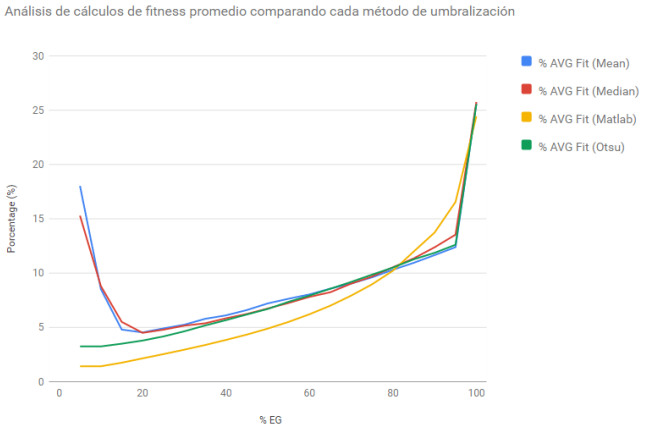


Figura 13: Análisis promedio de *cálculos de fitness* para cada TM. Fuente: Elaboración propia.

En la Figura 14, se presenta el análisis promedio de la diferencia de las salidas del modelo entre Canny y el terreno de verdad (GT) durante la evolución del modelo para cada método de umbralización. Con el método Otsu, el modelo presenta más igualdad a la salida de Canny durante toda su evolución, seguido por Mean, Median y Matlab. Los métodos Otsu y Mean hacen que el modelo obtenga la menor diferencia hacia GT, seguido por Median y Matlab. El modelo llega a misma salida de Canny con todos los métodos de umbralización, ya que ambos poseen la misma identificación, pero con ningún método pudo igualar el terreno de verdad, debido a la precisión de identificación de Canny. Lo anterior reafirma que la precisión de identificación del modelo ABC-ED es totalmente dependiente de la integración que se le haga.

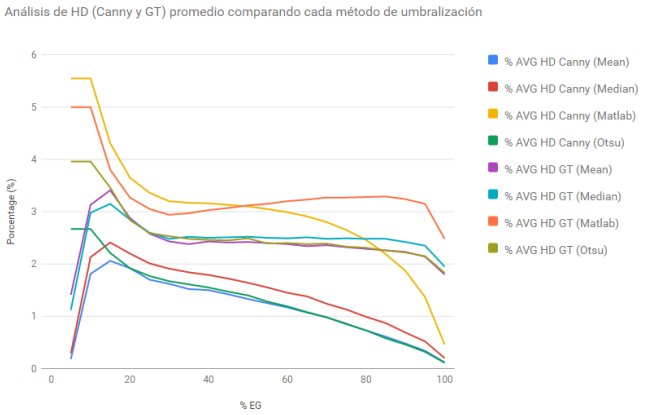


Figura 14: Análisis promedio de % HD Canny y % HD GT por TM. Fuente: Elaboración propia.

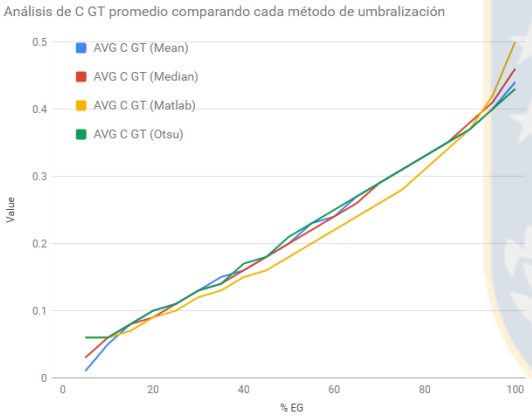


Figura 15: Análisis promedio de C GT por TM. Fuente: Elaboración propia.

En la Figura 15, se presenta el análisis promedio de la métrica C entre las salidas del modelo y el terreno de verdad para cada método de umbralización. Se observa que el modelo obtiene muy similares resultados para cualquier método. Lo anterior implica que la correlación es dependiente de la precisión de identificación que se le integre. Se destaca que con Matlab se obtiene la mejor correlación final, pero no durante la evolución del modelo, en donde para $\{\,\%\,EG = 95\}$ la C es muy cercana entre los métodos.

En la Figura 16, se presenta el análisis promedio de ciclos de ejecución del modelo para cada método de umbralización. Se observa en la evolución del modelo que hasta el $\{\,\%\,EG = 95\}$, para todos los métodos de umbralización, el modelo requiere de una cantidad de ciclos muy similar, pero para el último 5 % de eficacia, con los métodos Otsu y Mean, el modelo requiere de menor cantidad de ciclos que los métodos Median y Matlab. Lo anterior es debido a los valores de umbrales calculados para cada imagen. Un menor valor en los umbrales, implica que el modelo va creando una mayor cantidad de fuentes durante su evolución, lo que provoca como consecuencia en su mecanismo interno de término de necesitar una localización de los pixeles dispersos no analizados en su evolución, por ende, necesitando de más ciclos y un mayor *tiempo de ejecución*. Lo anterior, es lo que le pasó al modelo, en mayor magnitud, al usar el método Matlab.

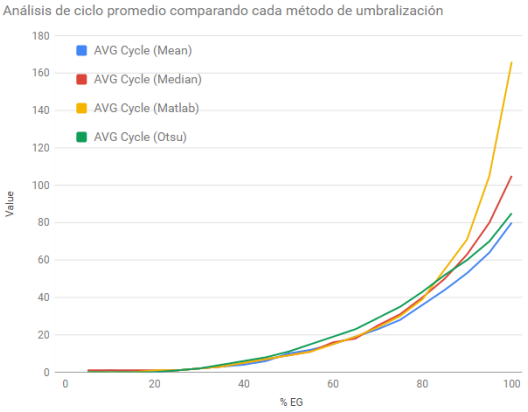


Figura 16: Análisis promedio de ciclos por TM. Fuente: Elaboración propia.

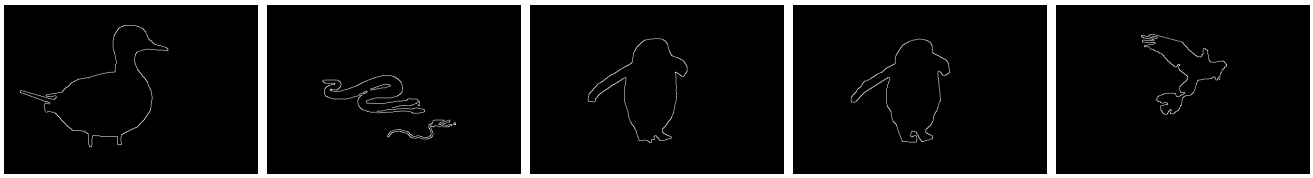
A continuación, se realiza un análisis del aspecto de completitud del modelo, i.e., cuando cada salida del modelo es igual a la salida de Canny respectiva, o de forma cuantitativa, cuando cada salida tiene un 100 % de eficacia y el % HD Canny = 0.

En la Tabla 3, se presentan los resultados de las 10 mejores salidas del modelo en términos de *cálculos de fitness* (% Fit) o *análisis de pixeles* ordenados de forma ascendente. Para la primera salida (Rank = 1), el modelo necesitó de 57 ciclos de ejecución, de un **10.24 %** de *análisis de pixeles* del total de pixeles de la imagen, teniendo un **0.78 %** de diferencia con el terreno de verdad, una correlación de 0.5 y necesitando de **0.1** segundos en *tiempo de ejecución*. En la Figura 17 se presentan estas salidas ordenadas acorde a su Rank.

