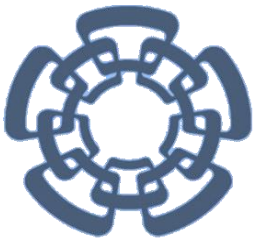


CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN DE ELECTRÓNICA DEL ESTADO SÓLIDO

"Algoritmo de Optimización Meta-heurístico Aplicado a una Red Neuronal Celular"

TESIS QUE PRESENTA:

ING. JESÚS ENRÍQUEZ GAYTÁN

PARA OBTENER EL GRADO DE MAESTRO EN CIENCIAS EN LA ESPECIALIDAD DE:

INGENIERÍA ELÉCTRICA

Directores de la Tesis:

DR. FELIPE GÓMEZ CASTAÑEDA

DR. JOSÉ ANTONIO MORENO CADENAS

México, D.F Noviembre,2015

Resumen

Los algoritmos meta-heurísticos son algoritmo de optimización de alto nivel que utilizan técnicas heuristicas para la exploración del espacio de búsqueda. Los algoritmo de optimización basados en Colonia Artificial de Abejas (ABC por sus siglas en inglés Artificial Bee Colony) y Cúmulos de Partículas (PSO por sus siglas en inglés Particle Swarm Optimization), son técnicas evolutivas inspiradas en el comportamiento social de las abejas y bancos de peces respectivamente. El uso de los algoritmos meta-heurísticos se debe a que presentan una buena alternativa para la resolución de problemas como fueron:

- Obtención de mascarillas para Redes neuronales Celulares, en procesamiento de imágenes.
- Segmentación de imágenes, mediante umbralización.
- Aproximador de funciones.
- Entrenamiento de redes neuronales.

Esto debido a que los algoritmos antes mencionados proporcionan una solución de alta calidad en un tiempo razonable. Para aprovechar estas características y poder disminuir el tiempo de procesamiento se realizó la aplicación tanto del ABC como del PSO a una tarjeta de evaluación que contiene un dispositivo FPGA, para las siguientes tareas:

- Segmentación de imágenes, mediante umbralización con PSO.
- Aproximador de funciones con ABC.

Los resultados obtenidos mediante estas tecinas de optimización meta-heurística, tanto para software como para hardware, dieron buenos resultados para las aplicaciones antes mencionadas.

Abstract

The metaheuristics algorithms are optimization algorithm high level using other techniques to exploration of the search space. The optimization algorithm based on artificial bee colony (ABC) and particle swarm (PSO), are evolutionary techniques inspired by the social behavior of bees and fish stocks respectively. The use of meta-heuristic algorithms is due to present a good alternative for the resolution of problems as they were:

- Obtaining masks Cellular Neural Networks in image processing.
- Image segmentation by thresholding.
- function approximator.

Training of neural networks. This is due to the above algorithms provide a high quality solution in a reasonable time. To take advantage of these features and to reduce application processing time both ABC PSO was performed as an evaluation board containing an FPGA device, for the following tasks:

- Image segmentation by thresholding with PSO.
- function approximator with ABC.

The results obtained by these tesinas meta-heuristic optimization for both software and hardware, gave good results for the above applications.

Agradecimientos

A las primeras personas, que les quiero agradecer son a mis asesores el Dr. Felipe Gómez Castañeda y el Dr. José Antonio Moreno Cadenas, que sin su entrega, dedicación y conocimientos, no hubiera sido posible realizar esta tesis.

"Un buen maestro nunca se olvida y su enseñanza dura toda la vida"

Agradezco a los excelentes profesores que me han permitido aprender y seguir superándome académicamente.

Agradezco al CONACYT por el apoyo económico y confianza brindad durante mis estudios de maestría.

Agradezco al CINVESTAV, a esta institución de calidad, por acogerme y brindarme todo su apoyo.

Agradezco a los miembros del jurado, el Dr. Alfredo Reyes Barranca y al Dr. Víctor Hugo Ponce Ponce, por su apoyo y por el tiempo que dedicaron para la revisión de este trabajo de tesis.

Dedicatoria.

Al creador de todas las cosas, el que me ha protegido durante todo mi camino y me ha dado la fortaleza para superar todos los obstáculos presentes en mi vida.

Dedico esta tesis a mis padres, que principalmente han sabido formarme con buenos principios, hábitos y valores, los cuales están presentes en mí y son mis pilares día a día.

Especialmente a mi padre, por haberme enseñado que nunca nos debemos de rendir y que en esta vida nadie regala nada.

Especialmente a mi madre, la cual con su ternura y apoyo incondicional me ha dado la fortaleza y seguridad para soñar, viajar, y por ello cumplir todas mis metas.

A mis hermanos que siempre han estado ahí como una luz en la obscuridad, para guiarme y ser mi compañía en todos los momentos de mi vida.

A mi novia, la cual a pesar de la distancia siempre se ha mantenido incondicional, dándome su amor y su ternura para no desfallecer en mis momentos de soledad.

A toda mis familiares y amigos que siguen estando cerca de mí, las cuales con su cariño y apoyo, me hacen crecer emocional y espiritualmente, exigiéndome siempre sacar lo mejor de mí.

Contenido

Lista de figuras

Lista de Tablas

Introducción

Día a día y constantemente se nos presentan problemas de optimización, que en si son problemas para obtener los mejores resultados, por ejemplo, el uso eficiente del tiempo, trabajo, dinero, recursos, etc. Los problemas de optimización pueden ser resueltos utilizando procesos lógicos (experiencia) y/o pequeñas fórmulas matemáticas.

El principal objetivo de los problemas de optimización, sería dar la óptima solución a una función objetivo (f(x)), esto mediante la mejor configuración de un conjunto de variables para alcanzar el objetivo planteado, pero cada vez, nos tenemos que enfrentar a un conjunto de problemas, de los cuales van en aumento su complejidad y precisión. Esto genera que los algoritmos de optimización clásicos sean insuficientes para dar solución a estos problemas en tiempos aceptables y en uso de recursos (software y hardware), por lo cual se recurre al uso de algoritmos heurísticos.

Un algoritmo heurístico es un "procedimiento simple, a menudo basado en el sentido común, que se supone que ofrecerá una buena solución (pero no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido".

El uso de algoritmos heurísticos, ha mostrado ser insuficiente para la solución de problemas complejos. Por lo cual, ha emergido una nueva clase de algoritmos de búsqueda, básicamente cambiando las características de los algoritmos heurísticos a un mayor nivel, intentando tener una mayor eficiencia y efectividad en la exploración del espacio de búsqueda.

Un algoritmo meta-heurístico, combina el prefijo griego "meta" ("más allá", aquí con el sentido de "nivel superior") y "heurístico" (heuriskein, "encontrar"). El término meta-heurística apareció, por primera vez en el artículo seminal sobre búsqueda tabú de Fred Glover en 1986. En este tipo de algoritmos, su procedimiento de búsqueda no garantiza obtener el óptimo del problema, pero si, uno muy aproximado. A diferencia de los algoritmos heurísticos, estos tratan de escapar de óptimos locales, orientando la búsqueda en cada momento a encontrar un óptimo global.

En este trabajo nos enfocaremos en algoritmos meta-heurísticos bio-inspirados; se le llama: bios-inspirados, debido a que su modelo matemático trata de simular el comportamiento de sistemas naturales.

Estos algoritmos meta-heurísticos están inspirados en sistemas biológicos en la naturaleza, como podrían ser una colonia de abejas, una parvada de aves, una colonia de hormigas, etc., que entran en la clasificación de algoritmos de inteligencia colectiva (Swarm Intelligence – SI, por sus siglas en inglés), se clasifican de esta forma debido a que cada miembro del sistema, tiene información de los demás miembros y con esta información generan un sistema sinérgico y de auto-aprendizaje, con un objetivo en común, el cuál sería por ejemplo, el encontrar la fuente de alimento más abundante. Solamente nos enfocaremos en el algoritmo de la Colonia Artificial de Abejas (Artificial Bee Colony

Algorithm - ABC, por sus siglas en inglés) y en el algoritmo de Cúmulo de Partículas (Particle Swarm Optimization – PSO, por sus siglas en inglés).

- El algoritmo de Colonia Artificial de Abejas, se basa en el comportamiento inteligente de las abejas creado por Dervis Karaboga. El ABC es un algoritmo de optimización basado en población de abejas, en donde las fuentes de alimento son reconocidas como la solución óptima del problema y el propósito de las abejas es buscar la fuente de alimento con mayor néctar para obtener la máxima eficiencia de recolección.

  Las abejas artificiales, se mueven por el espacio de búsqueda eligiendo las fuentes de alimento dependiendo de su experiencia pasada y de las referencias de sus compañeros de colmena, las cuales ajustan las posiciones de búsqueda de las abejas encargadas de explorar. El sistema se divide en tres grupos: abejas empleadas, exploradoras y observadoras.

  Las abejas exploradoras se mueven de una forma aleatoria, por todo el espacio de búsqueda y su objetivo es el encontrar las fuentes de alimento con la mayor cantidad de néctar, memorizando, las posiciones de las mejores fuentes encontradas hasta ese momento, olvidando las posiciones pasadas.

  Otro conjunto de abejas (abejas observadoras y abejas empleadas), se encarga de explotar las fuentes de alimento, en la búsqueda de mejorar la cantidad y calidad de las fuentes de alimento. Y de esta forma se combinan la exploración local y la exploración global.

- La optimización mediante cúmulo de partículas, fue desarrollada por el psicólogo-sociólogo Jammes Kennedy y por el ingeniero electrónico Russell Eberhart en 1995, basados en el comportamiento de parvada de pájaros o banco de peces, el PSO busca en un espacio de trabajo, el óptimo de una función objetivo mediante el ajuste vectorial de las partículas. El ajuste vectorial de cada partícula o individuo, se basa en un esquema de la influencia de las partículas con mayor éxito en su entorno.

  Cada partícula, tiene el potencial para encontrar el óptimo de una función objetivo, donde las partículas cambian sus velocidades y posiciones constantemente, pero permanecen a una distancia próxima una de otras dentro del cúmulo o nube de partículas, cada una de las partículas cuenta con un “fitness”, una posición y un vector de velocidad, que dirige sus movimientos, estos movimientos son apoyadas, principalmente, por las partículas con mayor éxito en el grupo o por la partícula con mayor éxito global según sea el caso.

En esta tesis, los algoritmos antes mencionados fueron utilizados en distintas tareas, como sigue:

- Búsqueda de mascarillas, para procesamiento de imágenes en redes neuronales celulares.
- Segmentación de imágenes.
- Aproximador de funciones (supervisado).
- Entrenamiento de redes neuronales como aproximador de funciones.

Pero principalmente serán usados para la obtención de mascarillas para el procesamiento de imágenes en la red neural celular (Cellular Neural Network – CNN, por sus siglas en inglés).

Una CNN, se basa en los principios de la lógica celular y se ha aplicado principalmente en las distintas ramas de procesamiento de imágenes digitales, como procesamiento morfológico, filtrado espacial, filtrado en frecuencia y análisis temporal de imágenes; son creadas por Leon O. Chua en 1987, pero fueron publicadas hasta 1988 en conjunto con L. Yang, los cuales se basaron principalmente en las redes neuronales artificiales y los autómatas celulares para formular las CNN.

Una CNN, es un circuito analógico compuesto de arreglos bidimensionales de células locales, donde cada célula o neurona es un sistema que posee una entrada, una salida y una variable de estado.

En el procesamiento de imágenes, la CNN se caracteriza por una conexión local de las células entre células vecinas, por lo cual la información se propaga por todo el sistema y esto genera que todas las células conectadas contengan parte de la información de las demás células.

Para formulación de las mascarillas de una CNN, se genera una función objetivo, partiendo de un sistema de desigualdades provenientes de las condiciones dinámicas para cada célula, donde se busca minimizar esta función, por lo cual nos apoyaremos de los algoritmos meta-heurísticos bio-inspirados, antes mencionados, para la obtención de las mascarillas adecuadas, para distintas tareas de procesamiento de imágenes en la CNN.

El algoritmo de Colonia Artificial de Abejas y algoritmo de Optimización mediante Cúmulos de Partículas serán implementados en una tarjeta de evaluación que contiene un dispositivo FPGA, para las tareas de aproximador de función y segmentación de imágenes, respectivamente.

Las siglas FPGA significan Field Programmable Gate Array. Como su nombre indica, se trata de un dispositivo compuesto por una serie de bloques lógicos (puertas, registros, memorias, flip/flops, etc) programables, es decir, la interconexión entre estos bloques lógicos y su funcionalidad no viene predefinida sino que se puede programar y reprogramar, según será necesario.

Objetivos

Objetivos generales

Obtención de mascarillas para el procesamiento de imágenes binarias para una red neuronal celular, utilizando técnicas meta-heurísticas bio-inspiradas.

Objetivo especifivcos

- Implementación algoritmo de la Colonia Artificial de abejas, para resolución de diferentes tareas de procesamiento de imágenes en una red neuronal celular.
- Diseño de una red neuronal celular básica para imágenes binarias.
- Evaluación y análisis tanto del algoritmo de la Colonia Artificial de abejas como para el algoritmo de optimización para nube de partículas con diferentes tareas.
- Implementación de una tarjeta de evaluación que contiene un dispositivo FPGA, para acelerar los tiempos de procesamiento.

Capítulo 1. Optimización con Técnicas Meta-Heurísticas.

En este capítulo se presentaran los algoritmos meta-heurísticos, como son el algoritmo de la Colonia Artificial de Abejas y algoritmo de Cúmulo de Partículas, fundamentando primero qué es un algoritmo de optimización, para dar el encuadre a los algoritmos de inteligencia colectiva a utilizar en este trabajo, como es el algoritmo de la Colonia Artificial de Abejas, ABC Algorithm y en el algoritmo de Cúmulo de Partículas, PSO Algorithm.

1.1 Algoritmo de Optimización.

El objetivo de un algoritmo de optimización, sería dar una solución adecuada y eficiente a los problemas de optimización, que cobran cada vez mayor importaría en todas las áreas de investigación e industriales. Pero un algoritmo de optimización consiste en buscar el resultado óptimo en un conjunto de soluciones posibles, para un problema en cuestión, aquellas que satisfagan de mejor manera en una serie de condiciones planteadas. La definición de "mejor" es relativa al problema que se está manejando, al método de solución y la tolerancia permitida. Al usar el término de "mejor" nos referimos a que hay más de una solución y que cada solución no tiene valores iguales. A la expresión de "mejores respuestas" se le denomina objetivo, y la función asociada se llama función objetivo.

Los problemas de optimización se pueden clasificar en dos formas:

- **Problemas de optimización combinatoria:**
  Se busca encontrar una secuencia ordenada de un conjunto de elementos, de manera que se maximice o minimice el valor de la función objetivo. Un ejemplo de éste sería algún problema de la teoría de grafos, como sería el problema de camino más corto; este problema consiste en encontrar un camino entre dos o más vértices de tal manera que la suma de los pesos de las aristas que lo conforman sea la mínima.

- **Problema de optimización numérica:**
  Se busca presentar un conjunto de variables, las cuales se sustituyen en la función objetivo de manera que su respuesta sea máxima o mínima. Un ejemplo de este problema, sería el entrenamiento de una red neuronal celular (consistente en la búsqueda de mascarillas y un umbral) En este caso, la red neuronal celular se diseña para ser aplicada en procesamiento de imágenes. Pero para cada tarea, se diseña una función objetivo donde la finalidad es encontrar el mínimo de éstas, para así poder obtener las mascarillas y su umbral, para cada tarea en específico.

En este trabajo nos enfocaremos en los problemas de optimización numérica. Estos serían definidos como el problema de búsqueda del vector x con componentes: x_1, x_2, . . . , x_n, el cual minimiza a la función objetivo f(x) .

Además, la función f(x) está sujeta a:

$$g_i(x) \leq 0, i = 1, \dots m \quad (1.1)$$
$$h_j(x) \leq 0, j = 1, \dots p \quad (1.2)$$

Donde $x \in \mathbb{R}^n$ es un vector cuyas n variables de decisión x = [x_1, x_2, … , x_n] y cada x_i, i=1,…,n está acotado por límites inferiores y superiores $L_i \leq x_i \leq U_i$. Además, de un espacio de búsqueda S y un conjunto F de todas las soluciones que satisfacen las restricciones del problema, conocidas como zona factible; m es el número de restricciones de desigualdades y p es el número de restricciones de igualdad. El problema busca minimizar la función objetivo f(x) la cual debe satisfacer las restricciones anteriores (1.1) y (1.2).

Deb [1] clasifica las técnicas usadas para resolver problemas de optimización numérica en las siguientes categorías:

1. Métodos tradicionales.

- Técnicas de variable simple: Divididas en métodos directos, los cuales utilizan la función a optimizar para guiar la búsqueda.
- Técnicas multivariable: Divididas en métodos directos y de gradiente. Realizan búsquedas en múltiples dimensiones, valiéndose en ocasiones de técnicas de variable simple.
- Técnicas para problemas con restricciones: Realizan búsquedas en espacios restringidos, usualmente utilizando técnicas multi-variable y/o de variable simple de manera iterativa.
- Técnicas especializadas: Métodos para problemas particulares como programación entera (variables que sólo toman valores enteros).

2. Métodos no tradicionales: Métodos que incorporan conceptos heurísticos para mejorar la búsqueda, por la complejidad del problema y donde la aplicación de métodos tradicionales se hacen casi imposible.

1.2 Algoritmo de Optimización Meta- Heurística.

Los algoritmos meta-heurísticos, como su nombre lo dice, quedan por encima de la heurística. Podríamos definir como heurística un método iterativo o de aproximación sucesiva, basándose en la experiencia acumulada del conocimiento del desarrollo de algoritmos en curso.

Entonces la heurística, es la capacidad de un sistema para realizar cambios o innovaciones positivas para sus fines.

Entre las diferencias entre meta-heurística y heurística, sería que la heurística es un algoritmo muy rápido que se ajusta de manera fiel a un problema de optimización, pero es difícil el definirlo en algunos problemas, lo que causa que las soluciones proporcionadas puedan no ser las óptimas. Por el contrario los algoritmos meta-heurísticos suelen ser fáciles de acoplar a los problemas de optimización, por lo que suelen ofrecer resultados justos o acertados a la realidad y en tiempos cortos. Los algoritmos heurísticos suelen no ser confiables, esto debido a que presentan estancamiento en mínimos locales (mínimo relativo) y no les es posible escapar de éstos; un mínimo local está dado por un valor menor que los valores de la función en puntos cercanos (Figura 1. 1), pero que no es el menor de todos los valores que sería el mínimo global (mínimo absoluto).

Los algoritmos meta-heurísticos presentan, un funcionamiento de búsqueda más homogénea sobre todo el espacio de búsqueda, lo que los mantiene en la expectativa de encontrar el mínimo global.

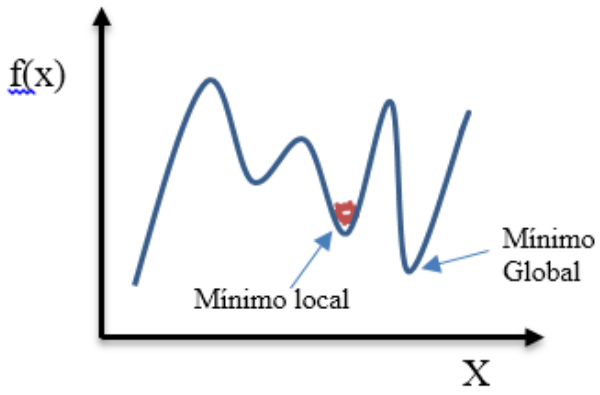


Figura 1. 1 Diagrama de una función objetivó f(x), señalando mínimo local y mínimo global.

Entre los diferentes algoritmos meta-heurísticos, se encuentran los bio-inspirados, los cuales tiene comportamientos colectivos presentes en ecosistemas de la naturaleza, generalmente de carácter descentralizado y auto-organizativo de los cuales se hablará más adelante [2].

Criterio para usar algoritmo meta-heurístico:

- Número posible de soluciones demasiado grande.
- La función de evaluación que describe la calidad de la solución varía con respecto al tiempo o tiene ruido.
- Las soluciones posibles están altamente restringidas, lo cual dificulta encontrar alguna solución factible.

1.2.1 Intensificación y diversificación de los algoritmos Meta-heurísticos.

Las piedras angulares para cualquier algoritmo meta-heurístico serían, la intensificación y la diversificación [3].

Intensificación es llamada la explotación, la cual se encarga de obtener la mejor solución local. Diversificación es llamada exploración, que tiene que ver con la intensidad de exploración del espacio de búsqueda y genera nuevas soluciones.

Una cantidad grande de diversificación, genera una convergencia rápida, pero puede perder de vista el óptimo global debido a que no intensifica en la búsqueda localizada, solo realiza un trabajo superficial.

En cambio, si se aumenta la intensificación, el proceso de búsqueda se hace más fino pero esto causa que se estanque en algún mínimo local y no se encuentre el mínimo global deseado; este proceso es lento. Por lo cual, lo óptimo es encontrar un balance entre intensificación y diversificación, esto ya que un balance equitativo entre ambos no es el óptimo.

1.2.2 Técnicas de aleatorización.

El principio de funcionamiento de los algoritmos meta-heurísticos, es el iniciar sus variables de decisión de una forma aleatoria, esto para poder tener un panorama completo del espacio de búsqueda y por consiguiente influyen en los resultados a obtener.

Por ejemplo, en las condiciones iniciales se puede optar por partir desde un punto aleatorio y al actualizar los datos volvemos a aleatorizar estos mismos, lo que nos genera muy buenos resultados y es una forma sencilla para ser implementada.

Las formas más usadas para aleatorizar los datos, sería la utilización de algoritmos de distribución probabilística, entre los cuales podemos usar los de distribución uniforme,

distribución gaussiana y distribución de Lévy; la finalidad de utilizar datos aleatorios es para poder trabajar en la búsqueda del valor mínimo global [3] [4]. También nos permite poder mantener una visión general de todo el espacio de búsqueda, así evitar el estancarnos en mínimos locales. Pero existe la discusión de que si este proceso no hace más ardua la búsqueda, por no mantener un proceso consecutivo de presentación de los datos, lo que generaría una lenta convergencia. Por lo contrario, debido a los algoritmos meta-heurísticos y sus técnicas de exploración, éstos facilitan y aumentan la probabilidad de una buena convergencia [3].

1.2.3 Características de los algoritmos meta-heurísticos.

Los algoritmos meta-heurísticos, en comparación de los algoritmos de optimización clásicos, cuentan con otra lógica y configuración, como son:

- La población inicia aleatoriamente y progresa buscando la mejor solución posible de una forma iterativa.
- Trabajan a ciegas, por lo cual no saben si llegaron a la solución óptima. Debido a esto, se les debe detener después de varios ciclos o iteraciones.
- Son algoritmos aproximativos y, por lo tanto, no garantizan la obtención de la solución óptima.
- Son relativamente sencillos; todo lo que se necesita es una representación del espacio de búsqueda, un conjunto de soluciones posibles y un mecanismo para explorar el campo de búsqueda.
- Son generales. Prácticamente se pueden aplicar en la resolución de cualquier problema de optimización de carácter numérico.
- La regla de selección del óptimo de la función objetivo depende del instante del proceso y de la historia hasta ese momento.

La forma de trabajar de los algoritmos meta-heurísticos será de la siguiente manera [4] [5]:

- Se genera un conjunto de soluciones al problema. De una forma aleatoria y limitada por las condiciones del problema.
- Se evalúa la función objetivo con las soluciones generadas.
- Se seleccionan las mejores soluciones de la población, con base en su valor en la función objetivo.
- Se generan nuevas soluciones a partir de las mejores soluciones.
- Se evalúan las nuevas soluciones.
- Se escoge la solución más óptima y que se usara de base para la siguiente iteración.
- Se repiten los procesos hasta llegar al número de iteraciones establecidas.

1.3 Algoritmo de Optimización Meta-heurísticos basados en Inteligencia Colectiva (Swarm Intelligence).

Los algoritmos de inteligencia colectiva, se basan en comportamientos sociales y colectivos identificados en la naturaleza, generalmente de carácter descentralizado y auto-organizativo. Como un ejemplo de éstos, está la organización de las abejas para la búsqueda de fuentes de alimento o la dinámica de una parvada de aves buscando alimento en un campo. El algoritmo de inteligencia colectiva trata de emular el comportamiento de sistemas sociales biológicos presentes en el medio natural, esto debido a que estos sistemas trabajan de una forma inteligente y sustentable, para la supervivencia ejecutando tareas de adquisición de alimento, manteniendo el ahorro de tiempo y con el menor desgaste de cada miembro del sistema. El comportamiento de cada miembro variará según las circunstancias, por ejemplo: en el caso de la decisión del área explorar.

También, cada miembro del sistema debe mantener una distancia mínima entre cada miembro y evitar así ser aislados; cada miembro propone un óptimo, pero solo un conjunto de los mejores se guardarán y se utilizarán para las iteraciones posteriores [6], de los cuales se obtendrá el óptimo global.

Entre las principales características de los algoritmos meta-heurísticos de inteligencia colectiva se tiene que cada miembro comunicará su información hacia los demás miembros o vecinos, lo que generará, un trabajo sinérgico inteligente: Cada uno de los miembros o partículas aprenden de sus experiencias pasadas, así como también de los mejores resultados del grupo en el transcurso o evolución del algoritmo.

1.3.1 Algoritmo de Optimización mediante Cúmulos de Partículas (PSO).

Optimización mediante Cúmulo de Partículas o mejor conocido como PSO por sus siglas en inglés (Particle Swarm Optimization), fue desarrollado por el psicólogo-sociólogo Jammes Kennedy y por el ingeniero electrónico Russell Eberhart en 1995, y la cual está basada en el comportamiento de vuelo de las parvadas de pájaros o en el movimiento de los bancos de peces (Figura 1. 2) [7][8]. Este algoritmo busca en un espacio de trabajo, el óptimo de una función objetivo (función de ajuste ó “fitness”) mediante el ajuste cartesiano de las partículas; éste se caracteriza por trabajar con un conjunto de soluciones (población) en cada iteración y en donde el resultado tiene que ver con la evolución y manipulación de la población.

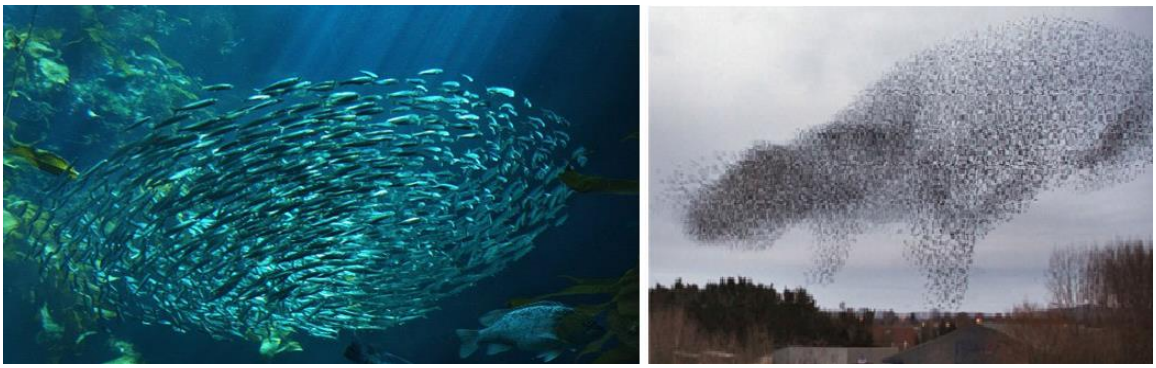


Figura 1. 2 Ejemplo de banco de peces y parvada de aves.

Los miembros de dichos cúmulos cuentan con un “conjunto de creencias” (espacio de búsqueda), donde cada miembro cuenta con un criterio propio de búsqueda y una influencia de creencia de los miembros con mejores resultados. Esto se puede ver de la siguiente manera. Para cada miembro de cúmulo:

- Su propio conocimiento sobre el entorno (valor de “fitness”).
- Su conocimiento histórico o experiencia anterior (su memoria).
- El conocimiento de experiencias anteriores de los miembros vecinos, y con mejores resultados.

El PSO es, un algoritmo donde cada miembro tiene un potencial para dar solución a la función objetivo, estos se mueven en consecuencia de su experiencia y el impacto de los demás miembros (”si son los mejores del grupo se lo comunican a los demás miembros de este grupo”), cada miembro está conformado por un la función de evaluación o de "aptitud" (fitness), una posición y un vector de velocidad que dirige sus movimientos.

1.3.1.1 Descripción del Algoritmo de Optimización mediante Cúmulos de Partículas.

El algoritmo de PSO se estructura mediante un conjunto de miembros o partículas, las cuales vuelan por el conjunto de creencia multidimensional, donde cada partícula representa una solución al problema, estas se ajustan según sus experiencias y a la experiencia de un líder.

En su recorrido, las partículas evolucionan teniendo en cuenta la mejor solución encontrada en su recorrido y al líder. El líder puede ser visto como la mejor solución entre todas las partículas durante la evolución del PSO.

Antes se mencionó que PSO era un algoritmo iterativo, por lo cual en cada iteración las partículas modifican sus velocidades tomando en cuenta su experiencia y la información del líder, esto para optimizar su búsqueda y sobre todo es un proceso estocástico. La terminología a usar está definida en la tabla 1.1.

Terminología	Descripción
$X_i = (X_{i1}, X_{i2}, X_{i3}, \dots, X_{in})$	Vector de la posición de la i-ésima partícula en el espacio de búsqueda.
$pBest_i = (p_{i1}, p_{i2}, p_{i3}, \dots, p_{in})$	Vector con la mejor posición de la partícula hasta el momento.
$V_i = (V_{i1}, V_{i2}, V_{i3}, \dots, V_{in})$	Vector de velocidad de cada partícula (gradiente de dirección en la cual se moverá la partícula).
Fitness_Xi	Valor de solución actual, mediante el vector Xi en la función objetivo.
V_i^k	Velocidad de la partícula i en la iteración k.
w	Factor de inercia, el cual debe mantenerse entre 0.9 y 1.2. Valores altos provocan una búsqueda exhaustiva (mayor diversificación) y valores bajos una búsqueda más localizada (mayor intensificación). $w = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}} * iter$
X_i^k	Posición de la partícula i en la iteración k.
g_i	Representa la posición de la partícula con mejor fitness del entorno o del cúmulo.
Φ1	Componente cognoscitiva particular.
Φ2	Componente cognoscitivo social.
Vmax	Valor máximo de velocidad que puede alcanzar la partícula.
-Vmax	Valor mínimo de velocidad que puede alcanzar la partícula (no es conveniente fijarlo en cero, no se obtienen buenos resultados).

Tabla 1.1 Definición de términos para el PSO

Una descripción breve del algoritmo PSO en su forma más sencilla seria:

Primero inicializar la nube de partículas (Figura 1. 3), esto generando un vector de posición (X_i) y de velocidad (V_i), de una forma aleatoria (quizá con ayuda de un heurístico de construcción como es (1.3) y (1.4).

Vector de posición:

$$X_i = (X_1, X_2, X_3, \dots, X_n)$$

Donde, n representa el número total de variables a optimizar y este vector será por cada uno de los miembros del cúmulo, esto es igual para las velocidades.

$$X_{1:n} = Xmin + rand(Xmax - Xmin) \quad (1.\ 3)$$

Vector de velocidad:

$$V_i = (V_1, V_2, V_3, \dots, V_{\boldsymbol{n}})$$

Donde

$$V_{1:n} = Xmin + rand(Xmax - Xmin) \quad (1.\ 4)$$

El término rand será un número aleatorio que va de 0 a 1.

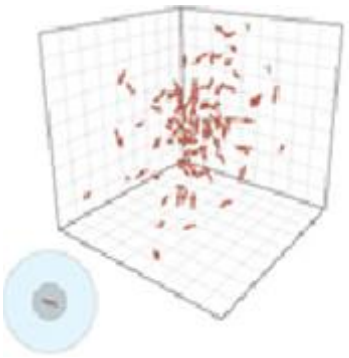


Figura 1. 3 Inicialización de las posiciones de las partículas en el espacio de búsqueda.

Una vez generadas las posiciones se calcula el Fitness_Xi de cada una de las partículas, y se actualiza $pBest_i$ con el vector inicial, esto únicamente en las condiciones iniciales.

Al iniciar las iteraciones y al actualizar la posición de la partícula, se vuelve a calcular el Fitness_Xi, y si el nuevo fitnesss es mejor al fitness anterior, se actualizan los valores de mejor posición en $pBest_i$, con los nuevos valores.

Después, el vector de velocidad (1. 5) de cada partícula es modificado en cada iteración, por:

- Su valor anterior de velocidad.
- Una componente cognoscitivo (experiencia propia de la partícula) modelado por $\Phi 1 * \text{rand}1\left(pBest_i + X_i^k\right)$, la cual representa la distancia entre la posición actual y la mejor conocida por esa partícula.
- Un componente social (experiencia compartida de la partícula como mejor resultado) modelado por $\Phi 2 * \text{rand}2(g_i - X_i^k)$, la que representa la distancia entre la posición actual y la mejor posición del vecindario.

$$V_i^{k+1} = W * V_i^k + \Phi 1 * \text{rand}1\left(pBest_i + X_i^k\right) + \Phi 2 * \text{rand}2(g_i - X_i^k) \quad (1.\ 5)$$

$$X_i = X_i + V_i \tag{1. 6}$$

Donde la ecuación (1. 6) genera una nueva dirección (Figura 1. 4), por lo cual se mantiene una búsqueda dinámica y heurística, debido a que siempre se está buscando en un espacio más satisfactorio para la solución de la función objetivo.

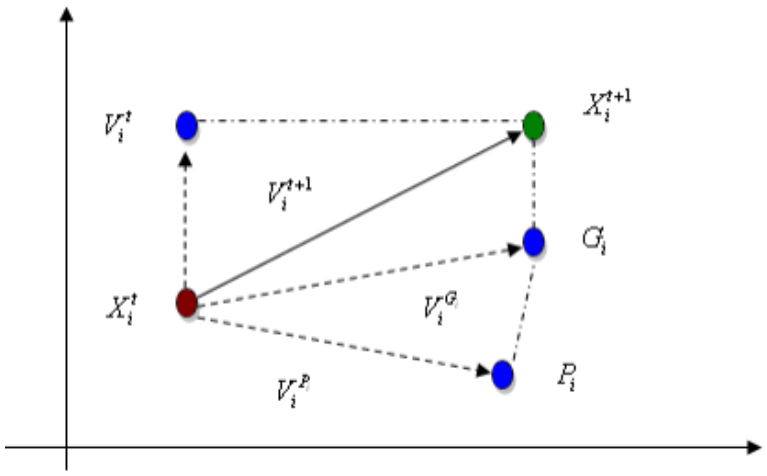


Figura 1. 4 Representación gráfica del movimiento de la partícula.

Después de generar las nuevas posiciones de las partículas, se le suma uno al número de iteraciones y se analiza nuevamente la función de fitness, esto se realizará hasta cumplir el número de iteraciones establecidas. En el siguiente subtema, se presentarán más formalmente los algoritmos de PSO.

1.3.1.2 Tipos de algoritmos PSO.

Existen diferentes tipos de PSO según la importancia que se le asigne a la parte cognitiva o a la parte social, así como el tipo de vecindario utilizado.

Los valores Φ1 representan a la parte cognitiva de la partícula en cuestión y el valor de Φ2 representa la parte social, los cuales influyen en la dirección de la partícula en cada iteración mediante la velocidad (1. 6); el algoritmo puede ser visto de 4 formas [7]:

1. Modelo completo.

En este modelo tanto la componente cognitiva, como la social influyen en la velocidad y por consiguiente en la dirección de la partícula.

$$\Phi 1, \Phi 2 > 0$$

2. Modelo cognitivo.

Únicamente el movimiento de la partícula es influenciado por la componente cognitiva.

$$\Phi 1 > 0 \; y \; \Phi 2 = 0$$

3. Modelo solo social.

Solo está presente en el movimiento la componente social.

$$\Phi 1 = 0 \text{ y } \Phi 2 > 0$$

4. Modelo social exclusivo.

La posición de la partícula en sí no puede ser la mejor de su entorno.

$$\Phi 1 = 0 \text{ y } \Phi 2 > 0 \; y \; g_i \neq X_i$$

Estudios realizados a la influencia de los componentes de aprendizaje (sociales y cognitivos) [8] no dicen que es recomendable usar valores Φ1= Φ2=2.

El PSO se puede dividir en dos modelos según sus vecindarios (esto sería debido a la cantidad y posición de las partículas que intervienen en el cálculo de la distancia en la componente social), PSO local y PSO global.

Algoritmo 1. 1. Pseudocódigo PSO Local

Inicializar cúmulo.
While no se alcance las iteraciones deseadas **do**
 For i=1 hasta n **do**
  Evaluar cada partícula X_i del cúmulo.
  If fitness(X_i) es mejor que fitness ($pBest_i$) **then**
   $pBest_i=X_i$;
   fitness($pBest_i$)=fitness(X_i);
  end if
 end For
 For i=1 hasta n **do**
  Escoger $lBest_i$, la partícula con mejor fitness de X_i entre el grupo de partículas.
  $\boldsymbol{V_i^{k+1} = W * V_i^k + \Phi 1 * \text{rand1}(pBest_i + X_i^k) + \Phi 2 * \text{rand2}(lBest_i - X_i^k)}$
  $\boldsymbol{X_i = X_i + V_i}$
 end For
end While

En el PSO Local, se calcula la distancia entre la posición actual de la partícula y la posición de la mejor partícula encontrada en su vecindario (algoritmo 1.1).