Rank	TM	uMin	uMax	Img Original ID	Img Level	% Efficacy	Cycle	% Fit	% HD Canny	% HD GT	C GT	t (s)
1	Median	1	3	15,088	1	100	57	10.24	0	0.78	0.5	0.1
2	Matlab	2	7	54,005	2	100	52	10.29	0	0.5	0.58	0.1
3	Mean	1	3	15,088	2	100	55	10.36	0	0.79	0.49	0.1
4	Matlab	3	9	108,069	1	100	57	10.48	0	0.58	0.49	0.11
5	Median	1	3	15,088	2	100	58	10.55	0	0.79	0.49	0.1
6	Median	1	3	43,051	3	100	70	10.59	0	0.6	0.55	0.14
7	Mean	1	3	196,073	5	100	74	10.63	0	0.91	0.47	0.15
8	Median	1	3	106,047	3	100	54	10.67	0	0.58	0.43	0.11
9	Matlab	3	9	106,047	2	100	58	10.68	0	0.59	0.45	0.12
10	Mean	2	5	135,069	4	100	51	10.72	0	0.56	0.49	0.09

Tabla 3: Las 10 salidas con 100 % de eficacia con mejor % Fit. Fuente: Elaboración propia.

(a) Rank 1. (b) Rank 2. (c) Rank 3. (d) Rank 4. (e) Rank 5.



(f) Rank 6. (g) Rank 7. (h) Rank 8. (i) Rank 9. (j) Rank 10.

Figura 17: Salidas con el mejor % Fit acorde a su Rank de la Figura **??**. Fuente: Elaboración propia.

En la Tabla 4, se presentan los resultados de las 10 mejores salidas del modelo en términos de la diferencia que tienen con su terreno de verdad (GT) respectivo ordenados de forma ascendente. De la tabla y el resto de los resultados, se infiere que la diferencia con GT es independiente del método de umbralización y de la cantidad de *análisis de pixeles*, por lo que solo depende de la precisión de la identificación integrada.

Rank	TM	uMin	uMax	Img Original ID	Img Level	% Efficacy	Cycle	% Fit	% HD Canny	% HD GT	C GT	t (s)
1	Median	1	3	28,096	2	100	6	40.28	0	0.09	0.49	0.03
2	Matlab	1	4	28,096	2	100	7	40.35	0	0.09	0.49	0.03
3	Mean	1	3	28,096	2	100	5	41.29	0	0.09	0.49	0.03
4	Otsu	25	64	28,096	2	100	3	47.77	0	0.09	0.49	0.03
5	Matlab	3	9	28,096	1	100	6	40.02	0	0.1	0.49	0.03
6	Median	1	3	28,096	1	100	6	40.58	0	0.1	0.49	0.03
7	Mean	1	3	28,096	1	100	6	40.82	0	0.1	0.49	0.02
8	Otsu	50	127	28,096	1	100	4	50.27	0	0.1	0.49	0.03
9	Median	1	3	28,096	4	100	7	39.22	0	0.13	0.39	0.04
10	Mean	1	3	28,096	4	100	8	40.65	0	0.13	0.39	0.04

Tabla 4: Las 10 salidas con 100 % de eficacia con el mejor % HD GT. Fuente: Elaboración propia.

En la Tabla 5, se presentan los resultados de las 10 mejores salidas del modelo en términos del valor de correlación ordenados de forma descendente. De la tabla y el resto de los resultados, se infiere que la correlación es independiente del método de umbralización, de la cantidad de *análisis de pixeles* y de la diferencia con GT, por lo que solo depende de la identificación integrada, al igual que la diferencia con el terreno de verdad.

Rank	TM	uMin	uMax	Img Original ID	Img Level	% Efficacy	Cycle	% Fit	% HD Canny	% HD GT	C GT	t (s)
1	Matlab	2	5	178,054	2	100	54	11.37	0	0.19	0.85	0.07
2	Otsu	76	191	178,054	2	100	35	13.81	0	0.19	0.85	0.1
3	Mean	126	255	178,054	2	100	37	100	0	0.19	0.85	0.11
4	Matlab	1	3	78,019	4	100	303	100	0	1.13	0.83	2.61
5	Matlab	1	4	21,077	3	100	127	100	0	0.46	0.82	0.36
6	Otsu	38	96	374,020	1	100	81	100	0	0.45	0.82	0.45
7	Matlab	1	2	374,020	1	100	124	100	0	0.45	0.82	0.3
8	Matlab	1	2	374,020	3	100	121	13.6	0	0.46	0.81	0.27
9	Otsu	38	96	374,020	3	100	81	100	0	0.46	0.81	0.44
10	Matlab	3	8	69,007	7	100	220	100	0	0.86	0.81	1.67

Tabla 5: Las 10 salidas con 100 % de eficacia con el mejor C GT. Fuente: Elaboración propia.

En la Tabla 6, se presenta un análisis de todas las salidas del modelo con 100 % de eficacia agrupadas acorde al grupo de porcentaje de *cálculos de fitness* (% Fit Group) que correspondan. Se observa que se presentan dos concentraciones principales de salidas en ambos extremos de la tabla. La primera concentración es hasta el grupo de 25 % de *análisis de pixeles* y la segunda concentración es el último grupo. Observar que como se consideran cuatro métodos de umbralización, cada imagen de entrada está considerada cuatro veces en los resultados.

La primera concentración de salidas esta relacionada a los resultados de *análisis de pixeles* del aspecto de evolución del modelo, en donde para todo método de umbralización, el modelo necesitó de un 25.35 % promedio de *análisis de pixeles* para el último grupo de eficacia. Además, haciendo una comparación entre ambas concentraciones de salidas, en la primera concentración se observa que a menor grupo de % Fit, la salidas del modelo son más similares a su terreno de verdad respectivo que la segunda concentración de salidas, necesitando de menos ciclos promedios de ejecución y teniendo más bajo el valor promedio de ambos umbrales. Por lo anterior, el modelo es más preciso en su detección de bordes a menor cantidad de *análisis de pixeles* que realice.

Amount	% Efficacy	% HD Canny	% Fit Group	% AVG Fit	AVG uMin	AVG uMax	AVG Cycle	% AVG HD GT
3,224	100	0	15	12.27	26	58	82	1.03
1,079	100	0	20	16.94	15	30	122	1.54
518	100	0	25	21.89	24	44	131	1.79
52	100	0	30	27.22	35	64	154	2.51
53	100	0	35	32.78	48	108	125	2.39
46	100	0	40	37.51	35	80	149	2.56
58	100	0	45	42.23	33	75	116	2.12
51	100	0	50	47.33	37	81	133	2.3
39	100	0	55	52.55	49	111	107	2.35
33	100	0	60	57.45	38	83	135	2.18
40	100	0	65	62.29	42	94	114	2.23
45	100	0	70	67.48	40	91	122	2.27
33	100	0	75	72.54	46	104	107	2.18
44	100	0	80	77.7	43	95	129	2.23
41	100	0	85	82.71	42	96	119	2.24
49	100	0	90	87.37	43	98	121	2.2
67	100	0	95	92.76	40	85	145	2.23
5,312	100	0	100	99.94	35	74	151	2.01