Algoritmo 1. 2. Pseudocódigo PSO Global.

Inicializar cúmulo.
While no se alcancé las iteraciones deseadas **do**
For i=1 hasta n **do**
Evaluar cada partícula X_i del cumulo.
If fitness(X_i) es mejor que fitness ($pBest_i$) **then**
$pBest_i=X_i$;
fitness($pBest_i$)=fitness(X_i);
end if
If fitness($pBest_i$) es mejor que fitness (gBest) **then**
gBest=pBest;
fitness(gBest)=fitness($pBest_i$);
end if
end For
For i=1 hasta n **do**
$$\boldsymbol{V_i^{k+1} = W * V_i^k + \Phi 1 * \mathbf{rand1}\left(pBest_i + X_i^k\right) + \Phi 2 * \mathbf{rand2}(\mathbf{gBest} - X_i^k)}$$
$$\boldsymbol{X_i = X_i + V_i}$$
end For
end While

A diferencia entre el PSO Local y el PSO Global, es la distancia entre la componente social, la cual viene dada por la diferencia entre la posición de la partícula actual y la posición de la mejor partícula encontrada hasta ese momento en el cúmulo ($gBest_i$).

1.3.2 Algoritmo de Optimización basado en Colonia Artificial de Abejas (ABC).

En la naturaleza podemos encontrar distintas tipos de organizaciones, jerarquías y sistemas de los cuales entre los más efectivos se encontrarían las colonias de abejas, manteniendo un sistema jerárquico funcional y sobre todo óptimo, para proveer de alimento a su colmena y mantener un sistema diferenciado en función de cada conjunto de miembros que forman el sistema.

El algoritmo de la colonia artificial de abejas es un algoritmo meta-heurístico bio-inspirado, en el comportamiento inteligente de las abejas creado por Dervis Karaboga en 2005 [9].

El ABC es un algoritmo de optimización basado en poblaciones, en el cual, la fuente de alimento es la solución óptima del problema, donde el propósito de las abejas es buscar la fuente de alimento (Figura 1. 5) con mayor néctar para obtener la mayor cantidad de alimento [10].

En la vida real, la colmena está conformada por abejas obreras y la abeja reina. Las abejas obreras tienen una sub-clasificación en las cuales su funciones son: las de recolectar néctar

de las flores, llevarlo a la colmena, realizar miel, realizar cera y defender la colmena; en cambio, la principal función de la abeja reina es la de procrear nuevas abejas para mantener o generar nueva población, siempre manteniendo un número funcional de abejas en la población para que sea sustentable el panal. Pero lo que es el tema de interés, es la forma de comunicación entre las abejas y sus diferentes funciones en la búsqueda de fuentes de alimento.

Figura 1. 5 Abeja obteniendo polen de una fuente de alimento.

Las abejas artificiales se mueven emulando el comportamiento de búsqueda de las abejas en la vida real, eligiendo las fuentes de alimento dependiendo de su experiencia pasada y de las referencias de sus compañeras de colmena, las cuales ajustan las posiciones de búsqueda esperando mejorar la cantidad y calidad del néctar de cada fuente de alimento.

El sistema se divide en tres grupos: abejas empleadas, exploradoras y observadoras. Las exploradoras se mueven de una forma aleatoria para buscar por todo el espacio de búsqueda y si encuentran una cantidad mayor de néctar, memorizan su posición y olvidan la anterior, por lo que de esta manera las abejas combinan exploraciones locales y global, ya que otro conjunto de abejas se encarga de explotar las fuentes de alimentos ya encontradas [11].

El algoritmo consiste en un conjunto de posibles soluciones "i" (fuentes de alimento); cada fuente de alimento $f(X_i)$ está representado por una posición en un espacio multidimensional $X_i \in \mathbb{R}^D, i = 1, \dots, M$, el parámetro D seria el número total de variables y M el número total de abejas.

El algoritmo se puede dividir en tres tipos de abejas, como sigue:

- Abeja exploradora.- Su principal función, es buscar por todo el espacio de búsqueda, nuevas fuentes de alimento, realizando su búsqueda de una forma aleatoria. Esta fase se utilizará para generar las condiciones iniciales (1. 7) y después de que las

abejas empleadas y observadoras no puedan mejorar la calidad de las fuentes de alimento.

$$X_i^j = X_{min}^j + rand(0,1)(X_{max}^j - X_{min}^j) \quad (1.7)$$

Donde X_{min}^j es el valor mínimo del espacio de búsqueda y X_{max}^j es el valor máximo del espacio de búsqueda.

- Abeja empleada.- Este conjunto de abejas se encarga de explotar las fuentes de alimento proporcionadas por las abejas exploradoras, a modo de mejorar la calidad del néctar de cada fuente de alimento (búsqueda del mínimo global). Éstas tienen un criterio de trabajo en el cual comparan la fuente de alimento anterior, con la fuente de alimento nueva, y si es mejor, se guarda en la memoria y se olvida la anterior.

$$V_i^j = X_i^j + \emptyset_i^j(X_i^j - X_k^j) \quad (1.8)$$

Donde $k \in \{1,2,\dots,M\}$, j$\in \{1,2,\dots,D\}$ y $k \neq i$, estos índices son elegidos aleatoriamente y $\emptyset_i^j$ es un número aleatorio que va de 0 a 1.

Después de que las abejas generan un nuevo vector (apoyada de sus variables anteriores y las posiciones de otras fuentes de búsqueda X_k^j), se vuelve a evaluar la nueva fuente de alimento y se genera un dato que proporciona la calidad del néctar mediante una función de fitness (1.9). Finalmente, las abejas empleadas comparten la información, con las abejas observadoras mediante el uso de datos probabilísticos (1.10).

$$fitness_i = \begin{cases} \dfrac{1}{1+f(X_i)} & if\ f(X_i) \geq 0 \\ 1 + abs\big(f(X_i)\big) & if\ f(X_i) < 0 \end{cases} \quad (1.9)$$

- Abeja observadora.- Este tipo de abejas analizan la información compartida por las abejas empleadas, con ello pueden apreciar la calidad de las fuentes de alimento y generar su propio criterio de explotación de la fuente de alimento. También cuentan con un criterio de abandono de fuente de alimento, con el cual después de varios ciclos si las abejas empleadas y estas mismas no lograron mejorar la calidad, las fuentes de alimento se abandonan y por medio de las abejas exploradoras se obtienen nuevas fuentes de alimento para ser explotadas.

$$p_i = \frac{fitness_i}{\sum_{n=1}^{M} fitness_n} \quad (1.10)$$

Algoritmo 1. 3 Pseudocódigo del cuerpo del Algoritmo de Colonia Artificial de Abejas (ABC).

- ~ Inicializar las variables **(Algoritmo 1. 4)**.
- ~ Evaluación

For Iter = 1 hasta IterMax **do**

- ~ Fase de Abejas Empleadas **(Algoritmo 1. 5)**.
- ~ Fase de Abejas Observadoras **(Algoritmo 1. 6)**.
- ~ Fase de Abejas Exploradoras **(Algoritmo 1. 7)**.
- ~ Memorización de las mejores soluciones.

End For

En el ABC, el número de abejas empleadas es igual al número de fuentes de alimento. El número total de abejas a utilizar se divide en 2 partes iguales, una parte corresponde al total de las abejas empleadas y la otra a las abejas observadoras, recordando que la cantidad de néctar de la fuente de alimento corresponde a la calidad (fitness) que está asociada a la solución.

El primer paso sería generar una distribución aleatoria de la población (número de fuentes de alimento - sn), las cuales están determinadas por el vector de solución $X_i(i = 1,2, \dots, D)$, donde D es la dimensión de las variables de optimización en el Algoritmo 1. 4.

Algoritmo 1. 4. Inicialización del Algoritmo de Colonia Artificial de Abejas (ABC).

For i= 1 hasta sn/2 **do**
  For j = 1 hasta D **do**
    Generación de X_i soluciones ec. 1 .7
  End For
  $Fracaso_i$ = 0 -- Contador de límite para abandono de i-fuente de alimento.
End For

Después de la inicialización, es evaluada la función objetivo, se obtiene un fitness de ésta y pasa a un ciclo iterativo, el cual limitará nuestros procesos, debido a que el algoritmo por sí solo no es capaz de saber cuándo llegó al resultado óptimo y por consiguiente hay que detenerlo.

En este ciclo iterativo, se encierran las fases de las abejas empleadas, exploradoras y observadoras, donde se inicia con la fase de las abejas empleadas en el Algoritmo 1. 5.

Algoritmo 1. 5. Fase de Abejas Empleadas del Algoritmo de Colonia Artificial de Abejas (ABC).

For i= 1 hasta sn/2 **do**
For j = 1 hasta D **do**
Generación de nuevas soluciones para una fuente de alimento mediante V_i (Ec.1.8) (1. 8), teniendo en cuenta, los índices de control antes descritos para esta fase.
Se obtiene el nuevo resultado de la fuente de alimento con la nueva solución propuesta y se genera el fitness de esta misma.
Se guarda la mejor solución entre V_i y X_i.
If V_i es mejor a X_i **Then** $Fracaso_i = 0$
Else $Fracaso_i = Fracaso_i + 1$
End If

End For
End For
Se obtiene el valor probabilístico (Ec. 1.8) de cada fuente de alimento, esta información es compartida con las abejas observadoras.

En la fase de abejas observadoras en el Algoritmo 1. 6, se vuelven a generar nuevas soluciones pero éstas condicionadas con un valor probabilístico, el que determina si se vuelven a generar estas soluciones o se quedan con las ya establecidas.

Algoritmo 1. 6. Fase de Abejas Observadoras del Algoritmo de Colonia Artificial de Abejas (ABC).

t = 0
i = 1
repeat
If random<p_i **Then**
t= t + 1
For j = 1 hasta D **do**
Se produce una nueva fuente de alimento V_i para las abejas observadoras (Ec. 1.8)
End For
Se genera el fitness para la nueva fuente de alimento y se aplica la selección del mejor fitness entre V_i y X_i. Y se guarda la mejor solución.

If V_i es mejor a X_i **Then** $Fracaso_i = 0$
Else $Fracaso_i = Fracaso_i + 1$
End If
i = i + 1
until t = sn/2

Después de la fase de abejas observadoras, se inicia la fase de las abejas exploradoras, pero éstas solo entran en acción cuando un límite ($Fracaso_i$) ha sobrepasado el número de intentos para mejorar la fuente de alimento, lo que genera que se abandone esta fuente de alimento y se proponga una nueva, esto de una forma aleatoria (Ec. 1.7), y para cada fuente de alimento que sobrepase el límite de búsqueda.

Algoritmo 1. 7. Fase de Abejas Exploradoras del Algoritmo de Colonia Artificial de Abejas (ABC).

If Max_Fracaso> $Fracaso_i$ **Then**
Remplazar X_i con un nuevo vector mediante el uso de la Ec. 1.7
$Fracaso_i = 0$
End If

1.4 Conclusiones.

En este capitúlalo se han presentado dos algoritmos meta-heurísticos, que se clasifican como algoritmos de inteligencia colectiva bio-inspirados, y como anteriormente se mencionó, se basan en comportamientos colectivos en la naturaleza, lo que genera que sean algoritmo inteligentes y con un alto grado de aprendizaje, de los cuales, ambos serán usados para resolver problemas de optimización numérica.

El uso de los algoritmo meta-heurísticos, se debe a su proceso de búsqueda, que le permite mantener una dinámica sinérgica entre cada miembro que forma, ya sea la nube de partículas o la colonia, la cual permite una convergencia al óptimo global; también hay que mencionar que pueden ser no fiables para ciertos problemas, esto debido a la complejidad del problema (lo que generará tiempos muy extensos para la resolución o datos inexactos) y/o al criterio de limitación del proceso de búsqueda (esto debido que no tiene un criterio para detenerse al encontrar los datos deseados, provocando que si el número de iteraciones es menor a las necesarios para llegar al resultado, no se obtengan los valores óptimos para la resolución del problema y si el número de iteraciones es mayor al necesario para obtener los valores óptimos, la resolución del problema tendrá tiempos innecesarios).

En sí, en este trabajo de tesis solo se hace uso del PSO y el ABC, como se encuentran diseñados por su autores [9] [12] , pero existen otras alternativas propuestas de las cuales el diseño de búsqueda, exige más tiempo, las cuales no fueron necesarios para la resolución de las tareas presentadas en este trabajo.

Cabe mencionar que estos algoritmos meta-heurísticos, se seguirán presentando en todas las áreas de investigación debido a su sencillez, versatilidad y exactitud aceptable de estos mismos.

Capítulo 2. Teoría Básica de las Redes Neuronales Celulares

(Cellular Neural Network – CNN).

En este capítulo, se presentará la fundamentación teórica de las redes neuronales celulares en la función de procesamiento de imágenes. Comenzando en presentar a los creadores de este algoritmo y cómo surgió la idea de este arreglo de redes neuronales, para dar lugar a la teoría de las redes neuronales celulares, su estructura, modelo eléctrico y condiciones iniciales, para finalizar con algunas aplicaciones de la CNN en procesamiento de imágenes binarias.

2.1 Redes Neuronales Celulares.

En el procesamiento digital de imágenes se encuentran muchos modelos de procesamiento, entre ellos se encuentran las Redes Neuronales Celulares, mejor conocidas como CNN (Cellular Neural Network), donde la finalidad es la búsqueda de información en la imagen, como sería la eliminación de ruido o detección de bordes, etc.

El sistema CNN, se basa en los principios celulares donde cada célula es un subsistema que se conforma por una entrada, una salida y una variable de estado representada por ecuaciones diferenciales ordinarias representativas de la dinámica de la CNN.

La CNN es propuesta por Leon O. Chua y Lin Yanf en 1988 [13]. Las redes neuronales celulares se "estructuran" de las redes neurales artificiales y las autómatas celulares. De las redes neuronales artificiales, por su funcionamiento asíncrono paralelo, y de las autómatas celulares, por su estructura (por su forma de distribuir los elementos). En trabajos de investigación interdisciplinarios se encontró que la CNN es capaz de simular varias partes del sistema nervioso de la retina humana y la estructura de la corteza cerebral en primates, incluyendo al del ser humano [14].

Entre sus principales aplicaciones encontramos que es ampliamente usada para el reconocimiento de patrones, procesamiento de imágenes y video, procesamiento no lineal en general y solución de ecuaciones diferenciales, entre otras.

Las redes neuronales artificiales están controladas por un conjunto de mascarillas, las que determinan el funcionamiento de la CNN; existen muchas formas para generarlas, de las cuales se estará hablando en el próximo capítulo.

Se podría definir una red CNN como un circuito no lineal dinámico compuesto por unidades recurrentes, llamada celdas, que se interconectan localmente en el espacio, estructurando diferentes topologías como: Rectangular, hexagonal, toroide, esférica, etc.

Y que pueden ser definidas matemáticamente según las siguientes características.

- Dinámica de celda.
- Ley sináptica.
- Condiciones de contorno.
- Condiciones iniciales.

2.2 Arquitectura estándar de la CNN

Existen tres modelos de CNN que son los más usados: las CNN de una sola capa, la CNN multicapa y las CNN en tiempo discreto y continuo. El primer modelo es el más común y en el que nos enfocaremos.

La unidad básica de una red neuronal celular es llamada celda, ver Figura 2. 1. Cada célula (celda) en la red celular está conectada a sus vecinas aledañas, que al igual que cada celda, solo puede interactuar con otras celdas contiguas y sobre las cuales ésta ejerce su influencia de forma directa, pero todas las celdas están en comunicación debido a efecto de propagación de la dinámica de tiempo continuo de la CNN [13] . Este vecindario está definido por lo general mediante una esfera, centrada en la propia celda a analizar, cuyo radio r especifica el tamaño de la vecindad (Figura 2. 2).

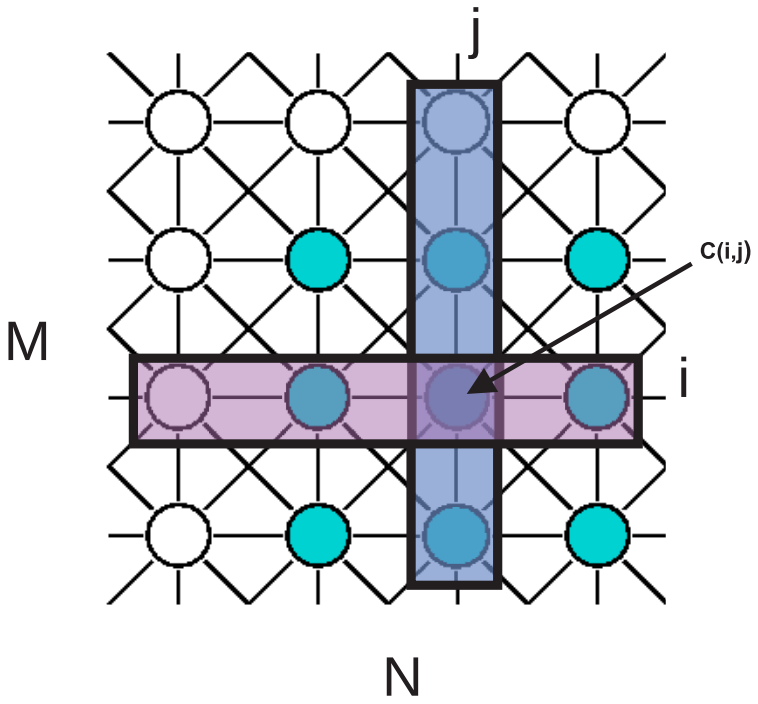


Figura 2. 1 CNN 4x4 Celda C (i, j).

Una red neuronal celular bidimensional de MxN, se entiende que tiene MN celdas en M filas y N columnas, distribuidas uniformemente sobre un plano. La celda (i,j) será denotada por

C(i,j), llamando a la celda de la i-ésima fila y j-ésima columna (siendo i =1,2,…,M y j=1,2,…,N). La CNN puede ser usada para realizar procesamiento de imágenes en forma masiva y paralela, asociando cada pixel de la imagen a una celda de la CNN, esto debido a su forma rectangular, lo que permite ser asociado fácilmente pixel como celda.

Cada celda C(i,j) interactúa con otras celdas que están en el radio de influencia o "vecindario" Nr(i,j). El vecindario-r de una celda C(i,j) en una red neuronal celular está definido por:

$$N_r(i,j) = \{C(k,l) | \max\{|\mathrm{k}-\mathrm{i}|, |\mathrm{l}-\mathrm{j}|\} < r, 1 \leq k \leq M; 1 \leq l \leq N\}$$

Donde r es entero positivo y es llamado radio de vecindario o esfera de influencia. Pero el arreglo neuronal puede tomar otras topologías; en la Figura 2. 2 se muestra una topología rectangular de orden 1, esto para un vecindario de radio 1 (sombreado) de una celda C(i,j). Donde cada círculo representa una neurona (celda) y las líneas indican las conexiones bidireccionales.