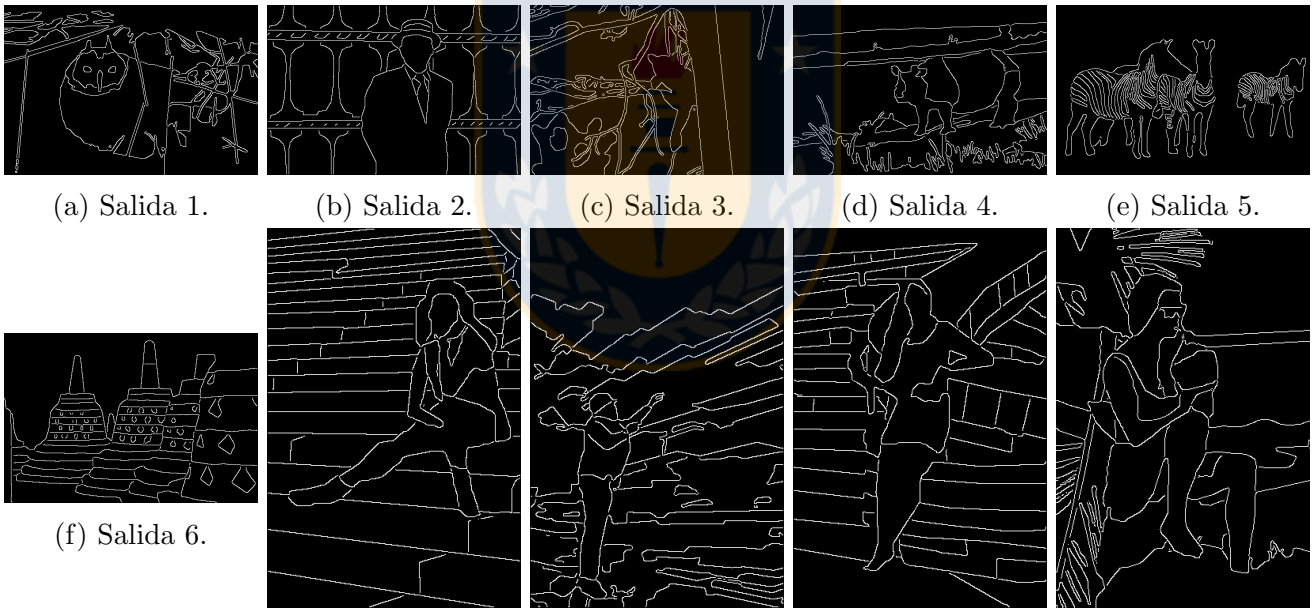
Tabla 6: Análisis de todas las salidas con 100 % de eficacia agrupadas por el % Fit correspondiente. Fuente: Elaboración propia.

En la Figura 17 se presentan salidas del modelo que corresponden a la primera concentración de salidas, en particular, al primer grupo de *análisis de pixeles* (% Fit Group = 15), y en la Figura 18 se presentan salidas del modelo que corresponden a la segunda concentración de salidas. Haciendo una comparación entre las salidas de ambos extremos de concentraciones, y por ende, una comparación entre sus imágenes de entrada respectiva, se concluye que la razón de por qué el modelo varía en la cantidad de *análisis de pixeles* que realiza, es debido al tipo de imagen, en términos de la cantidad de bordes que presenta y cómo estos están dispersos en la imagen. Luego, el modelo tiene un mejor desempeño en las imágenes con menos regiones de objetos y más focalizadas que en imágenes más complejas con gran cantidad de contornos distribuidos, sin embargo, para este último tipo de imágenes, el modelo necesita un promedio de 25.35 % de *análisis de pixeles* para detectar el 95 % de lo que detecta Canny, el cual requiere de analizar toda la imagen.

En la Figura 19, se presenta una comparación visual de lo analizado anteriormente. Se presentan cuatro salidas del modelo, en donde las primeras dos salidas corresponden a las Figuras 17a

y 17j del (% Fit Group = 15) de la primera concentración, y las dos últimas salidas corresponden a las Figuras 18a y 18j de la segunda concentración. Por cada fila en la Figura 19 se presenta la imagen de entrada I, la salida del modelo al obtener el 95 % y 100 % de eficacia y el terreno de verdad de I. Haciendo una comparación entre las salidas del 95 % y 100 % de eficacia, se observa que la salida del 95 % presenta una leve diferencia en lo detectado con respecto a la de 100 %, debido a que el modelo ya tiene detectado la estructura de bordes representativa de I, obteniendo así una buena aproximación al 100 % de eficacia y al terreno de verdad, pero necesitando de un promedio de 25.35 % de *análisis de pixeles*. Haciendo una comparación de la salida del 95 % o 100 % de eficacia con el terreno de verdad, se observa visualmente una gran igualdad estructural, sin embargo, el modelo detecta los bordes con un leve desplazamiento con respecto a su ubicación en el terreno de verdad. Lo anterior se debe a la precisión de identificación de Canny integrada al modelo.

El promedio ponderado de *análisis de pixeles* del modelo, considerando todas las salidas con 100 % de eficacia para el dataset BSDS500 usado, es de un 59.25 %. Este valor está en función del tipo de imágenes de entrada del dataset. Dado que el dataset tiene una mayor cantidad de imágenes de entrada con elevada cantidad de bordes distribuidos a detectar, el valor del promedio ponderado de *análisis de pixeles* del modelo sube.

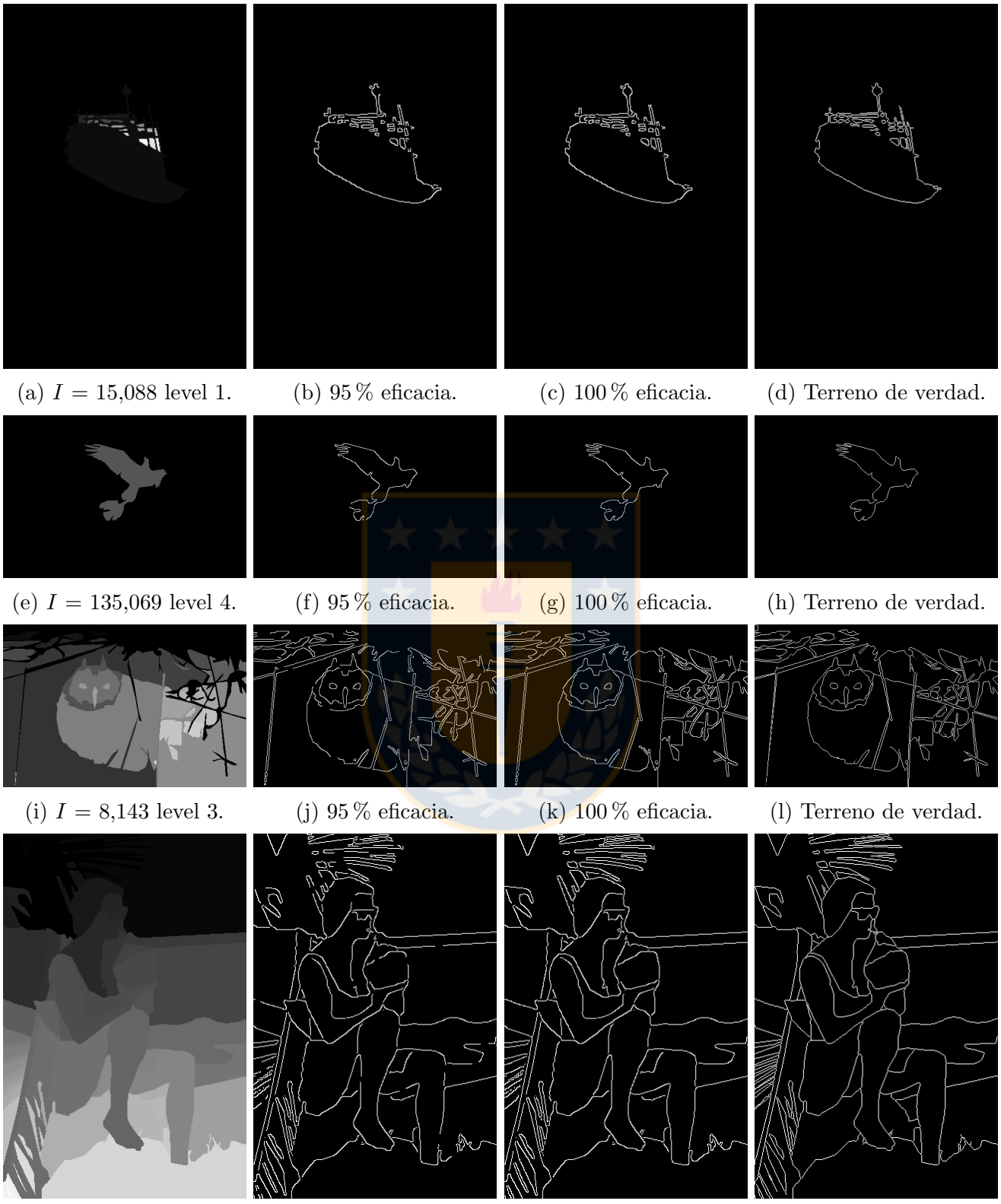


(a) Salida 1. (b) Salida 2. (c) Salida 3. (d) Salida 4. (e) Salida 5.

(f) Salida 6.

(g) Salida 7. (h) Salida 8. (i) Salida 9. (j) Salida 10.

Figura 18: Imágenes de salida pertenecientes a { % Fit Group = 100}. Fuente: Elaboración propia.



(a) I = 15,088 level 1. (b) 95 % eficacia. (c) 100 % eficacia. (d) Terreno de verdad.

(e) I = 135,069 level 4. (f) 95 % eficacia. (g) 100 % eficacia. (h) Terreno de verdad.

(i) I = 8,143 level 3. (j) 95 % eficacia. (k) 100 % eficacia. (l) Terreno de verdad.

(m) I = 81,090 level 4. (n) 95 % eficacia. (ñ) 100 % eficacia. (o) Terreno de verdad.

Figura 19: Muestra de salidas de ABC-ED. En orden: la imagen de entrada I, la salida del modelo al obtener el 95 % y 100 % de eficacia y el terreno de verdad de I. Fuente: Elaboración propia.

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 el detector de bordes de Canny para la identificación y el algoritmo ABC para la localización, obteniendo así el modelo ABC-ED desarrollado en este trabajo, del cual se implementó un prototipo usado para evaluar experimentalmente el desempeño del modelo. Por ende, se comprobó la hipótesis planteada al solucionar exitosamente el problema propuesto, cumpliendo el objetivo general y específicos al usar correctamente la metodología definida para este trabajo.

El algoritmo ABC simula el comportamiento natural de las abejas de miel en su recolección de comida, logrando un inteligente algoritmo de búsqueda que combina la explotación con la exploración de una forma descentralizada y auto-organizada. Este algoritmo es integrado con el detector de bordes de Canny para lograr una eficiente búsqueda de bordes en una imagen, creando así el modelo ABC-ED. Luego, el modelo ABC-ED detecta los bordes de una imagen combinando una búsqueda local por vecindad con una búsqueda global por exploración, y clasifica un pixel como borde por medio de la identificación de Canny.

Para la creación del modelo ABC-ED, 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 analizar su desempeño usando un gran conjunto de entradas, de tal forma de analizar el promedio de sus resultados. Dos aspectos del modelo fueron analizados: el de evolución y el de completitud. En el aspecto de evolución, se analiza el desempeño durante toda su ejecución, y en el aspecto de completitud, se analiza el desempeño cuando termina su ejecución. El análisis de desempeño comprende la cantidad de *análisis de pixeles*, la diferencia entre la salida del modelo y la salida final de Canny y su terreno de verdad respectivo, y ciclos de ejecución.

En la experimentación, se usaron cuatro métodos de umbralización: Mean, Median, Matlab y Otsu. Además, para evaluar las salidas del modelo con las salidas de Canny y el terreno de verdad, se usaron las métricas Distancia de Hamming (HD) y Correlación (C). Considerando todo análisis de desempeño, el modelo obtiene similares resultados para cualquier método usado, pero logra sus mejores resultados con el método de umbralización Otsu, seguido cerca por Mean y por último un empate entre Matlab y Median.

Dado que el modelo integra la identificación de Canny, el modelo detecta a lo más todo lo que detecta Canny pero con una diferente búsqueda. Luego, la eficacia del modelo corresponde a la precisión de identificación de Canny. Por lo anterior, los resultados del modelo fueron agrupados según el porcentaje de eficacia que corresponda, creando así, grupos de porcentaje de eficacia (% EG) agrupados cada 5 %, por lo que el % EG g, corresponden a las salidas del modelo que tienen un porcentaje de eficacia $\in](g-5)\,\%,\ g\,\%]$.

Para un $\{\,\%\,EG\ =\ 100\}$, i.e., el último 5 % de eficacia para obtener la misma salida que

Canny, el modelo necesitó, como promedio, una cantidad de *análisis de pixeles* de un **25.35 %** teniendo una diferencia con la salida final de Canny respectiva de un **0.22 %** con respecto al total de pixeles de la imagen.

El modelo llega a misma salida de Canny con todos los métodos de umbralización, ya que ambos poseen la misma identificación, pero con ningún método pudo obtener la igualdad con el terreno de verdad, debido a la precisión de identificación de Canny; puesto que la precisión de identificación del modelo ABC-ED es totalmente dependiente de la integración que se le haga.

El modelo es más preciso en su detección de bordes a menor cantidad de *análisis de pixeles* que realice; puesto que tiene un mejor desempeño en las imágenes con menos regiones de objetos y más focalizadas que en imágenes más complejas con gran cantidad de contornos distribuidos, sin embargo, para este último tipo de imágenes, el modelo necesita un promedio de 25.35 % de *análisis de pixeles* para detectar el 95 % de lo que detecta Canny, el cual requiere de analizar siempre toda la imagen. Logrando una buena aproximación visual con una leve diferencia a la salida final en donde ya se presenta la estructura de bordes representativa de la imagen.

Luego, y por lo tanto, se propone como **trabajo futuro** la continuación en la mejora de eficiencia y eficacia del modelo. Para la eficiencia, se proponen cuatro mejoras: Fase de Obreras Exploradoras por demanda, completo análisis local de vecindad, disminución del tamaño de la población de fuentes SN y mayor integración con Canny. Para la eficacia, se propone integrar las mejoras de Canny que han sido desarrolladas en el tiempo o integrar un método con una identificación de bordes más precisa.

La Fase de Obreras Exploradoras por demanda, consiste en la eliminación de esta fase en el ciclo de ejecución del modelo, pero al momento de que alguna fuente se agote, a esta se le aplique exploración inmediatamente. Lo anterior, aceleraría el *tiempo de ejecución* del modelo en sus otras dos fases considerablemente, siendo más acorde al comportamiento real de las abejas de miel.

El completo análisis local de vecindad, consiste en que al momento de crear una fuente, a esta se le analice toda su vecindad, con el fin de suprimir inmediatamente vecinos que no sean fuentes, logrando que el modelo no los considere al momento de elegir un vecino candidato en su localización por vecindad. Lo anterior no aumentará la cantidad de *análisis de pixeles* considerablemente; puesto que la vecindad de una fuente es analizada en el proceso de ejecución del modelo, pero si aceleraría al modelo disminuyendo su cantidad de ciclos de ejecución y por ende su *tiempo de ejecución*.