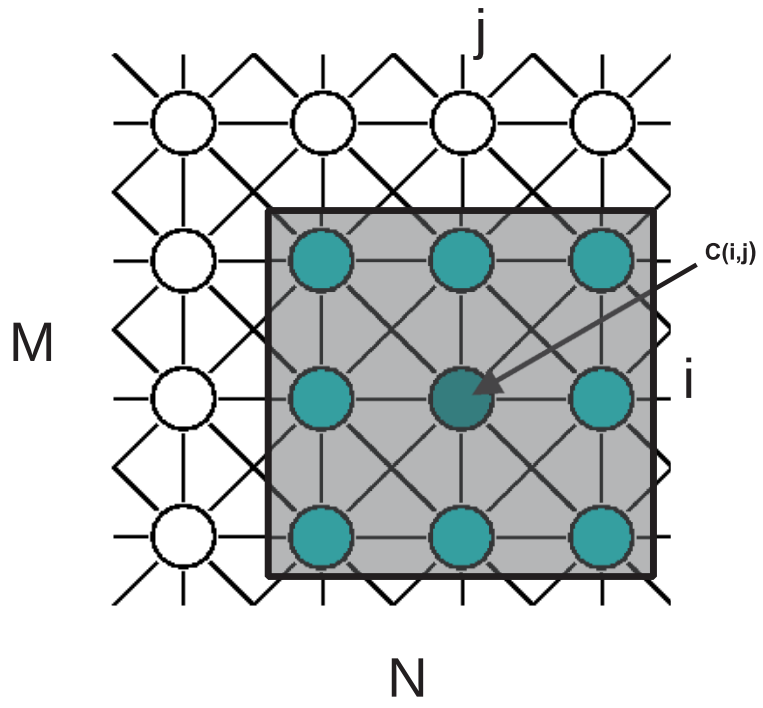


Figura 2. 2 CNN celda C(i,j) y vecindario de radio 1 (sombreado).

Las redes CNN pueden presentar diversas topologías y conectividades diferentes (Figura 2. 3). En la práctica, lo habitual es que las redes CNN formen estructuras bidimensionales de una sola capa y radio de vecindad reducido.

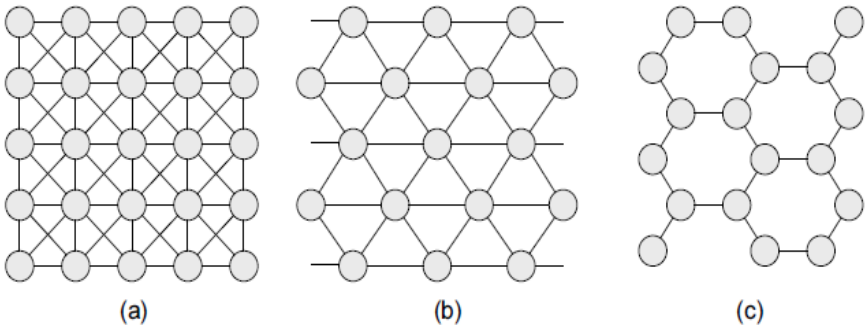


Figura 2. 3 Ejemplo de CNN con distintas topologías. a) Rectangular, b) Triangular y c) Hexagonal.

2.2.1 Estructura de la celda

Una celda está conformada por un número de entradas (valores de las celdas vecinas), las cuales están ponderadas por un factor. Donde el conjunto de estos forman la mascarilla o plantillas que determinan la intensidad de conexión sinápticas entre las celdas y la propagación de la información, de forma semejante como ocurre en las redes neuronales biológicas. Las señales son sumadas e integradas en un modelo de tiempo continuo. El resultado corresponde a la celda $X_{i,j}$ la cual pasa a una función de trasferencia $f(x_{ij})$ la que proporciona el valor de salida de la celda $y_{i,j}$. Se puede observar el diagrama de una celda de una CNN en la Figura 2. 4.

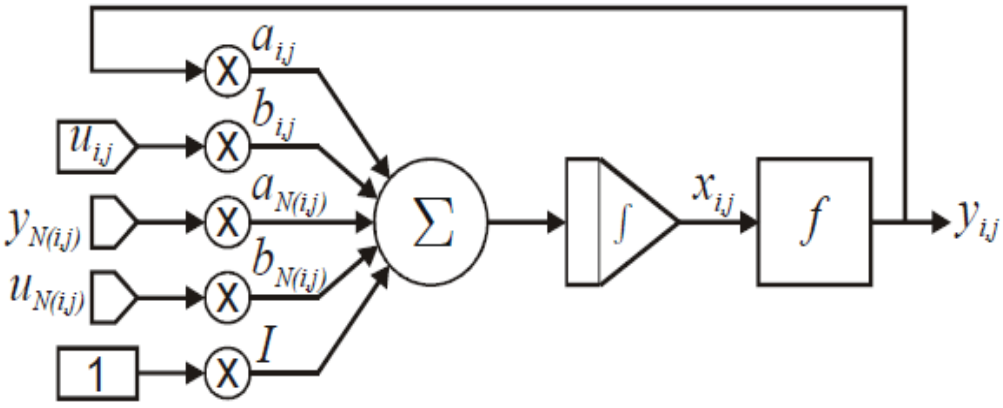


Figura 2. 4 *Diagrama a bloques de una celda CNN.*

2.2.2 Ecuaciones de una CNN

El éxito del modelo de la ecuación diferencial original de Chua y Yang [13] radica en la simplicidad de ésta, ya que se conforma de un arreglo celular que está descrito por un conjunto de ecuaciones diferenciales ordinarias, donde cada neurona interna C(i,j) estará definida por la ecuación de estado siguiente:

$$C\frac{dv_{xij}(t)}{dt} = -\frac{v_{xij}(t)}{R_x} + \sum_{C(K,L)\in Nr(i,j)} A(i,j;k,l)v_{ykl}(t) + \sum_{C(K,L)\in Nr(i,j)} B(i,j;k,l)v_{ukl} + I_{ij} \quad (2.1)$$

Donde:

$$1 \le i \le M; 1 \le j \le N.$$

Haciendo C = 1 y R_x = 1, obtenemos la ecuación de estado normalizada:

$$\frac{dv_{xij}(t)}{dt} = -v_{xij}(t) + \sum_{C(k,l)\in Nr(i,j)} A(i,j;k,l)v_{ykl}(t) + \sum_{C(k,l)\in Nr(i,j)} B(i,j;k,l)v_{ukl} + I_{ij} \quad (2.2)$$

Donde:

- V_{ukl}, v_{xij}, v_{ykl} son las variables de entrada, estado y salida de C (i, j) respectivamente.
- Se asume que v_{ukl} tiene una magnitud constante.
- A y B son 2 matrices del tamaño de la vecindad definida. La matriz A se denomina también matriz de retroalimentación, debido a que ejerce una función de control al retroalimentar la salida; a la matriz B se le conoce como matriz de control, más un bias (I_{ij}) el cual ajusta la salida.

La matriz A(i,j : k,l), que representa la mascarilla de retroalimentación, y la matriz B(i,j : k,l), que sería la mascarilla de control en conjunto con el umbral, serán las encargadas de definir las diferentes tareas de procesamiento de las imágenes de entrada en la CNN.

Las matrices de retroalimentación (A), de control (B) y el umbral (*I*) serán representadas de la siguiente manera, esto para una CNN rectangular de radio de vecindario 1 y considerando parámetros espaciales y temporales invariables, siendo este el caso más común.

$$A = \begin{bmatrix} a_{i-1,j-1} & a_{i-1,j} & a_{i-1,j+1} \\ a_{i,j-1} & a_{i,j} & a_{i,j+1} \\ a_{i+1,j-1} & a_{i+1,j} & a_{i+1,j+1} \end{bmatrix}, \mathrm{B} = \begin{bmatrix} b_{i-1,j-1} & b_{i-1,j} & b_{i-1,j+1} \\ b_{i,j-1} & b_{i,j} & b_{i,j+1} \\ b_{i+1,j-1} & b_{i+1,j} & b_{i+1,j+1} \end{bmatrix} \text{ e } I_{ij}$$

Observando la Ec. 2. 2, cada elemento $a_{k,l}$ de la matriz A se multiplica por la salida y_{kl} de la neurona $C_{k,j}$. De igual manera cada elemento de $b_{k,l}$ de la matriz b se multiplica por la entrada $y_{k,l}$ de la neurona $C_{k,l}$..

La función de trasferencia $f(x_{ij})$ está representada por elementos no lineales en cada celda $x_{l,J}$ y nos produce la salida $y_{i,j}$. Está definida por:

$$f(x_{ij}) = \frac{1}{2}|x_{ij} + 1| - \frac{1}{2}|x_{ij} - 1|$$

Por lo que la ecuación de salida de la celda se establece como:

$$v_{yij} = f(x_{ij}(t)) = \tfrac{1}{2}|x_{ij}(t) + 1| - \tfrac{1}{2}|x_{ij}(t) - 1| \quad \begin{cases} 1, si\ V_{yIJ} \geq 1 \\ V_{yIJ}, si - 1 < V_{yIJ} < 1 \\ -1, si\ V_{yIJ} \leq -1 \end{cases} \qquad (2.\ 3)$$

Se puede observar en la Figura 2. 5, el esquema de la función de trasferencia, donde v_{yij}=-1 corresponderá a un pixel blanco y v_{yij}=1 corresponderá a una pixel negro.

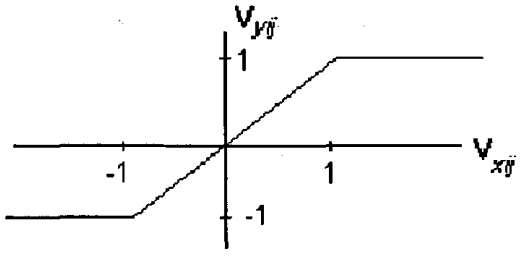


Figura 2. 5 Función de trasferencia $f(x_{ij})$, no lineal.

2.2.3 Condiciones iniciales, neuronas estándar y de frontera.

La ecuación de estado (Ec. 2. 2), nos pide una retroalimentación de su salida (caso inicial será la misma que la entrada) y un vecindario completo para iniciar el proceso que evolucionará con el tiempo y por lo largo de las celdas de la red, donde las condiciones iniciales tienen que ser muy claras $V_{ij}(0)$.

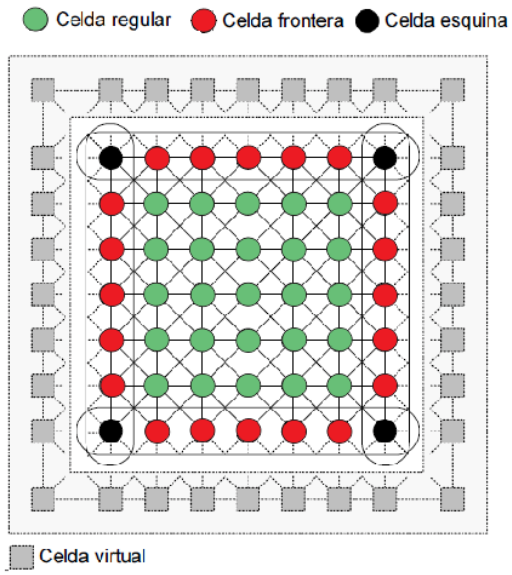


Figura 2. 6 Celdas regulares, celdas esquinas y celdas frontera. 7x7 y r=1.

Cuando hablamos de la red CNN, tenemos que pensar en la condición de frontera por que la celda tiene que contar con su vecindario completo, por lo cual se divide en neuronas estándar o regular, neuronas esquina y neuronas frontera (Figura 2. 6). En la Figura 2. 7 se ve una imagen de 19x16, recordando que el pixel de color blanco es de valor -1 y cada pixel negro es de valor 1.

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	1	-1	1	-1	-1	1	1	-1	1	1	1	-1
-1	1	-1	1	-1	1	-1	-1	1	1	-1	1	1	1	-1
-1	1	-1	1	-1	1	-1	-1	1	1	-1	1	1	1	-1
-1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1
-1	1	-1	1	-1	1	-1	-1	-1	1	-1	-1	1	-1	-1
-1	1	-1	1	-1	1	-1	-1	-1	1	-1	-1	1	-1	-1
-1	1	-1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1	1	-1
-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1	1	-1
-1	-1	1	-1	-1	1	1	-1	1	1	-1	1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Figura 2. 7 Imagen de 19x16 pixeles original.

La Figura 2. 7 sería la imagen original pero en la orilla de la imagen no se cuenta con el total de vecinos suficientes para llevar a cabo el proceso, por lo que se le coloca un marco a la imagen con el valor de 1 y por lo cual la imagen quedará de 20 x 17, Figura 2. 8. A esta condición se le conoce como frontera fija (Dirichlet). En este tipo de frontera, está asignada por un valor constante. También existen otras condiciones las cuales pueden ser usadas en vez de ésta, como sería la de frontera de Neumann, en la cual la variable de estado V_{xij} de las neuronas perpendiculares a la frontera están restringidas a ser iguales a las de unas con otras, o la condición de frontera periódica, en la cual las neuronas extremo son conectadas a las de su extremo opuesto. Pero en nuestro caso solo usaremos la condición de frontera fija.

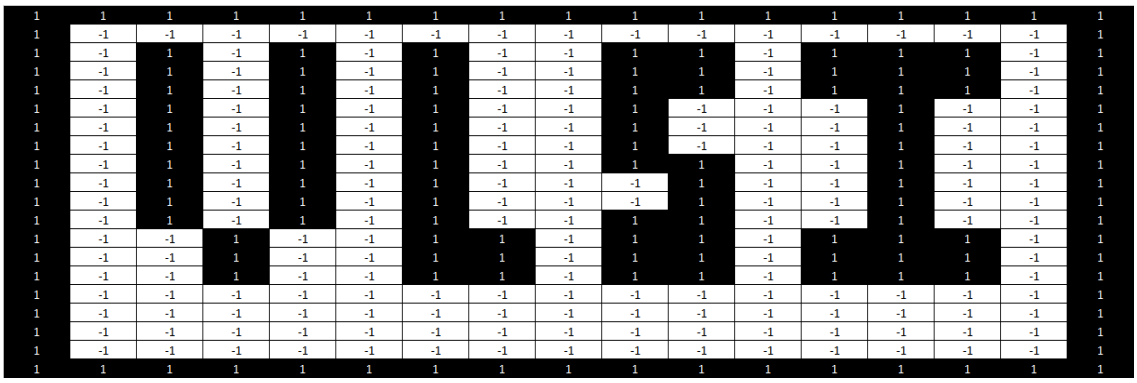


Figura 2. 8 Imagen 20x17 con marco inicial.

2.2.4 Modelo Eléctrico

La CNN en tiempo continuo puede ser representada por un circuito analógico no lineal (Figura 2. 9).

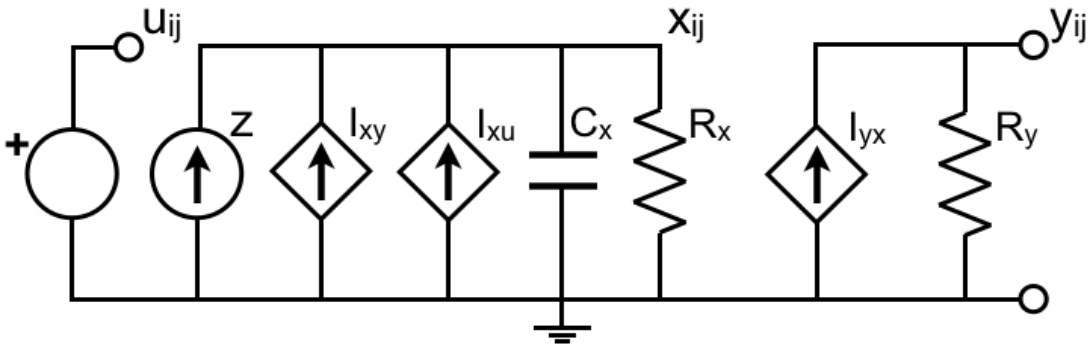


Figura 2. 9 Circuito no-lineal correspondiente a CNN de tiempo-continuo descrita en la Ec. 2.2.

Las fuentes de corriente controladas por voltaje correspondientes a la contribución de las celdas vecinas, se expresan de la siguiente manera:

$$I_{xy} = \sum_{(k,l)\in N(i,j)} A * y_{kl}\,(T)$$

$$I_{xu} = \sum_{(k,l)\in N(i,j)} B * u_{kl}$$

Y la fuente de corriente controlada por voltaje para obtener la salida $y_{ij}(t)$, seria:

$$I_{yx} = \frac{1}{R_y} * g(x_{ij})$$

Donde $g(x_{ij})$ es la función de salida de la Ec. 2. 2, el valor de x_{ij} puede ser encontrado por las leyes de corriente de Kirchhoff, obteniendo:

$$C_x \frac{dx_{ij}(t)}{dt} = -\frac{1}{R_x} x_{ij}(t) + I_{xy} + I_{xu} + z$$

Es evidente que $C_X * R_X = 1$, y por consiguiente se obtiene la Ec. 2. 2. También el valor de y_{ij} puede ser obtenido por las leyes de Kirchhoff y sería:

$$y_{ij} = R_y * I_{yx} = g(x_{ij})$$

2.3 Aplicaciones de las CNN.

Usando una simple CNN para imágenes binarias, y sobre todo apoyándonos de las mascarillas presentadas por Chua y Támas Roska [13], se seleccionaron algunos ejemplos de procesamiento de imágenes, en el cual sólo se consideran vecindarios de orden 1 para topologías rectangulares. Las siguientes imágenes fueron simuladas en un simulador en línea a través de internet [15] donde se ofrece una breve descripción del simulador CNN, contando con las características de que cada célula del sistema dinámico es idéntica y su vecindario es de un radio de 3x3; éste fue diseñado por Martin Hänggi [16] .

Rellenado de huecos.

Mascarillas para rellenado de huecos.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1.5 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ I=.5}$$

En la Figura 2. 10, se presenta una imagen de entrada con huecos en una trayectoria cerrada, los cuales serán rellenados por un fondo negro.