La disminución del tamaño de la población, consiste en asignar un menor valor al parámetro SN, pero manteniendo su relación con el tamaño de la imagen. Lo anterior, podría mejorar completamente el modelo al necesitar una menor cantidad de *análisis de pixeles*, ciclos de ejecución MCN y *tiempo de ejecución*. Con $r * c$, el tamaño de la imagen y dado que ya se experimentó con $SN = \sqrt[2]{r * c}$, se sugiere $SN \in \{\sqrt[3]{r * c}, \sqrt[4]{r * c}, \sqrt[5]{r * c}\}$.

La mayor integración con Canny, consiste de eliminar la técnica de adelgazamiento de bordes de la fase 2 del modelo, pero integrarla a su fase 1. Una correcta integración, podría mejorar al modelo en necesitar una menor cantidad de *análisis de pixeles*, ciclos de ejecución y *tiempo de ejecución* al ir suprimiendo fuentes en la fase 1 que posteriormente hubiesen sido eliminadas en la fase 2.

Bibliografía

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

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

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

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

[5] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

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

[14] 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.

[15] 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.

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

[17] Jaime Vásquez Feijoo. Localización eficiente en detección de bordes en imágenes adaptando el algoritmo abc. *Memoria de Titulo. Universidad de Concepción*, 2016.

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

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

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

[21] 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.

[22] 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.

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

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

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[34] M AdelsonVelskii and Evgenii Mikhailovich Landis. An algorithm for the organization of information. Technical report, DTIC Document, 1963.

[35] Richard W Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

[36] Chris A. Glasbey. An analysis of histogram-based thresholding algorithms. *CVGIP: Graphical models and image processing*, 55(6):532–537, 1993.

[37] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[38] Gábor J Székely, Maria L Rizzo, Nail K Bakirov, et al. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769–2794, 2007.

Anexos

A. Algoritmos secundarios del modelo ABC-ED

Algoritmo 13 Pseudo-algoritmo ABC-ED. Funcion: obtener-vecino-candidato.
Fuente: Elaboración propia.

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

Algoritmo 14 Pseudo-algoritmo ABC-ED.
Función: hay-mas-posibles-fuentes-para-crear. Fuente: Elaboración propia.

```
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 15 Pseudo-algoritmo ABC-ED.
Función: obtener-nueva-unica-fuentecomida-selecta. Fuente: Elaboración propia.

```
1: function OBTENER-NUEVA-UNICA-FUENTECOMIDA-SELECTA( )
2:     FuenteComida f_s; rand_i ← 0; rand_j ← 0; intentos_maximos ← SN; fs_fit ← 0;
3:     for intento ← 0 to intentos_maximos do
4:         rand_i ← rand(0, r − 1); rand_j ← rand(0, c − 1);
5:         if EXISTE-POSICION-COMO-FUENTECOMIDA(rand_i, rand_j) == false ∧
6:             EXISTE-POSICION-COMO-RECHAZADA(rand_i, rand_j) == false then
7:             fs_fit ← CALCULAR-FITNESS(rand_i, rand_j);
8:             if fs_fit ≥ μ_min then
9:                 f_s ← CREAR-NUEVA-FUENTECOMIDA(rand_i, rand_j, fs_fit); // f_s ∈ NFS.
10:                return f_s;
11:            else AGREGAR-COMO-POSICION-RECHAZADA(rand_i, rand_j); // en RP.
12:            end if
13:        end if
14:    end for
15:    return null;
16: end function
```

Algoritmo 16 Pseudo-algoritmo ABC-ED. Función: Calcular-Fitness.
Fuente: Elaboración propia.

```
1: function CALCULAR-FITNESS(i, j)
2:     SUAVIZAR-PIXEL-Y-VECINDAD(IM, i, j); // usando expresión (7).
3:     // Usando expresión (1) y (3) respectivamente.
4:     CALCULAR-GRADIENTE-MAGNITUD-Y-DIRECCION(IM, i, j, Sobel);
5:     return OBTENER-GRADIENTE-MAGNITUD(IM, i, j);
6: end function
```

Algoritmo 17 Pseudo-algoritmo ABC-ED. Función: obtener-forma-de-reemplazo.
Fuente: Elaboración propia.

```
1: function OBTENER-FORMA-DE-REEMPLAZO( )
2:     forma ← 0; r ← 0;
3:     if HAY-MAS-POSIBLES-FUENTES-PARA-CREAR( ) ∧ IFS ≠ ∅ then
4:         r ← rand(1, 100);
5:         if ε ≥ r then forma ← Nueva_Exploracion;
6:         else forma ← Fuentes_Inactivas;
7:         end if
8:     else if HAY-MAS-POSIBLES-FUENTES-PARA-CREAR( ) ∧ IFS == ∅ then
9:         forma ← Nueva_Exploracion;
10:    else if HAY-MAS-POSIBLES-FUENTES-PARA-CREAR( ) == false ∧ IFS ≠ ∅ then
11:        forma ← Fuentes_Inactivas;
12:    else forma ← No_Mas_Fuentes_de_Reemplazo;
13:    end if
14:    return forma;
15: 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 para la generación de su prototipo.

Con el fin de desarrollar 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.

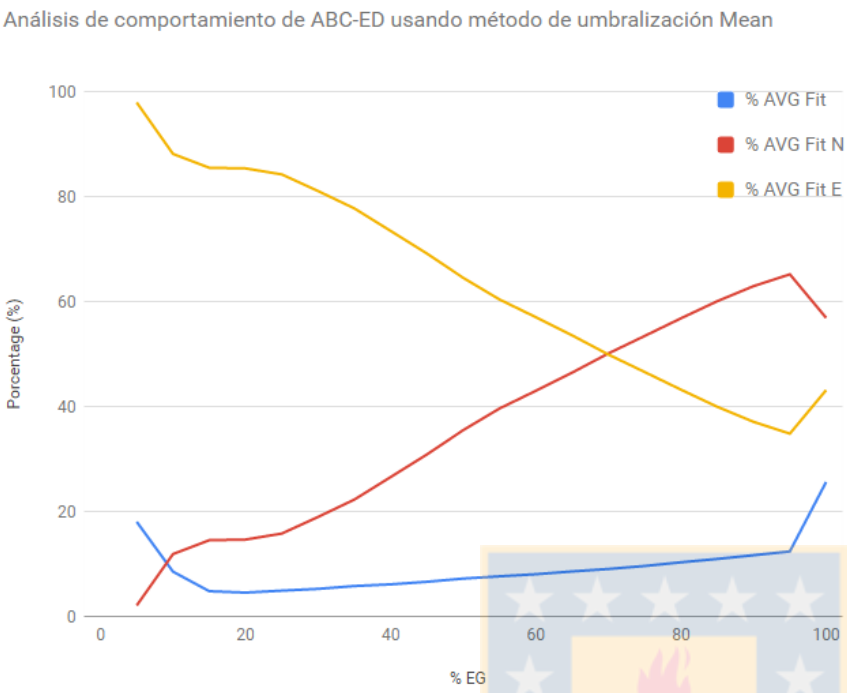
Propósitos y/o funciones de cada clase:

- **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 valido para cada imagen de entrada y salida que se procese.
- **ClassicEdgeDetection:** tiene implementado el detector de bordes de Canny con todos sus procedimientos: filtro de gauss, cálculo de gradiente magnitud y dirección, no-máxima supresión, doble umbralización e histéresis.
- **ABC_EdgeDetection:** realiza la detección de bordes con el algoritmo ABC adaptado. Posee todo lo descrito sobre el modelo ABC-ED construido. Al realizar cálculos de $fitness$, adelgazamiento de bordes, doble umbralización e histéresis se llama a la función respectiva de la clase ClassicEdgeDetection.
- **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.

- **Experiment:** realiza la experimentación explicada en sección 4.2.1. En esta clase, se usan las clases ABC_EdgeDetection, ClassicEdgeDetection, myDataHandler y myMatlabInterface y db_mysql.
- **db_mysql:** realiza la conexión, lectura y escritura de datos a la base de datos MySQL por medio de MySQL Connector/C++.

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 cada salida de suavizado gaussiano, magnitud del gradiente, dirección del gradiente, no-máxima supresión y doble umbralización; booleano (*bool*) para histéresis.

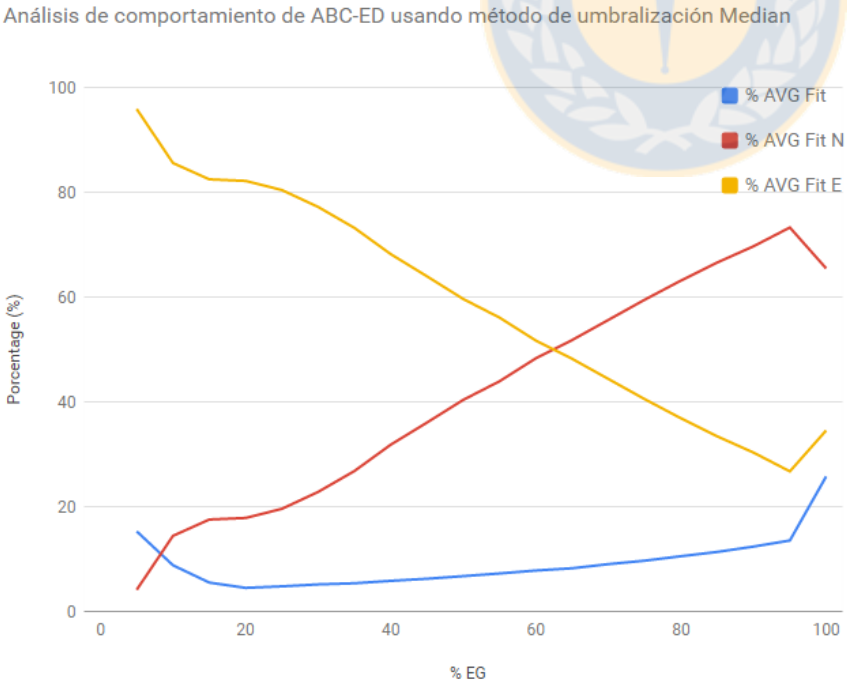
C. Graficos y Tablas secundarias de Experimentación



(a) Gráfico de análisis de comportamiento de TM Mean.

% EG	% AVG Fit	% AVG Fit N	% AVG Fit E
5	18.03	2.08	97.92
10	8.53	11.88	88.12
15	4.8	14.52	85.48
20	4.54	14.64	85.36
25	4.91	15.79	84.21
30	5.25	18.96	81.04
35	5.79	22.27	77.73
40	6.11	26.56	73.44
45	6.6	30.87	69.13
50	7.21	35.5	64.5
55	7.64	39.62	60.38
60	8.04	42.99	57.01
65	8.57	46.45	53.55
70	9.04	50.11	49.89
75	9.61	53.44	46.56
80	10.31	56.8	43.2
85	10.95	60.06	39.94
90	11.66	62.93	37.07
95	12.39	65.18	34.82
100	25.62	56.88	43.12

(b) Tabla de Figura 20a.

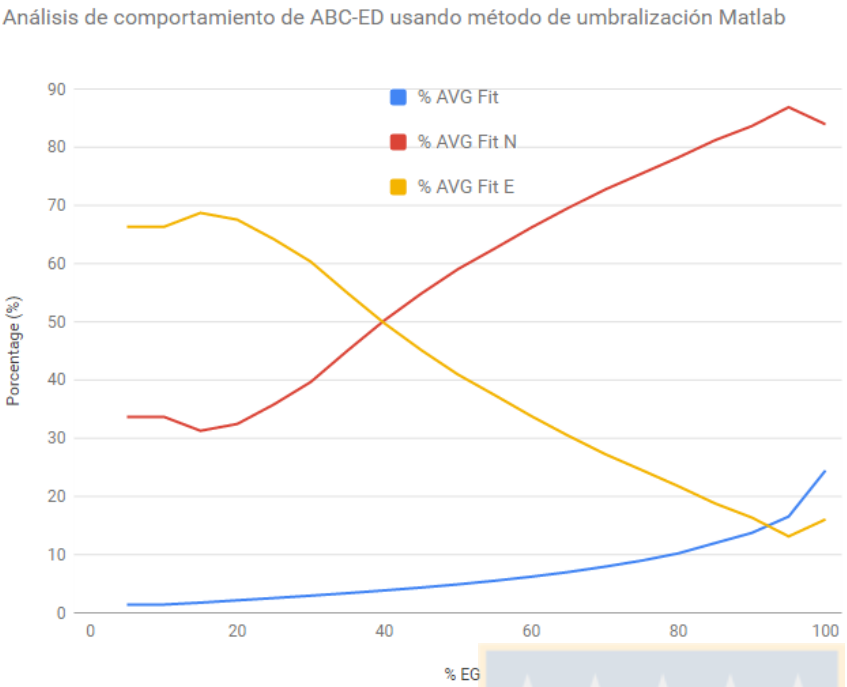


(c) Gráfico de análisis de comportamiento de TM Mean.

% EG	% AVG Fit	% AVG Fit N	% AVG Fit E
5	15.29	4.13	95.88
10	8.81	14.43	85.57
15	5.51	17.54	82.46
20	4.5	17.84	82.16
25	4.79	19.58	80.42
30	5.16	22.81	77.19
35	5.38	26.8	73.19
40	5.84	31.82	68.19
45	6.24	36.05	63.95
50	6.74	40.38	59.62
55	7.25	43.93	56.07
60	7.81	48.31	51.69
65	8.24	51.79	48.21
70	9.03	55.64	44.36
75	9.69	59.49	40.51
80	10.54	63.15	36.85
85	11.36	66.59	33.41
90	12.38	69.69	30.31
95	13.54	73.27	26.73
100	25.76	65.46	34.54

(d) Tabla de Figura 20a.

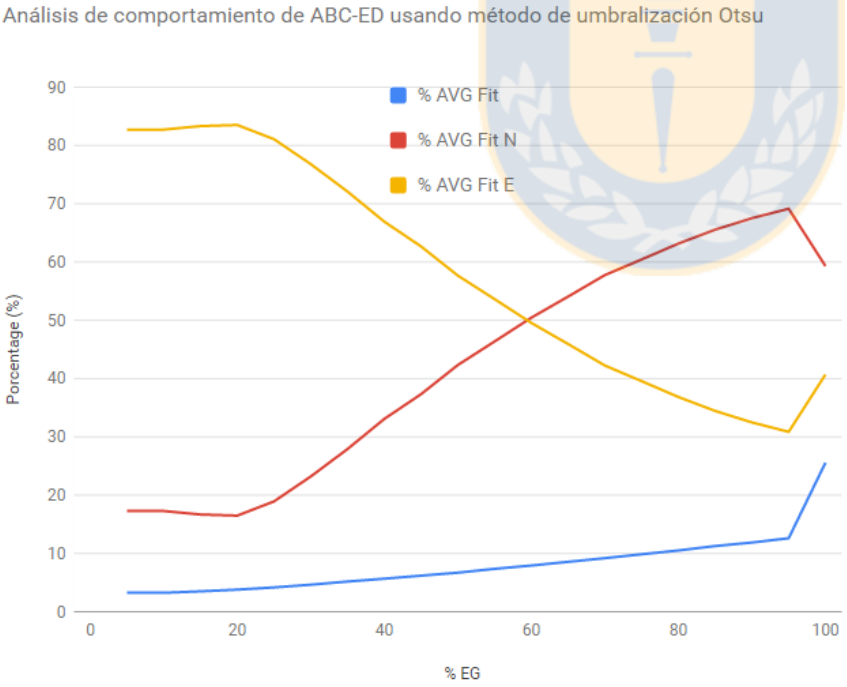
Figura 20: Análisis de comportamiento de ABC-ED para TM Mean y Median. Fuente: Elaboración propia.