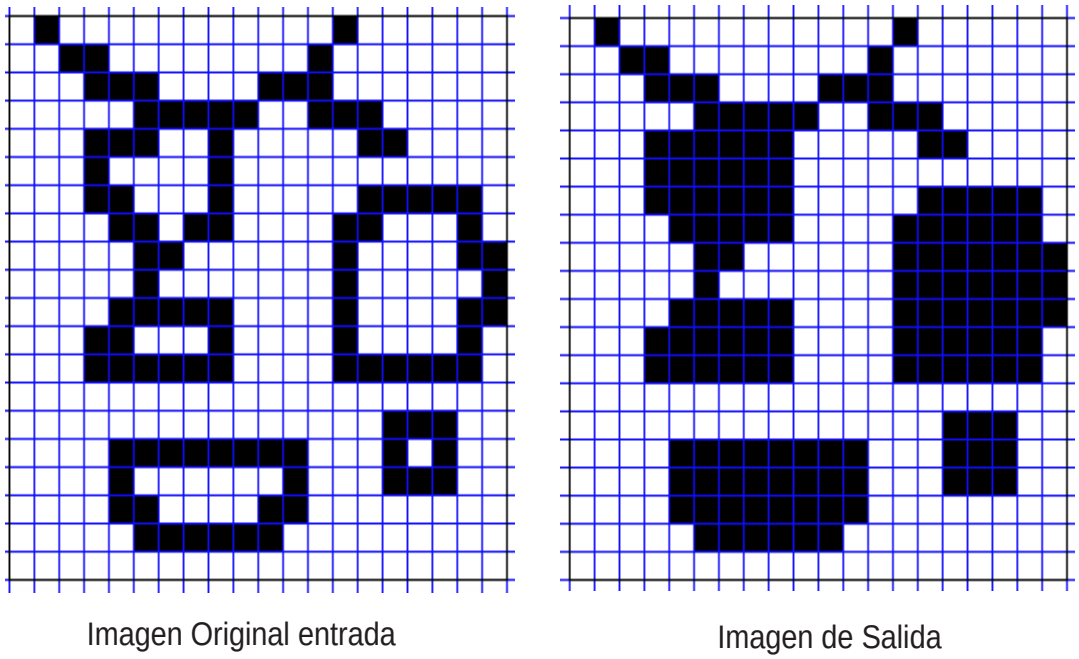


Figura 2. 10 Rellenado de Huecos.

Removedor de ruido.

Mascarillas para remover ruido.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ I=0}$$

En la Figura 2. 11 pero ahora aplicando el principio de ruido, en el cual un pixel será considerado como ruido cuando no tenga un vecino inmediato horizontal y vertical en un pixel del mismo valor.

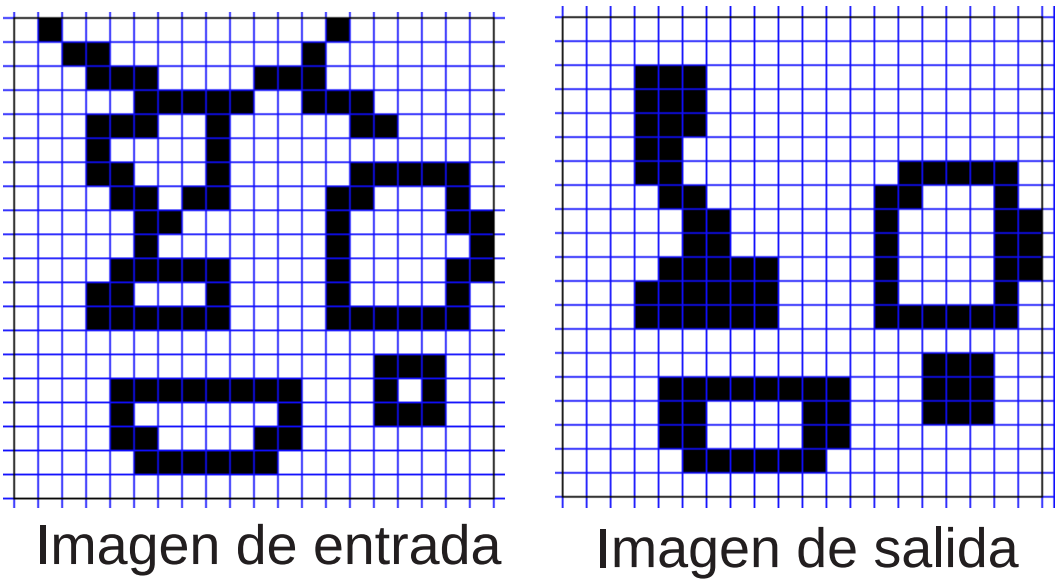


Figura 2. 11 *Removedor de ruido.*

Detector de bordes.

Mascarillas para detectora de bordes.

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$ I=-1

En la Figura 2. 12 se presenta una imagen de entrada y de salida para la aplicación de detección de bordes. Siendo las condiciones:

1. Para toda neurona que posea una entrada blanca su salida sea siempre blanca.
2. Para una neurona con entrada negra y la entrada de todas sus vecinas sea blanca, su salida deberá ser blanca.
3. Para todos pixeles negros, con excepción de (2), la salida deberá ser negra.

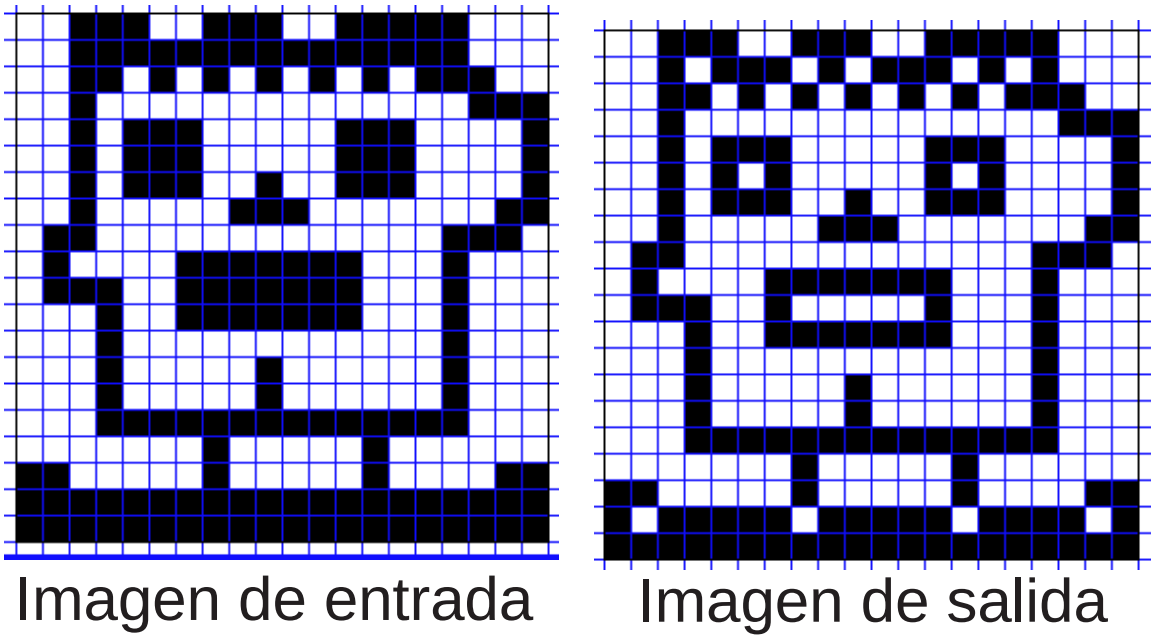


Figura 2. 12 Detectora de bordes.

2.4 Conclusiones.

En este capítulo se han presentado los conceptos básicos de las redes neuronales celulares, su arquitectura y dinámica de funcionamiento. Estos conceptos servirán de base para la generación de mascarillas optimizadas mediante el uso de algoritmo meta-heurísticos.

Las CNN's se "estructuran" de las redes neurales artificiales y las autómatas celulares, por lo cual se considera un sistema de inteligencia artificial, teniendo un rendimiento tan bueno que le permite tener una amplia gama de aplicaciones, de lo cual únicamente nos hemos centrado en el procesamiento de imágenes. El éxito de las CNN radica en la semejanza que presenta en las células biológicas de los seres vivos, donde la propagación de la información es de una forma muy eficientes aunque el procesamiento de esta información suele no ser tan bueno en cada célula, por presentar algún deterioro, pero esto no impide que la red celular no llegue a un convergencia aceptable y de calidad.

Al igual que las redes neuronales artificiales, presentan un alto nivel de aprendizaje mediante la experiencia y sobre todo un aspecto de abstracción que consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan.

La problemática del uso clásico de la CNN, el procesamiento de imágenes, es que se tiene que asignar una célula a cada pixel. Esto genera que los requerimientos de recursos sean vastos y el proceso lento.

Capítulo 3. Generación de mascarillas de CNN y otras aplicaciones de los algoritmos Meta-heurísticos en software.

En este capítulo se presentará la forma de obtención de las mascarillas para procesamiento de imágenes de una CNN, mediante la utilización de algoritmos meta-heurísticos. Las tareas a realizar, serian:

- Removedor de ruido
- Detector de bordes

También en este capítulo se desarrollarán otros problemas de optimización, resueltos mediante el uso de algoritmos meta-heurísticos, como:

- Segmentación de imágenes, mediante el PSO.
- Aproximador de funciones (procedimiento supervisado), mediante el ABC.
- Entrenamiento de redes neuronales, mediante el ABC.

Estas aplicaciones se realizaron en una PC con procesador CPU - Intel i7-3530QM 2.40GHz, ram 16 GB, el simulador matemático usado fue el Matlab R2014a de Mathworks.

3.1 Aplicación de Algoritmos Meta-Heurísticos para la obtención de mascarillas de una CNN.

A diferencia de las redes neuronales artificiales, en la CNN no existe un proceso específico para la obtención de mascarillas, siendo un tema clave para el funcionamiento de la CNN. Entre los métodos más usuales para la formulación de las mascarillas, se tiene:

- Método analítico.

Están basados en una serie de reglas locales, que caracterizan la dinámica de la celda y éstas dependen de las condiciones de las celdas vecinas. Estas reglas son transformadas en una ecuación cuyo mínimo global corresponde a la solución o mascarillas funcionales.

- Algoritmos de aprendizaje local.

Este método está derivado del entrenamiento de redes neuronales artificiales, tal como recurrent backpropagation, backpropagation through time y aprendizaje por perceptrón.

- Algoritmos de aprendizaje global.

Este modelo se caracteriza por usar técnicas de optimización estocásticas. Este proceso es muy semejante al local, con la diferencia de que los datos de entrada y salida son presentados de manera global a toda la red y no solamente a una neurona de forma local.

Cada método depende del problema en cuestión, ya que cada uno tiene diferentes criterios de exactitud.

El método a utilizar fue propuesto por K. Nakai y A. Ushida [17], podría ser clasificado como un método analítico y en parte en un método de aprendizaje local, ya que cuenta con rasgos de ambos. En una forma coloquial de explicarlo, es que si establecemos la entrada de nuestra red neuronal celular podemos forzar la respuesta de la ecuación de estado para obtener los resultados deseados, esto mediante la formulación de un sistema de desigualdades lineales, que a su vez se trasforman en una función objetivo mediante la función de penalización, y con esta función objetivo poder encontrar las mascarillas mediante un algoritmo de optimización, que en nuestro caso será el algoritmo de optimización basado en la colonia artificial de abejas (ABC).

3.1.1 Método de diseño de desigualdades para las mascarillas de una CNN.

Lo primero para generar las desigualdades correspondientes a las distintas tareas de procesamiento de imágenes mediante un CNN, es que la CNN alcance el estado estable, por lo que es necesario que se cumplan las siguientes condiciones:

1. Que se garantice que la salida de CNN llegue a los puntos de equilibrio $V_{yij} = \pm 1$, esto mediante del cumplimento, de:

$$A(I, J: K, L) > 1 \tag{3. 1}$$

2. Se considerará que la variable de estado de cada celda $V_{ij}(t)$, es igual al valor de la entrada que posea dicha celda, esto es, cuando $\frac{dV_{ij}(t)}{dt} = 0$, lo que hace que la Ec. 2.2 se convierta en:

$$v_{xij} = \sum_{C(k,l)\in Nr(i,j)} A(i,j;k,l)v_{ykl} + \sum_{C(k,l)\in Nr(i,j)} B(i,j;k,l)v_{ukl} + I_{ij} \tag{3. 2}$$

Donde v_{xij} y V_{ykl}, se encuentran en estado estacionario.

Para generar el arreglo de desigualdades es necesario que nuestra ecuación de estado cumpla las consideraciones siguientes, y de las cuales A y B son derivadas de la Ec. 3.2:

A. Se desea que la respuesta de la celda sea v_{yij}=+1. Para poder llegar a esta respuesta, es necesario que la Ec. 3.2, cumpla los valores $v_{xij} < 1$.

$$1 \leq \sum_{C(k,l)\in Nr(i,j)} A(i,j;k,l)v_{ykl} + \sum_{C(k,l)\in Nr(i,j)} B(i,j;k,l)v_{ukl} + I_{ij} \tag{3. 3}$$

B. Se desea que la respuesta de la celda sea v_{yij}= - 1. Para poder llegar a esta respuesta, es necesario que la Ec. 3.2, sea $v_{xij} \leq -1$.

$$-1 \geq \sum_{C(k,l)\in Nr(i,j)} A(i,j;k,l)v_{ykl} + \sum_{C(k,l)\in Nr(i,j)} B(i,j;k,l)v_{ukl} + I_{ij} \qquad (3.\ 4)$$

C. Cuando se desea $v_{yij}(\infty)$=+1, entonces se puede presentar el caso en que $V_{xij}(0) < 1$, entonces debe satisfacer $\frac{dV_{ij}(t)}{dt} > 0$, esto para garantizar el cruce por cero en toda la región y alcanzar la saturación (+1). Entonces las ecuación de estado (2.2) debe cumplir:

$$0 < -V_{xij}(0) \sum_{C(k,l)\in Nr(i,j)} A(i,j;k,l)v_{ykl}(0) + \sum_{C(k,l)\in Nr(i,j)} B(i,j;k,l)v_{ukl}(0) + I_{ij} \qquad (3.\ 5)$$

D. Cuando se desea $v_{yij}(\infty)$=-1, entonces se puede presentar el caso en que $V_{xij}(0) > -1$. Entonces, se debe satisfacer $\frac{dV_{ij}(t)}{dt} < 0$, esto para garantizar el cruce por cero en toda la región y alcanzar la saturación (-1). Las ecuación de estado (2.2) debe cumplir:

$$0 > -V_{xij}(0) \sum_{C(k,l)\in Nr(i,j)} A(i,j;k,l)v_{ykl}(0) + \sum_{C(k,l)\in Nr(i,j)} B(i,j;k,l)v_{ukl}(0) + I_{ij} \qquad (3.\ 6)$$

Donde $V_{xij}(0) = V_{uij}$; el conjunto de las desigualdades pasadas más la desigualdad (3. 1) se puede generar el sistema de desigualdades de estabilidad. Al generar todas las desigualdades que describen el resultado deseado (más adelante se aclarará este párrafo) se procede a transformarlas en una función objetivo [18].

3.1.2 Formulación de Función Objetivo para las mascarillas de una CNN.

Una vez generado el conjunto de desigualdades presentes para cada caso posible, se pasa a generar la función objetivo, la cual será analizada por el algoritmo de optimización basado en colonia artificial de abejas, para que sean optimizados los valores de las mascarillas, donde los resultados más óptimos serán aquellos que se aproximen más al centro del área de solución de las desigualdades.

Pero para poder llevar a cabo este análisis, hay que pasar de un problema de restricciones a uno sin restricciones, esto se realiza mediante el uso de la función de penalización. La labor de la función de penalización es convertir un conjunto de desigualdades en forma de la función objetivo; lo primero será agregar un término o factor a cada una de las desigualdades, teniendo en cuenta si la solución considerada cumple las restricciones, esto de modo de disminuir el valor de la función y para que tenga mejor probabilidad de sobrevivencia a la búsqueda de un mínimo de optimización [19].

Pasos para generar la función objetivo.

1. Se reescriben todas las desigualdades de la siguiente forma: $f_k(\boldsymbol{T}) \geq 0$, donde **T** son los coeficientes de las mascarillas. Por consiguiente la función objetivo puede ser definida como:

$$F(\boldsymbol{T}) = \sum f_k(\boldsymbol{T})^2$$

2. Para que la búsqueda se encuentre dentro del espacio de búsqueda, se agrega una constante ϕ (característica de las funciones de penalización), por lo cual la ecuación anterior será definida como:

$$\hat{F}(\boldsymbol{T}) = \sum \hat{f}_k(\boldsymbol{T})^2 \qquad (3.\ 7)$$

Donde:

$$\hat{f}_k(\boldsymbol{T}) = \begin{cases} f_k(\boldsymbol{T}) - \phi & f_k(\boldsymbol{T}) - \phi \geq 0 \\ p + m * (f_k(\boldsymbol{T}) - \phi) & f_k(\boldsymbol{T}) - \phi < 0 \end{cases} \qquad (3.\ 8)$$

Las constantes p y m son cantidades reales positivas y grandes, por lo que el resultado será un número muy grande, y por consiguiente, será eliminado por nuestro algoritmo de optimización. Cuanto **T** no cumple con una o varias de las desigualdades, serán despreciadas por nuestro algoritmo de optimización, ya que busca minimizar la función [20].

3.1.2.1 Conjunto de desigualdades a función objetivo.

Para ejemplificar el proceso antes descrito, se proponen las siguientes desigualdades, las cuales mediante el uso de la función de penalización, serán presentadas en una función objetivo con ϕ = 1, por ejemplo:

- $x + y - 2 < 0 \ \rightarrow \ \hat{f}(-x - y + 2 - 1)^2$
- $x - y > 0 \ \rightarrow \ \hat{f}(x - y - 1)^2$
- $x > 0 \ \rightarrow \ \hat{f}(x - 1)^2$

- $y > 0 \rightarrow \hat{f}(y-1)^2$

El siguiente paso será generar la función objetivo, esto mediante el uso de la ec. (3. 7).

$$\hat{F}(\boldsymbol{T}) = \hat{f}(-x-y+2-1)^2 + \hat{f}(x-y-1)^2 + \hat{f}(x-1)^2 + \hat{f}(y-1)^2$$

Teniendo la función objetivo, se procede a realizar la búsqueda del mínimo mediante el uso del algoritmo de optimización basado en la colonia artificial de abejas (ABC) en Matlab. En la Figura 3. 1, se puede apreciar gráficamente la función objetivo y los resultados obtenidos mediante la aplicación del ABC, donde se utilizó una colonia de abejas de 30 miembros y 100 iteraciones. Obteniendo de resultado de X=1, y =0.3333 y $\hat{F}(\boldsymbol{T}) = 0.66667$.