(a) Gráfico de análisis de comportamiento de TM Matlab.

% EG	% AVG Fit	% AVG Fit N	% AVG Fit E
5	1.42	33.69	66.31
10	1.42	33.69	66.31
15	1.75	31.28	68.72
20	2.15	32.45	67.55
25	2.54	35.82	64.18
30	2.95	39.67	60.33
35	3.38	45.06	54.94
40	3.85	50.24	49.76
45	4.34	54.83	45.17
50	4.89	59.04	40.96
55	5.51	62.58	37.42
60	6.21	66.2	33.8
65	7	69.56	30.44
70	7.93	72.71	27.29
75	8.97	75.47	24.54
80	10.22	78.25	21.75
85	11.99	81.21	18.79
90	13.74	83.64	16.36
95	16.55	86.88	13.13
100	24.45	83.94	16.05

(b) Tabla de Figura 21a.

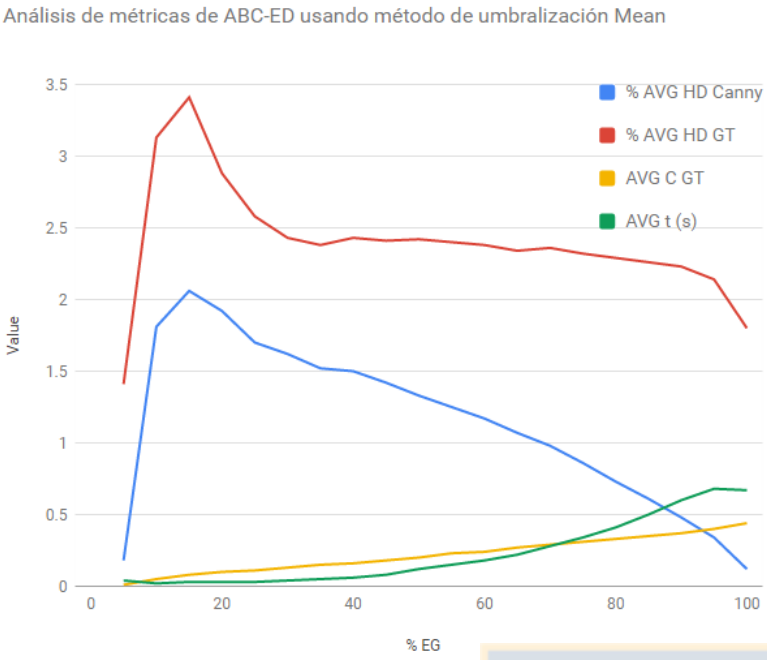


(c) Gráfico de análisis de comportamiento de TM Otsu.

% EG	% AVG Fit	% AVG Fit N	% AVG Fit E
5	3.25	17.28	82.72
10	3.25	17.28	82.72
15	3.5	16.68	83.32
20	3.79	16.49	83.51
25	4.17	18.93	81.07
30	4.64	23.21	76.79
35	5.18	27.92	72.08
40	5.67	33.08	66.92
45	6.19	37.33	62.67
50	6.7	42.33	57.67
55	7.36	46.37	53.63
60	7.93	50.45	49.55
65	8.56	54.07	45.93
70	9.19	57.76	42.24
75	9.86	60.44	39.56
80	10.51	63.16	36.84
85	11.27	65.55	34.45
90	11.87	67.52	32.48
95	12.61	69.14	30.86
100	25.56	59.31	40.69

(d) Tabla de Figura 21c.

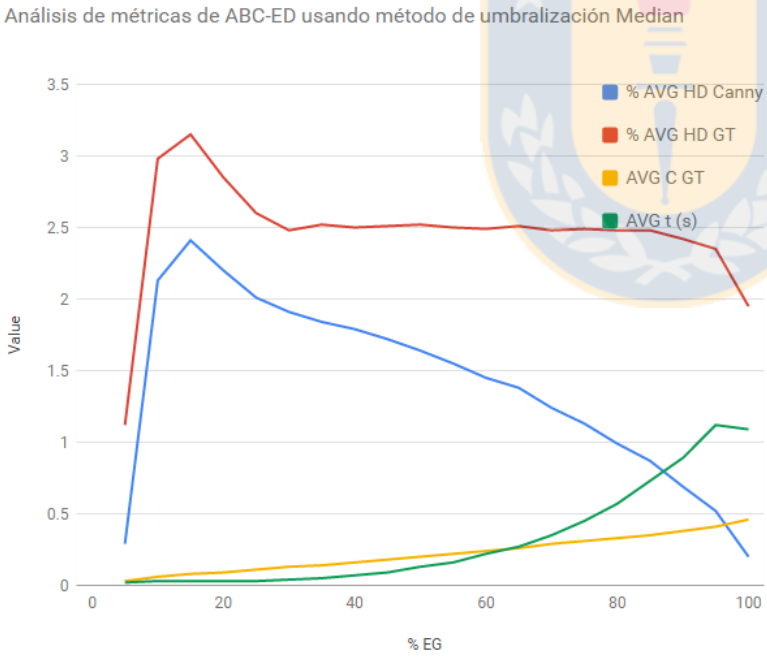
Figura 21: Análisis de comportamiento de ABC-ED para TM Matlab y Otsu. Fuente: Elaboración propia.



(a) Gráfico de análisis de métricas de TM Mean.

% EG	% AVG HD Canny	% AVG HD GT	AVG C GT	AVG t (s)
5	0.18	1.41	0.01	0.04
10	1.81	3.13	0.05	0.02
15	2.06	3.41	0.08	0.03
20	1.92	2.88	0.1	0.03
25	1.7	2.58	0.11	0.03
30	1.62	2.43	0.13	0.04
35	1.52	2.38	0.15	0.05
40	1.5	2.43	0.16	0.06
45	1.42	2.41	0.18	0.08
50	1.33	2.42	0.2	0.12
55	1.25	2.4	0.23	0.15
60	1.17	2.38	0.24	0.18
65	1.07	2.34	0.27	0.22
70	0.98	2.36	0.29	0.28
75	0.86	2.32	0.31	0.34
80	0.73	2.29	0.33	0.41
85	0.61	2.26	0.35	0.5
90	0.48	2.23	0.37	0.6
95	0.34	2.14	0.4	0.68
100	0.12	1.8	0.44	0.67

(b) Tabla de Figura 22a.

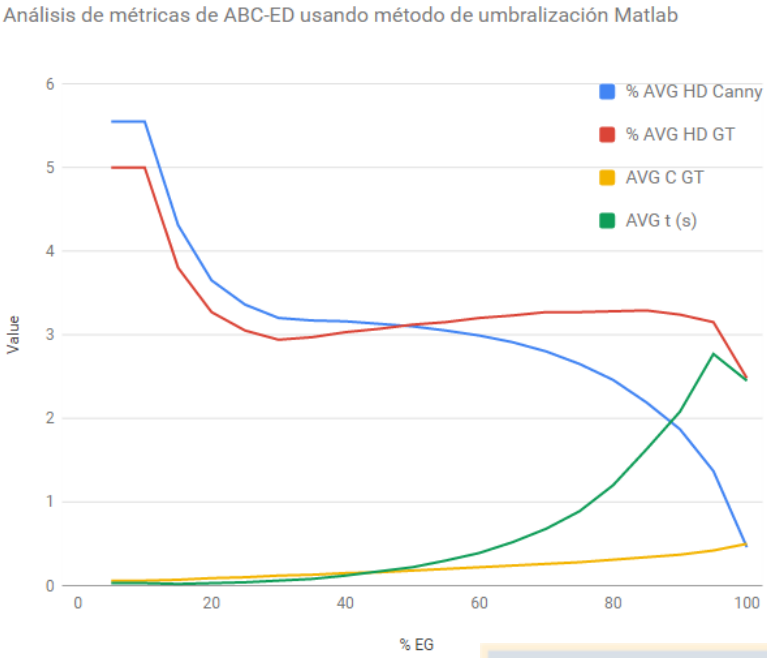


(c) Gráfico de análisis de métricas de TM Median.

% EG	% AVG HD Canny	% AVG HD GT	AVG C GT	AVG t (s)
5	0.29	1.12	0.03	0.02
10	2.13	2.98	0.06	0.03
15	2.41	3.15	0.08	0.03
20	2.2	2.85	0.09	0.03
25	2.01	2.6	0.11	0.03
30	1.91	2.48	0.13	0.04
35	1.84	2.52	0.14	0.05
40	1.79	2.5	0.16	0.07
45	1.72	2.51	0.18	0.09
50	1.64	2.52	0.2	0.13
55	1.55	2.5	0.22	0.16
60	1.45	2.49	0.24	0.22
65	1.38	2.51	0.26	0.27
70	1.24	2.48	0.29	0.35
75	1.13	2.49	0.31	0.45
80	0.99	2.48	0.33	0.57
85	0.87	2.48	0.35	0.73
90	0.69	2.42	0.38	0.89
95	0.52	2.35	0.41	1.12
100	0.2	1.95	0.46	1.09

(d) Tabla de Figura 22c.

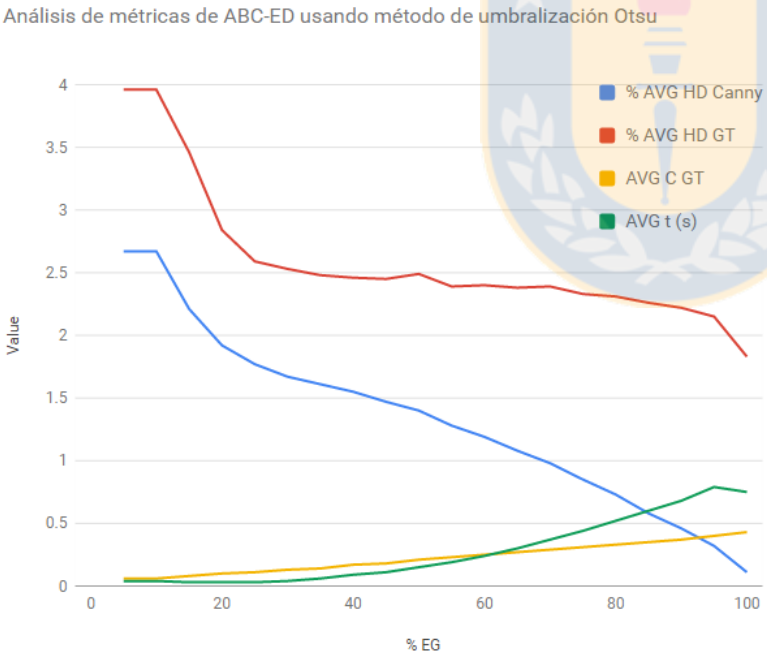
Figura 22: Análisis de métricas de ABC-ED para TM Mean y Median. Fuente: Elaboración propia.



(a) Gráfico de análisis de métricas de TM Matlab.

% EG	% AVG HD Canny	% AVG HD GT	AVG C GT	AVG t (s)
5	5.55	5	0.06	0.03
10	5.55	5	0.06	0.03
15	4.31	3.8	0.07	0.02
20	3.65	3.27	0.09	0.03
25	3.36	3.05	0.1	0.04
30	3.2	2.94	0.12	0.06
35	3.17	2.97	0.13	0.08
40	3.16	3.03	0.15	0.12
45	3.13	3.07	0.16	0.17
50	3.1	3.12	0.18	0.22
55	3.05	3.15	0.2	0.3
60	2.99	3.2	0.22	0.39
65	2.91	3.23	0.24	0.52
70	2.8	3.27	0.26	0.68
75	2.65	3.27	0.28	0.89
80	2.46	3.28	0.31	1.2
85	2.19	3.29	0.34	1.63
90	1.87	3.24	0.37	2.08
95	1.37	3.15	0.42	2.77
100	0.46	2.48	0.5	2.45

(b) Tabla de Figura 23a.



(c) Gráfico de análisis de métricas de TM Otsu.

% EG	% AVG HD Canny	% AVG HD GT	AVG C GT	AVG t (s)
5	2.67	3.96	0.06	0.04
10	2.67	3.96	0.06	0.04
15	2.21	3.46	0.08	0.03
20	1.92	2.84	0.1	0.03
25	1.77	2.59	0.11	0.03
30	1.67	2.53	0.13	0.04
35	1.61	2.48	0.14	0.06
40	1.55	2.46	0.17	0.09
45	1.47	2.45	0.18	0.11
50	1.4	2.49	0.21	0.15
55	1.28	2.39	0.23	0.19
60	1.19	2.4	0.25	0.24
65	1.08	2.38	0.27	0.3
70	0.98	2.39	0.29	0.37
75	0.85	2.33	0.31	0.44
80	0.73	2.31	0.33	0.52
85	0.58	2.26	0.35	0.6
90	0.46	2.22	0.37	0.68
95	0.32	2.15	0.4	0.79
100	0.11	1.83	0.43	0.75

(d) Tabla de Figura 23c.

Figura 23: Análisis de métricas de ABC-ED para TM Matlab y Otsu. Fuente: Elaboración propia.

D. Manual de Usuario

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

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. Ingresar el valor del parámetro umbral μ_{min} y μ_{max} y presione enter para cada uno. Se hará la detección de bordes de Canny. Al terminar, se mostrara cada imagen de salida en ventanas distintas en el siguiente orden al aplicar: filtro de gauss, gradiente magnitud, no-máxima supresión, doble umbralización e histéresis.

5. Se pregunta si desea calibrar el valor de los umbrales. Si (y) o no (n). Ingrese opción y presione enter. Si su opción fue "y", vaya al paso 4. Si su opción fue "n", se guardará en directorio de trabajo por defecto las imágenes de salida dadas por Canny.

6. Se pedirán los parámetros para ejecutar ABC-ED. Ingrese el valor para: MCN, $limit$ y ε presionando enter para cada uno. Luego, se pregunta si desea visualizar en ventana externa "v2", cada imagen de salida por cada ciclo de ejecución del modelo en su fase 1. Si (y) o no (n). Ingrese opción y presione enter.

7. Se realiza la ejecución del modelo. Luego de terminada la fase 1 del modelo, en "v2" se muestra la salida del modelo al aplicar no-máxima supresión, doble umbralización e histéresis. Al terminar la ejecución, se muestra en la terminal los resultados finales y se guarda en el directorio de trabajo por defecto cada imagen de salida del modelo.

8. 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.