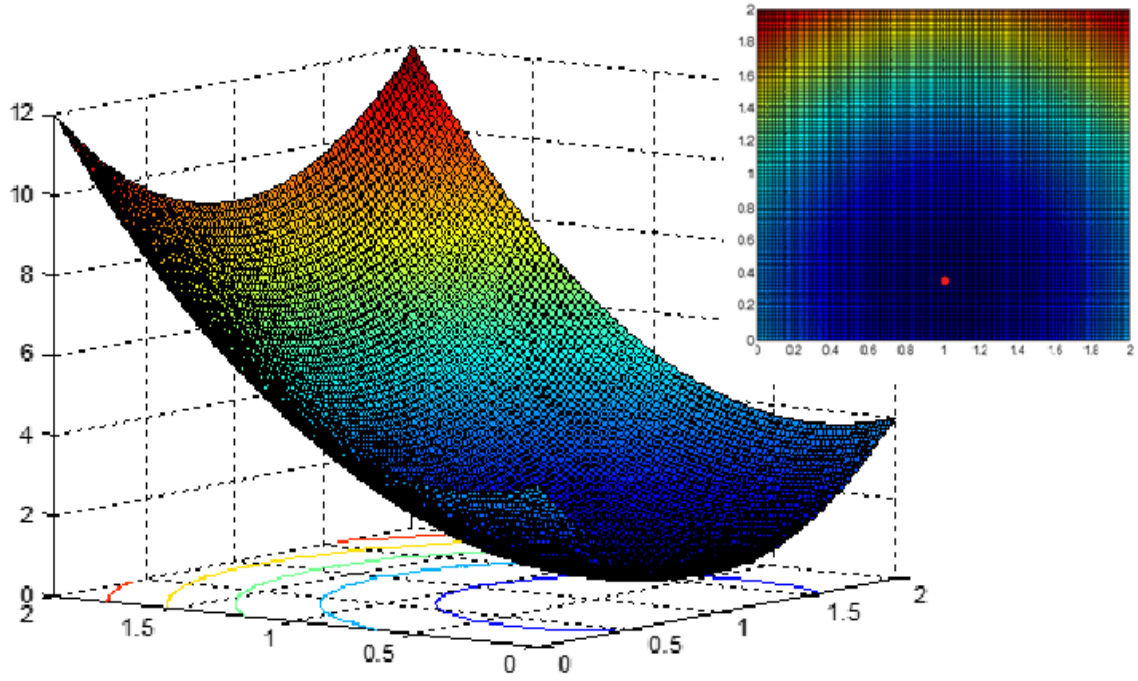


Figura 3. 1 Gráfico de la función $\hat{F}(\boldsymbol{T})$, generada mediante un proceso de desigualdades a función objetivo.

3.1.3 Diseño de mascarilla de la CNN para la tarea de remover ruido.

La idea es remover de una imagen pixeles blancos o negros que no se encuentren en un grupo que tenga las mismas características, a esto se le considerara ruido. En otras palabras, un pixel se considerara ruido cuando no tenga un su vecindario contiguo un pixel del mismo valor y para solucionarlo o remover el ruido, el pixel en cuestión cambiará su valor al de su vecindario, esto para mantener una uniformidad en la imagen binaria, ver Figura 3. 2.

En la selección de las variables de las mascarillas y las mascarillas en sí, no existe un procedimiento definido para elegirlas, por lo cual se hizo una selección heurística, pero podríamos decir que de manera general en las tarea que se necesite propagación de la

información entre las neuronas, la mascarilla A deber ser considerada, mientras en aquellas que solo es necesaria la información inicial, se consideraría la mascarilla B, pero se ha demostrado que distintos valores y órdenes en las mascarillas puede llevar al mismo resultado en la CNN.

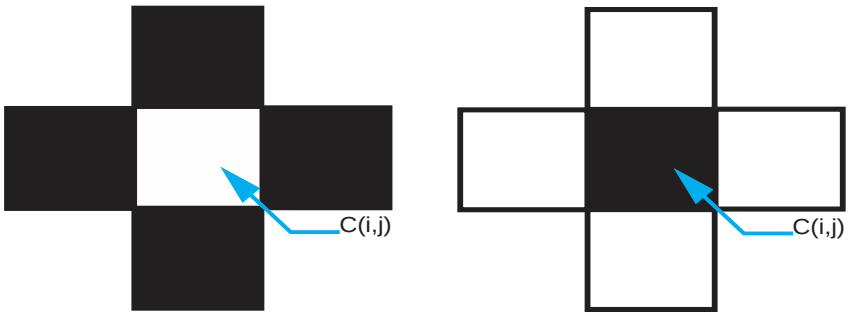


Figura 3. 2 Casos para que un pixel se considere ruido.

La configuración de las mascarillas A, B e I (umbral) a usar para una topología rectangular, será de la siguiente manera:

$$A = \begin{bmatrix} 0 & \mathrm{a} & 0 \\ \mathrm{a} & a_o & a \\ 0 & \mathrm{a} & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & b_o & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ y I}$$

Donde a_o, a, b_o son los valores a determinar.

Para generar el conjunto de desigualdades, hay que tener presente todas las configuraciones de entrada posibles para la neurona y las respuestas deseadas, también tomar en cuenta la frontera de tipo Dirichlet con valor cero.

A continuación se presenta un ejemplo de cómo se generar las desigualdades, las cuales formarán la función objetivo.

Un ejemplo sería, si tenemos un pixel de entrada $V_{uij} = +1$ (negro), se tiene que cumplir la condición D, mencionados en el subtema 3.1.1, esto para asegurar el cruce por la región lineal de la salida de la función de trasferencia hacia -1, por lo tanto:

$$0 > -V_{xij}(0) \sum_{C(k,l)\in Nr(i,j)} A(i,j;k,l)v_{ykl}(0) + \sum_{C(k,l)\in Nr(i,j)} B(i,j;k,l)v_{ukl}(0) + I_{ij}$$

Sustituyendo, las matrices propuestas, nos queda de la forma:

$$0 > 1 - a_o(1) - \mathrm{a}(1) - \mathrm{a}(1) - \mathrm{a}(1) - \mathrm{a}(1) + b_o(1) + I$$

La que se puede simplificar de la siguiente manera:

$$-a_o - 4\mathrm{a} + b_o + I \leq -1$$

Ahora para $V_{yij} = -1$, tenderá asintóticamente hacia un valor por debajo de -1, por lo que debe cumplir la condición D, esto es:

$$-1 \geq \sum_{C(k,l)\in Nr(i,j)} A(i,j;k,l)v_{ykl} + \sum_{C(k,l)\in Nr(i,j)} B(i,j;k,l)v_{ukl} + I_{ij}$$

Sustituyendo las matrices propuestas, nos queda de la forma:

$$-1 \geq a_o(1) - \mathrm{a}(1) - \mathrm{a}(1) - \mathrm{a}(1) - \mathrm{a}(1) + b_o(1) + I$$

la que se puede simplificar de la siguiente manera:

$$a_o - 4\mathrm{a} + b_o + I < 1$$

Este proceso se realizará también para transición de blanco al negro, los casos donde la entrada y la salida tienen los mismos valores, hasta generar el conjunto de desigualdades (Tabla 3.1).

Píxel de Entrada v_{uij}	Píxel de Salida v_{yij}	Neuronas del vecindario C_{kl} $k \neq i, l \neq j$		Desigualdades
		No. de Negros	No. de Blancos	
Negro (+1)	Blanco(-1)	0	4	-ao - 4a + bo+ I ≤ -1 B)
				ao - 4a + bo + I < 1 D)
		0	3	-ao - 3a + bo + I ≤ -1 B)
				ao - 3a + bo + I < 1 D)
Negro (+1)	Negro (+1)	1	3	ao – 2a + bo + I ≥ 1 A)
		1	2	ao – a + bo + I ≥ 1 A)
		2	2	ao + bo + I ≥ 1 A)
		2	1	ao + a + bo + I ≥ 1 A)
		3	1	ao + 2a + bo + I ≥ 1 A)
		3	0	ao + 3a + bo + I ≥ 1 A)
		4	0	ao + 4a + bo + I ≥ 1 A)
Blanco(-1)	Blanco(-1)	0	4	-ao – 4a - bo + I ≤ -1 B)
		0	3	-ao – 3a - bo + I ≤ -1 B)
		1	3	-ao – 2a - bo + I ≤ -1 B)
		1	2	-ao – a - bo + I ≤ -1 B)
		2	2	-ao - bo + I ≤ -1 B)
		2	1	-ao + a - bo + I ≤ -1 B)
		3	1	-ao +2a - bo + I ≤ -1 B)
Blanco(-1)	Negro (+1)	3	0	ao + 3a - bo + I ≥ 1 A)
				-ao + 3a - bo + I > -1 C)
		4	**0**	**ao + 4a - bo + I ≥ 1 A)**
				-ao + 4a - bo + I > -1 C)
Condición para Estabilidad				ao > 1

Tabla 3.1 Conjunto de desigualdades para la tarea de removedor de ruido.

En el Anexo A, se presenta el código en Matlab que simula el ABC, para la obtención de las mascarillas, así como el simulador hecho de la CNN donde se verifico el funcionamiento de las mascarillas. También se utilizó el simulador de Martin Hänggi [21], este simulador tiene la característica de que se puede observar la evolución de la CNN; esto nos permite entender mejor el funcionamiento de la CNN.

Las mascarillas obtenidas para la tarea de remover ruido donde ϕ=1, es:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1.8 & 0 \\ 1.8 & 2 & 1.8 \\ 0 & 1.8 & 0 \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ e } \mathbf{I}=0$$

Los resultados obtenidos en una imagen con ruido se pueden ver en la Figura 3. 3.

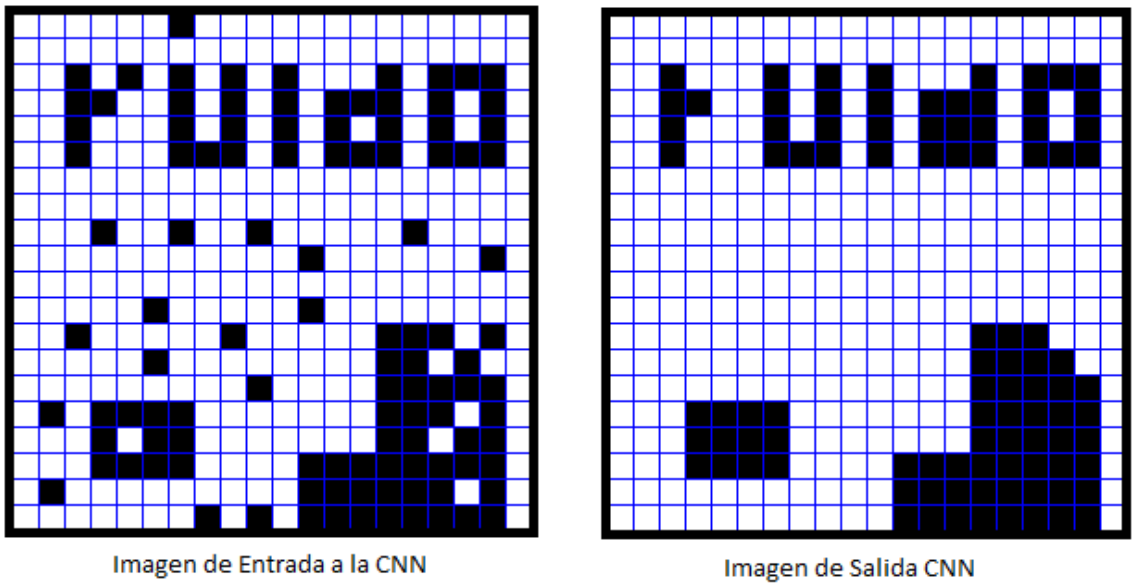


Figura 3. 3 Resultados del simulador de CNN para la tarea de remover ruido.

3.1.4 Diseño de mascarilla de la CNN para la tarea de detector de bordes.

La detección de bordes es una herramienta fundamental en el procesamiento de imágenes, esto para la detección y extracción de características, que tiene como objetivo la identificación de un punto donde la imagen binaria cambia su valor de 0 a 1 o viceversa. Esto facilita la compresión de grandes cantidades de información en la imagen.

A continuación se planteará el proceso para poder obtener las mascarillas correspondientes a la tarea de detección de bordes de una silueta negra, por lo que requerimos que una frontera de tipo Dirichlet con valor cero.

El planteamiento del problema se presenta de la siguiente manera:

1. Todas las neuronas que tengan una entrada blanca (-1) mantendrán su estado en blanco.

2. Cuando la entrada de la neurona sea negra (+1), y la entrada de todas las neuronas vecinas sea blanca su salida deberá ser blanca.
3. Para todos los casos de entrada negra, con excepción del caso anterior, la salida deberá ser negra.

Basándonos en la configuración de las mascarillas propuesta en [20], se presenta de la siguiente forma:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & a_o & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} 0 & \mathrm{b} & 0 \\ b & b_o & b \\ 0 & b & 0 \end{bmatrix} \text{ e } \mathbf{I}$$

Esta configuración es presentada de esta forma debido a que no es necesaria la propagación de la información de salida de las neuronas por lo cual también se podría descartar la mascarilla A.

Utilizando el proceso de generación de desigualdades se obtiene la

.2, con la cual se realiza la función objetivo a ser optimizada por el ABC, en la cual se utilizaron 30 abejas y 1000 iteraciones.

CASOS	Píxel de Entrada v_{uij}	Píxel de Salida v_{yij}	Neuronas del vecindario C_{kl} $k \neq i, l \neq j$		Desigualdades
			No. de Negros	No. de Blancos	
(1)	Blanco (-1)	Blanco (-1)	4	0	-ao + 4b - bo + I ≤ -1
			3	1	-ao + 2b - bo + I ≤ -1
			2	2	-ao- bo + I ≤ -1
			1	3	-ao – 2b - bo + I ≤ -1
			0	4	-ao – 4b - bo + I ≤ -1
			3	0	-ao +3b - bo + I ≤ -1
			2	1	-ao + b - bo+ I ≤ -1
			1	2	-ao – b - bo+ I ≤ -1
			0	3	-ao – 3b - bo + I ≤ -1
(2)	Negro(1)	Negro(1)	3	1	ao +2b+ bo + I ≥ 1
			2	2	ao + bo + I ≥ 1
			1	3	ao – 2b + bo + I ≥ 1
			0	4	ao – 4b + bo + I ≥ 1
			3	0	ao + 3b +bo + I ≥ 1
			2	1	ao +b +bo + I ≥ 1
			1	2	ao –b +bo + I ≥ 1
			0	3	ao – 3b + bo + I ≥ 1
(3)	Negro (1)	Blanco(-1)	4	0	-ao + 4b + bo+ I ≤ -1
					ao + 4b + bo + I < 1
Condición para Estabilidad					ao > 1

Tabla 3.2 Desigualdades para la obtención de mascarillas para la tarea de detención de bordes.

Las mascarillas resultantes, quedan de la siguiente forma:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2.9 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 1.8 & -1 \\ 0 & -1 & 0 \end{bmatrix} \text{ e } \mathbf{I} = -1$$

Los resultados obtenidos en una imagen, con mascarillas para detección de bordes en una CNN, se muestran en la Figura 3. 4.

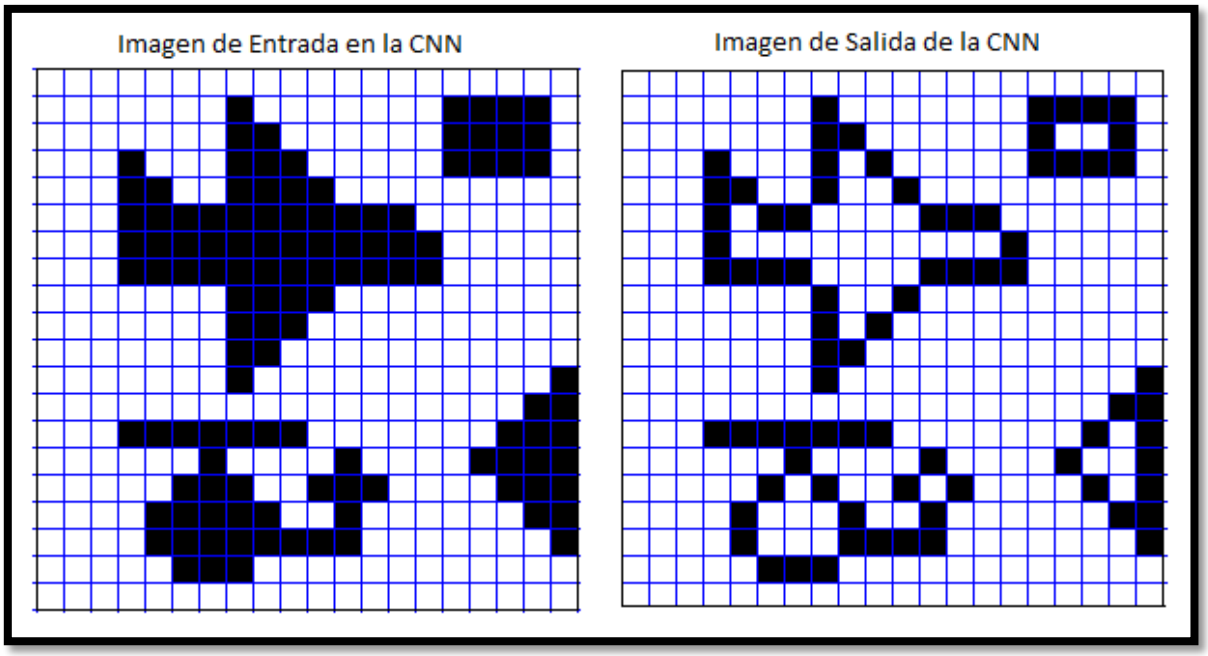


Figura 3. 4 Resultados del simulador de CNN para la tarea de detección de bordes.

3.2 Segmentación de Imágenes con Algoritmo de Optimización mediante Cúmulos de Partículas (PSO).

El análisis de imágenes comprende múltiples métodos, de los cuales se busca extraer información de esta misma. Entre todas las metodologías, una de la más usadas es la segmentación de imágenes las cual se ocupa de descomponer una imagen en sus partes constituyentes, es decir, los objetivos de interés y el fondo, basándose en las características locales que nos permite distinguir un objeto del fondo y objetos entre sí.

La segmentación de imágenes es una técnica importante en detección, clasificación y reconocimiento de patrones de interés. Existen muchos estudios y desarrollos para esta

tarea, como en nuestro caso sería la aplicación de un algoritmo meta-heurístico como es el PSO. La aplicación de esta técnica tiene interés principalmente en el área médica o industrial, por lo cual se utilizan imágenes de tomografías tomadas de internet.

La mayoría de las imágenes están conformadas por regiones o zonas que comparten características como son nivel de gris, textura, momento, etc. La forma de operación de la segmentación de una imagen es distinguir si un pixel pertenece, o no, a un objeto de interés [22].

Los métodos de segmentaciones se pueden agrupar en cuatro clases diferentes:

i. Métodos basados en pixeles, que a su vez pueden ser:
    a. Locales.- Éste es basado en las propiedades de los pixeles y su entorno.
    b. Global.- Éste se basa en la información global obtenida, por ejemplo de un histograma, el cual es la distribución de frecuencia de la cantidad de pixeles grises en una imagen en escala de grises.
ii. Métodos basados en bordes.
iii. Métodos basados en regiones, que utilizan las nociones de homogeneidad y proximidad geométrica, como las técnicas de crecimiento, fusión o división.
iv. Métodos basados en modelos.

En este trabajo nos enfocaremos el método basado en pixeles globales.

3.2.1 Descripción de la propuesta para segmentación de imágenes mediante PSO.

En el proceso para la función de segmentación de imágenes, se inicia transformando la imagen a segmentar a una imagen a escala de grises (Figura 3. 5), ésta es una escala empleada en el procesamiento de imágenes digitales en la cual cada pixel posee un valor equivalente a una graduación de gris. Esta arquitectura de la conversión de escala de grises está basada en la ecuación matemática que enlaza el espacio de YCrCb al espacio de RBG (Red, Green y Blue). La siguiente ecuación tiene la función de trasformar el pixel de color a uno en la graduación de grises:

$$Y = 0.299R + 0.587G + 0.114B$$

En la Figura 3. 6 se puede ver gráficamente el cubo de colores RGB y a lo largo de la diagonal que cruza del origen al punto (1, 1, 1), la graduación para la escala de grises con respecto al RGB.

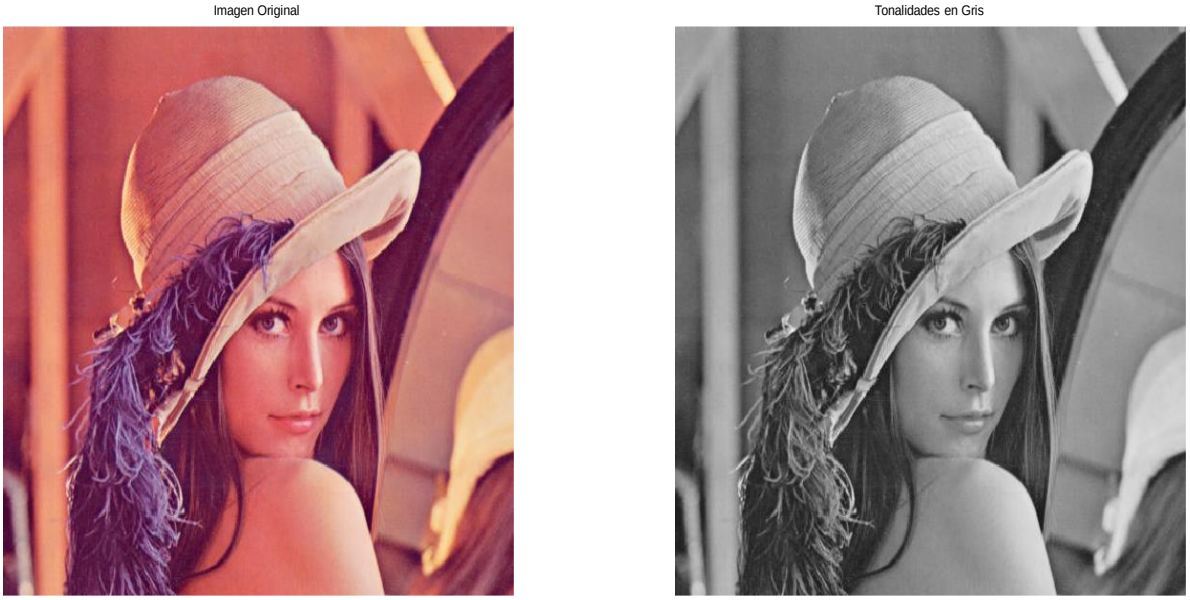


Figura 3. 5 Imagen de Lena trasformada en escala de grises 512x512 pixeles.

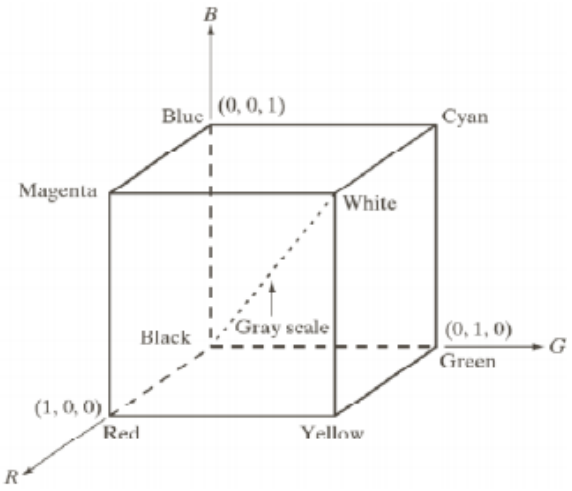


Figura 3. 6 Esquema cúbico de los colores RGB.

El siguiente paso sería el obtener la frecuencia de distribución de pixeles grises en la imagen (histograma). La escala de grises va de 0 a 255 niveles, donde el valor 255 corresponde a los pixeles banco y en valor 0 a los pixeles negros, donde el intervalo entre estos dos es un abanico de pixeles oscuros, grises oscuros, grises claros y claros. El histograma de una imagen consiste de una gráfica donde se muestra el número de pixeles, n_k, de cada nivel de gris, r_k, que aparecen en la imagen. Ejemplo, en la Figura 3.7, una pareja de oseznos polares en la nieve, donde su histograma, nos muestra que la mayor cantidad de pixeles son grises claros y blancos.

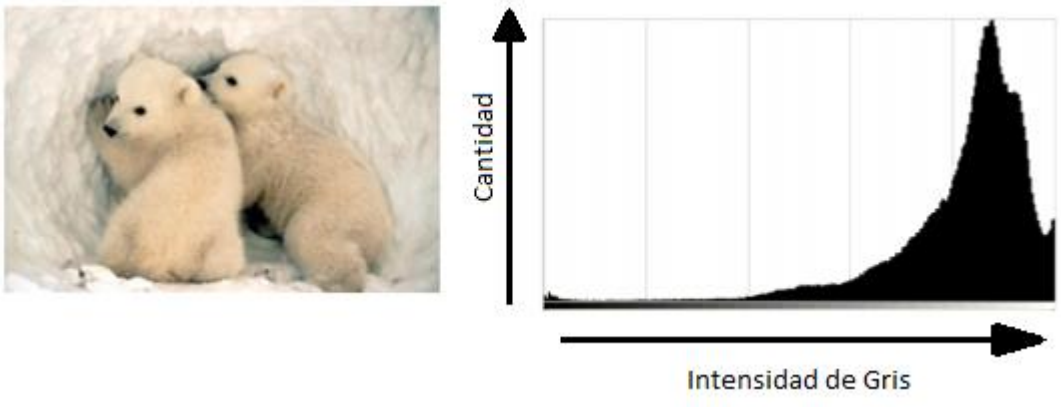


Figura 3.7 Fotografía con tonos claros y su correspondiente histograma.

Teniendo el histograma de la imagen a segmentar, se procede a inicializar el PSO, pero es necesario asociar una función objetivo, la cual será "la piedra angular del problema", ya que nos permite obtener el umbral de la imagen. En este caso se utilizó como función objetivo la función de densidad de probabilidad empírica [23], donde dado un valor *b*, que puede tener un valor entre 1<*b*<h, h es el número del histograma y I_i la intensidad del pixel en la i^{th} posición. La función de fitness, será:

$$F(b) = \frac{1}{n}\sum_{i=1}^{n} \delta(I_i - b) \tag{3. 9}$$

$$\text{Con: } \delta(k) = \begin{cases} 1 & if\ k = 0 \\ 0 & cualquier\ otro \end{cases}$$

Después de haber obtenido el umbral mediante el uso del algoritmo del PSO, se procede a binarizar la imagen, este es el último paso y es muy sencillo. En pocas palabras, se tiene que todo aquel pixel que tenga un valor mayor al umbral resultado del PSO, se convertirá en un pixel negro y de lo contrario en un pixel blanco, lo que se realizaría de la síguiete forma:

$$\text{Imagen umbralizada (i,j)} = \begin{cases} 1 & if\ gbest > imagen\ escala\ en\ grises\ (i,j) \\ 0 & cualquier\ otro \end{cases}$$

En Figura 3.8, se observa una tomografía encefálica tomada de internet, en la cual se aplicó el proceso de segmentación, donde la imagen se dividió en 2 regiones de importancia, Figura 3.8 (d). Se optó por una búsqueda en la región de 30 a 75 en el histograma y Figura 3.8 (e), se optó por una búsqueda en región 76 a 200 en el histograma, y la Figura 3.8 (f) corresponde a una búsqueda completa en todo el histograma. En este caso se utilizaron 30 partículas en 100 iteraciones con coeficientes cognoscitivos de 0.5 y una inercia de 0.5.

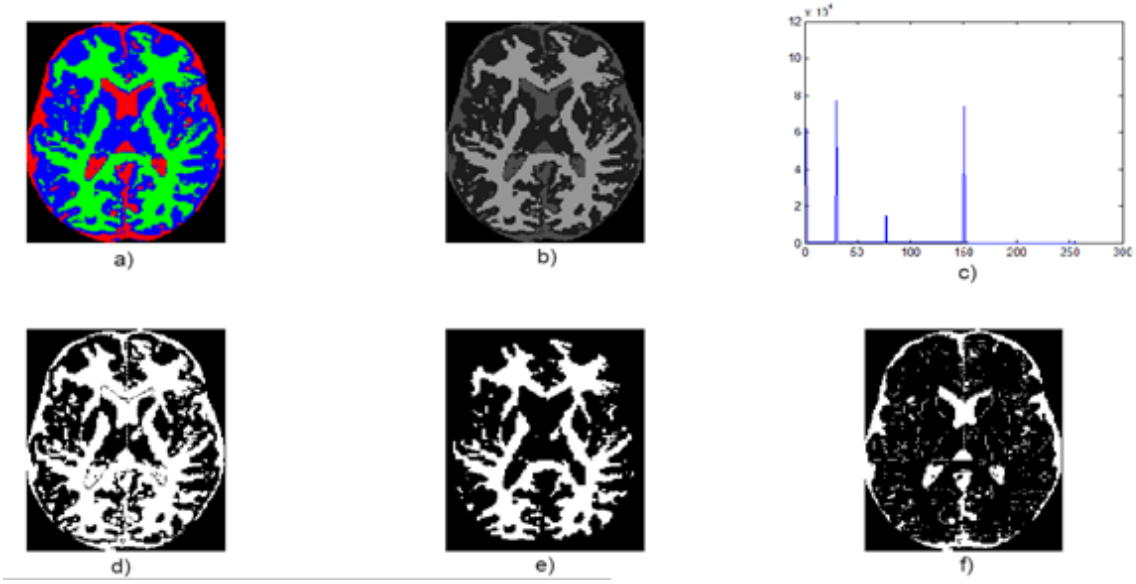


Figura 3.8 Tomografía encefálica, a) imagen original, b) imagen trasformada a escala de grises, c) histograma de la imagen b, d) región 1 umbral = 36, e) región 2 umbral 99 y f) Umbral global = 40

En la Figura 3. 9, se presenta otro caso de segmentación de imagen, con la diferencia que éste se dividió en 4 regiones para poder encontrar los distintos objetivos dentro de la misma imagen. Las características de funcionamiento del PSO, siguen siendo las mismas que anteriormente se mencionaron.

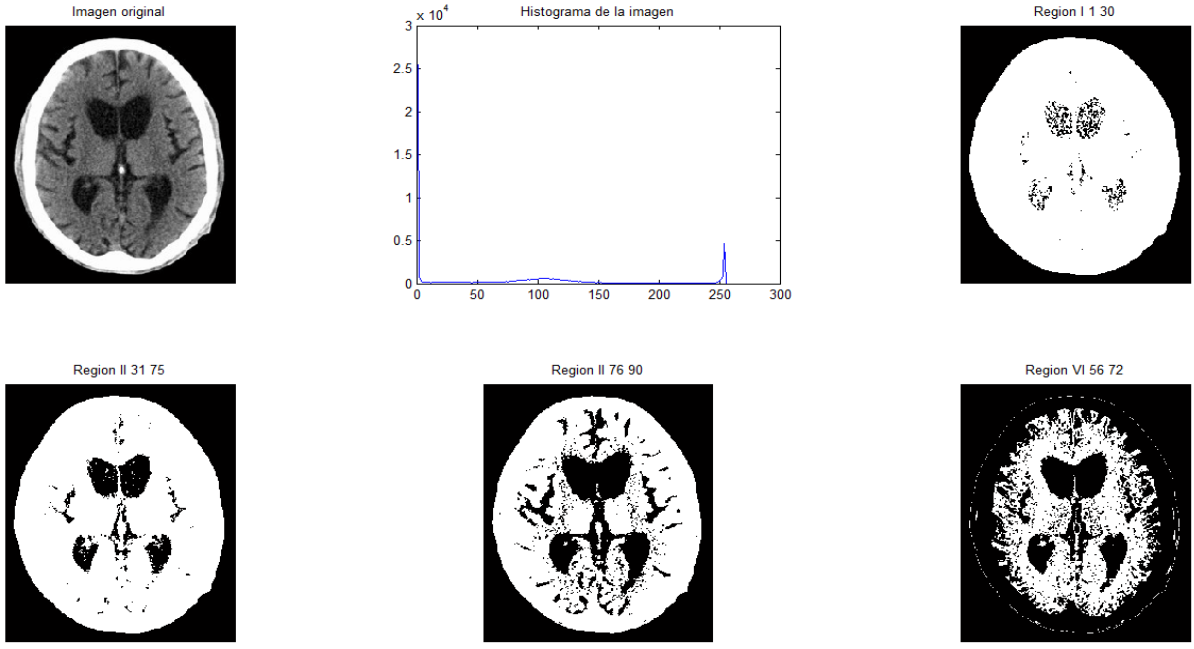


Figura 3. 9 Tomografía encefálica dividida en 4 regiones.

3.2.2 Pseudo-código para segmentación de imágenes mediante PSO.

Se aplicó el Algoritmo 1.8 que corresponde al algoritmo de PSO global con un modelo completo, ya que tanto se usó el coeficiente social como el coeficiente cognitivo (C1 y C2), ambos con valores de 0.5 y una inercia (w) de 0.5, a 100 iteraciones. La búsqueda se puede realizar por todo el histograma (0 a 255) o por regiones (segmentos del histograma), esto para obtener la mayor cantidad de información de la imagen.

Algoritmo 3.1

Lectura de imagen a segmentar.
Convertir imagen a escala de grises.
Generación de histograma.

Inicializar cúmulo.
La variable X_i puedo tomar valores de 0 a 255.

While no se alcance las iteraciones deseadas do
 For i=1 hasta n do
  Evaluar cada partícula X_i del cumulo mediante la ecuación de fitness (Ec. (3. 10)), es genera que se tenga que leer todo el histograma o región seleccionada con cada partícula para poder llegar al umbral óptimo.

  If fitness(X_i) es menor que fitness ($pBest_i$) then
   $pBest_i=X_i$;
   fitness($pBest_i$)=fitness(X_i);
  end if

 end For
 For i=1 hasta n do
  Escoger $lBest_i$, la partícula con mejor fitness de X_i.
  $V_i^{k+1} = W * V_i^k + \Phi 1 * \text{rand}1\left(pBest_i + X_i^k\right) + \Phi 2 * \text{rand}2(lBest_i - X_i^k)$
  $X_i = X_i + V_i$
 end For
end While

 TH= $lBest_i$

Binarizacion de la imagen, con apoyo del histograma.

3.3 Aproximador de funciones, usando ABC.

La técnica de análisis de componentes independientes (ICA, pos sus siglas en inglés, Independent Component Analysis), tiene una amplia gama de aplicaciones. Una forma sucinta de explicar mediante un ejemplo esta técnica, sería:

> Consideremos una fiesta en la que varias personas hablan de forma simultánea. En esta fiesta, cada persona percibe una señal acústica en la que se mezclan las voces del resto de los asistentes. Pero para que dos personas o más mantengan una conversación, es necesario que cada uno de los miembros pueda identificar las palabras emitidas por el interlocutor, por lo cual una actividad cognoscitiva del cerebro debe de realizarse para identificar la señal emitida por la persona que está hablando. Al cerebro esto se le facilita por que se apoya de las características acústicas y visuales, como es la intensidad de la voz, tono, gesticulación, mímica y movimiento de los labios, esto debido a que cada persona emite una señal que estadísticamente es independiente de las del resto de los miembros de la fiesta.
>
> Aquí es donde se puede hacer uso del análisis de componentes independientes, el cual es una técnica estadística que permite identificar fuentes estadísticamente independientes a partir de un conjunto de señales mezcladas. En otras palabras, ICA permite descomponer una determinada señal en las fuentes independientes. ICA trabaja de una forma que se considera "a ciegas" debido a que sólo se le presenta la mínima información con la cual esta puede obtener las fuentes.

El objetivo del aproximador de funciones, es el de aproximarse a la respuesta de un sistema o "función objetivo". Entre los modelos más usuales para trabajar con aproximadores de funciones, están las redes neuronales artificiales entrenadas con back-propagation o perceptrón.

Este aproximador de funciones, funciona de una forma supervisada mediante el uso de la función de error cuadrático medio, donde se analiza la respuesta en contra de una señal objetivo, para saber qué tanto se parece una de la otra. Para poder llevar a cabo este proceso y poder obtener los pesos necesarios para la separación de señales, se utilizará el algoritmo de optimización basado en colonia artificial de abejas (ABC).

3.3.1 Análisis de componentes Independientes (ICA).

El análisis de componente independieres, ICA (pos sus siglas en inglés, Independent Component Analysis), es un método para la separación a ciegas de fuentes, basado en la independencia estadística de dichas fuentes [24].

Consideremos un problema donde se encuentran dos fuentes, por ejemplo dos sonidos diferentes captados por un micrófono, donde la mezcla de dichos sonidos, se mezcla con un cierto coeficiente de ponderación (Figura 3. 10).

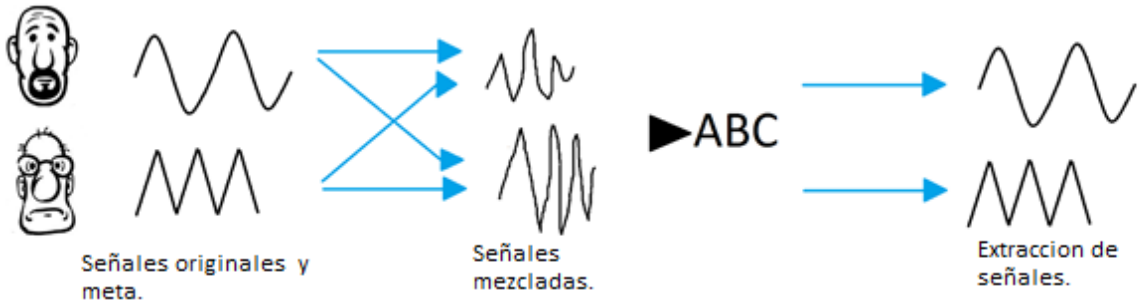


Figura 3. 10 Proceso de las señales.

Llamaremos a los sonidos mezclados como $X_1(t)$ y $X_2(t)$, y éstas serán las señales observadas. $S_1(t)$ y $S_2(t)$ son las señales originales, donde la relación entre ambas quedará de la siguiente manera:

$$X_1(t) = a_{11}S_1(t) + a_{12}S_2(t)$$
$$X_2(t) = a_{21}S_1(t) + a_{22}S_2(t) \quad (3.\ 11)$$

O de forma equivalente:

$$X = A * S \quad (3.\ 12)$$

Los coeficientes a_{ij} son constantes que cuantifican los pesos de la mezcla y que agruparemos en una matriz A, las cuales se asumen que son desconocidas, ya que estos valores dependen del ruido de las señales, la distancia entre las fuentes de sonido y el micrófono, la cantidad de mezclas entre las señales, etc. En un sistema ICA normal, lo único que se conoce son las señales mezcladas. Este problema es conocido como 'Separación Ciega de Fuentes' (BSS); se usa la palabra a ciegas por el hecho que sólo se cuenta con la mínima información de las señales originales.

En cambio, en nuestro caso tendremos las señales mezcladas y las señales originales, por eso es que es un procesos supervisado, ya que tiene un objetivo específico.

La solución de un problema ICA, es el encontrar la matriz inversa de a_{ij}, la cual llamaremos W_{ij}, tal que nos permita separar las señales $S_1(t)$ y $S_2(t)$.

$$S_1(t) = W_{11}X_1(t) + W_{12}X_2(t)$$
$$S_2(t) = W_{21}X_1(t) + W_{22}X_2(t) \quad (3.\ 13)$$

La solución mediante el uso de análisis de componentes independientes, comienza con un proceso de "blanqueado" de las señales, y usando también el algoritmo de gradiente o de punto fijo (fast ICA). En nuestro caso no abundaremos en esto, debido a que nuestra metodología es supervisada y la búsqueda de la matriz inversa de solución se realiza mediante el uso del ABC.

3.3.1 Descripción de la propuesta para aproximador de funciones mediante ABC.

La solución para este tipo de problemas, el cual llamaremos como aproximador de funciones supervisado, seria:

1.- Contar con las señales objetivo y la mezcla de las señales, en Figura 3. 11 se muestran las señales objetivo (S1= $X * \mathrm{Sin}(2\pi * 1000 * \mathrm{tiempo})$ y S2= $X * \mathrm{Sin}(2\pi * 800 * \mathrm{tiempo})$) y las mezclas (X1 = (S1+ S2*0.5) y X2 = (S2+ S1*0.5)) que corresponden a una proporción de 0.50 y 0.50, respectivamente.

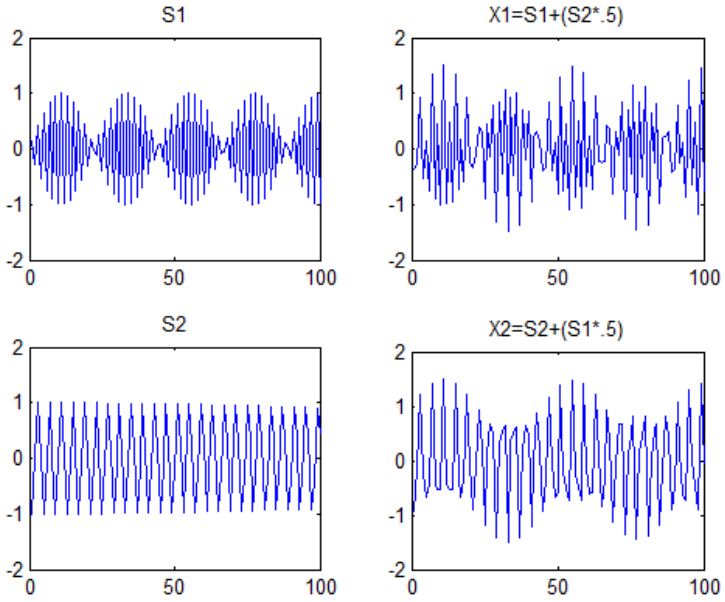


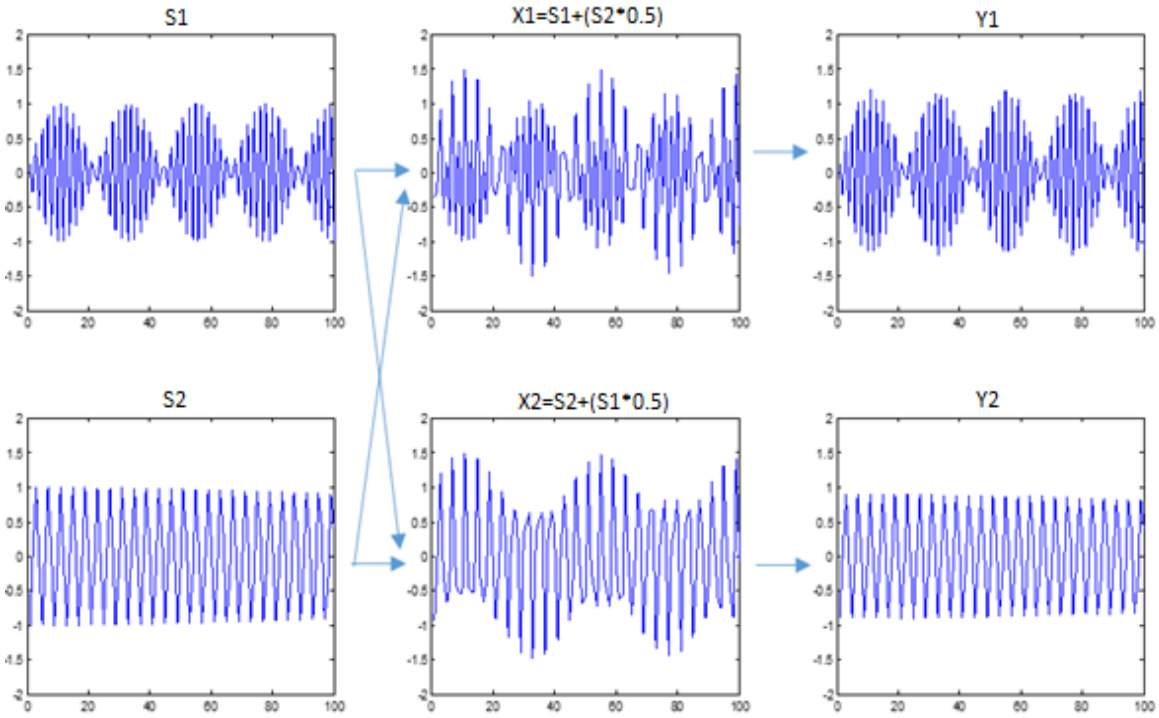
Figura 3. 11 Señales meta (señales originales) y mezclas para aproximador de función supervisado.

2.- Se procede a inicializar el ABC, en este caso se utilizó una colonia de 30 abejas y 500 iteraciones, con la primera condición de que la búsqueda de la primer abeja iniciara en $W = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Esta condición es usada meramente porque el proceso ICA toma este valor como condición inicial, pero gracias al excelente funcionamiento del algoritmo de ABC, no es indispensable; también puede iniciar de una forma aleatoria o heurística. La función a

minimizar (función objetivo) sería la función de error cuadrático medio, Ec. (3. 14), cuyo valor se espera tener lo más pequeño posible.

$$ECM = \frac{1}{n}\sum_{i=1}^{n}(\gamma i - Yi)^2 \qquad (3.\ 14)$$

En la función de ECM, γi corresponde al valor de n predicciones y Yi es el vector de los valores objetivo, recordando que el algoritmo trata de buscar el mínimo. En la Figura 3. 12 se muestran los resultados y las señales separadas por medio de ABC.




```
Iteraciones = 500  Abejas = 30
X inicial= 1   0   0   1
Elapsed time is 0.991181 seconds.
A= 1.5035    -0.70264    -0.59954      1.2007
```

Figura 3. 12 Proceso de recuperación de señales mezcladas, mediante ABC.

3.3 Entrenamiento de redes neuronales mediante ABC.

Las funciones realizadas por el cerebro humano: pensar, recordar y resolver problemas, ha inspirado a muchos científicos en modelar de forma aproximada estos procesos; en el área de las ciencias de la Ingeniería a dado como resultado a las redes neurales artificiales

Las Redes Neuronales Artificiales, ANNs (Artificial Neural Networks) están inspiradas en las redes neuronales biológicas del sistema nervioso humano. Éstas están conformadas por elementos que funcionan de forma semejante a las neuronas biológicas en su forma más básica y su organización es parecida a la que se presenta en la corteza cerebral.

Las ANNs presentan una serie de características propias del sistema nervioso de diversos organismos inferiores y superiores biológicos, como sería el aprender de la experiencia, generalizar de ejemplos previos a ejemplos nuevos y con capacidades de abstracción de las características principales de una serie de datos [25].

- Aprender de la experiencia.- Adquieren el conocimiento por medio del estudio, ejercicio o experiencia, el cual es almacenado, al igual que lo hace el sistema nervioso, en las conexiones inter-neuronales o sinapsis. Las ANNs son capaces de adaptarse de modo de contar con una salida consistente.
- Generalizar.- Las ANNs generalizan (extender o ampliar un concepto, idea o modelos), esto se refiere a que pueden obtener una respuesta correcta a pesar de que las entradas presenten pequeñas variaciones debido a efecto de ruido; se cuenta con una altísima plasticidad y una gran adaptabilidad.
- Abstracción.- Las ANNs son capaces de abstraer (reconocer) la esencia de un conjunto de entradas que aparentemente no presentan aspectos comunes.

Las ANNs, tratan de seguir la “arquitectura modelo” de una neurona bilógica, como sería la de la Figura 3. 13. La mayoría de las neuronas codifican sus salidas como una serie de breves pulsos periódicos, llamados potenciales de acción, que se originan cercanos al soma o cuerpo celular y se propagan a través del axón. Luego, este pulso llega a la sinapsis y de ahí a las dendritas de la neurona siguiente.

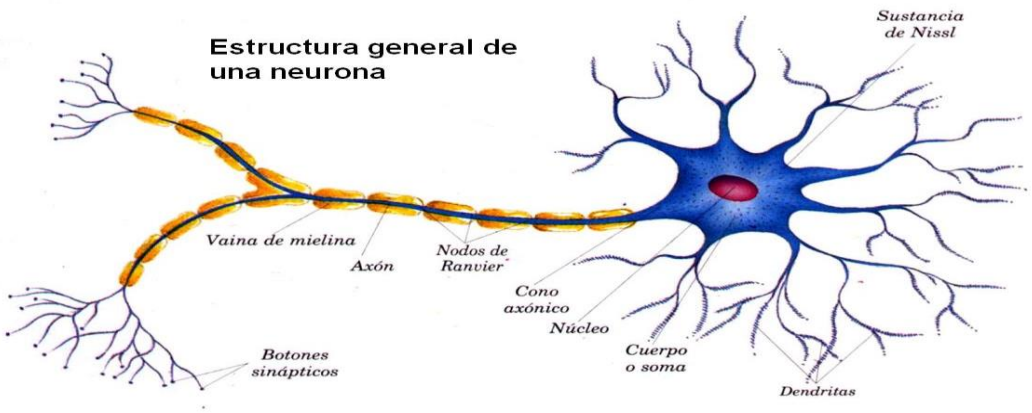


Figura 3. 13 Estructura general de una neurona biológica.

El comportamiento de interés para el desarrollo de una neurona artificial mediante una neurona biológica, seria:

- El impulso que llega a una sinapsis y el que sale de ella no son iguales en general, ya que este pulso depende de la cantidad de neurotransmisores, y esta cantidad cambia durante el proceso de aprendizaje, y es aquí donde se almacena la información. Una sinapsis modifica el pulso, ya sea reforzándolo o debilitándolo.
- En el soma se suman las entradas de todas las dendritas. Si estas entradas sobrepasan un cierto umbral, entonces se trasmite un pulso por el axón.

Las ANNs, tienen un área amplia de aplicación como son las siguientes:

- Análisis y procesado de señales
- Reconocimiento de imágenes
- Control de procesos
- Filtrado de ruido
- Procesado de lenguaje
- Diagnósticos médicos
- Otros

Por lo general, el entrenamiento de las ANNs se realiza por medio de un algoritmo como el de propagación hacia atrás (retro-propagación). Estos algoritmo suelen ser muy complejos y lentos para alcanzar la converger al resultado correcto, tomando largos tiempos de procesamiento.

Por lo cual la se busca facilitar este proceso de entrenamiento con la utilización de algoritmos meta-heurísticos. En nuestro caso se utilizó el ABC para entrenar redes neuronales multi-capa, en la tarea de aproximador de funciones. La arquitectura de la ANN se debe a la complejidad de la función a aproximar, por lo cual se estableció una arquitectura de 2 entradas, 5 neuronas en la primera capa, 2 en la segunda capa y una de salida, de la cual se hablará más adelante.

3.3.1 Redes Neuronales Artificiales.

En las Redes Neuronales Artificiales (ANNs), una neurona es una unidad analógica, la cual tiene varias entradas y las combina normalmente con una suma básica. La suma de las entradas es modificada por una función de transferencia y el valor de la salida de esta función de transferencia se toma como la respuesta de la neurona artificial. Esta salida puede ser conectada a las entradas de otras neuronas artificiales mediante conexiones ponderadas correspondientes a la eficacia de la sinapsis de las conexiones neuronales. La Figura 3. 14 representa una neuronal artificial.

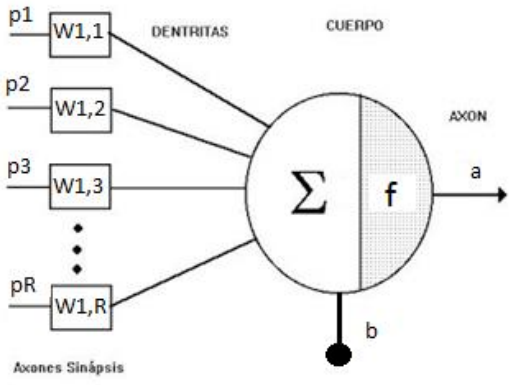


Figura 3. 14 Diagrama de una Neurona Artificial.

En la figura anterior se muestra una neurona artificial con R entradas, y donde cada entrada p1, p2, p3,… pR, cuenta con su propio peso $W_{1,1}$, $W_{1,2}$, $W_{1,3}$,…, $W_{1,R}$, respectivamente, generando la matriz de pesos, por lo cual la ecuación representativa de esta neurona queda de la siguiente forma:

$$a = f(\mathbf{W}p+b) \tag{3. 15}$$

Donde W es una matriz y p es un vector. El bias b es un ajuste propio de cada una de las neuronas. La función de transferencia f es una función no-lineal que tiene como argumento el valor escalar: **Wp**+b. La función f es también una cantidad escalar. Existe una variedad de funciones de trasferencia que pueden ser utilizadas en la ANN, entre las más frecuentes se tienen las que se presentan en la Figura 3. 15.

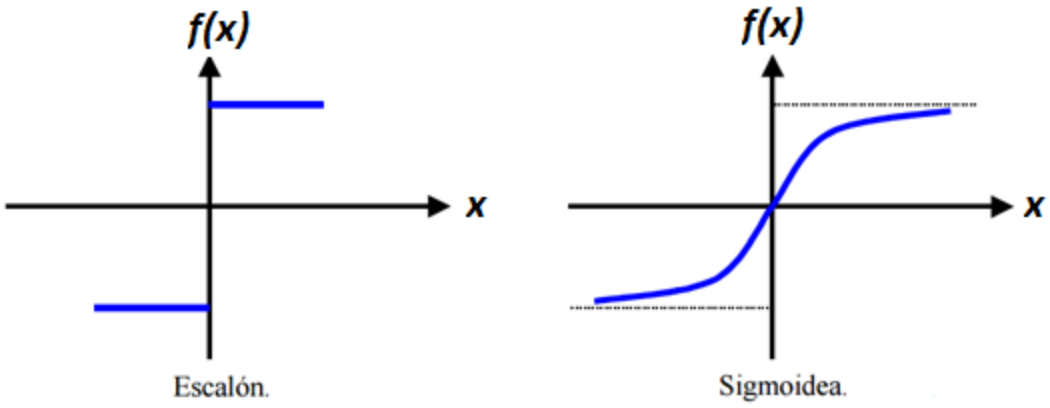


Figura 3. 15 Funciones de trasferencia más usuales en las ANN.

3.3.2 Entrenamiento de redes neuronales multicapa, por ABC.

Las redes neuronales multi-capa están formadas por un conjunto de neuronas organizadas en capas [12], en donde cada capa puede tener n, m o, p, q, etc. cantidad de neuronas, y cuyo número siempre es el necesario para el problema en cuestión, ya que a mayor número de neuronas, se incrementa el tiempo de entrenamiento. La utilización de diferentes capas de neuronas se debe a que una sola capa de neuronas generalmente no es suficiente para solucionar el problema satisfactoriamente; la forma del arreglo sigue una forma piramidal, con un número decreciente de neuronas de la entrada hacia la salida. Las conexiones se denominan conexiones "feedforward" o hacia delante, un ejemplo seria el que se muestra en la Figura 3. 16.

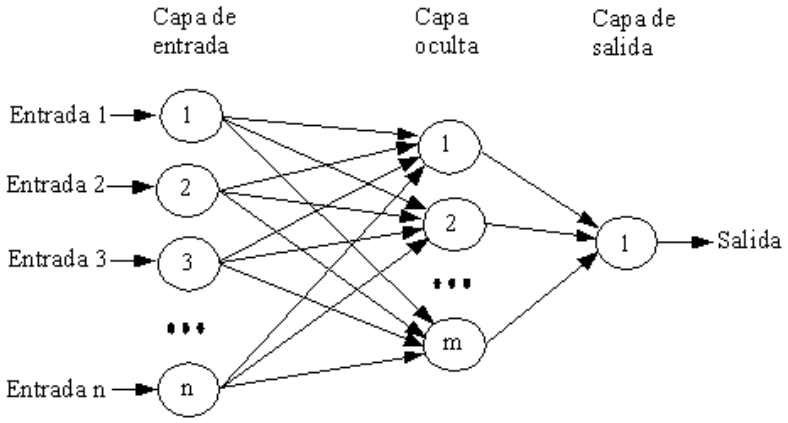


Figura 3. 16 Red neuronal multi-capa.

3.3.1 Descripción de la propuesta para entrenamiento de redes neuronales artificiales mediante el algoritmo de la colonia artificial de abejas.

El funcionamiento de esta ANN es de una forma supervisada, lo que requiere de pares de datos consistente de patrones de entrada y salida correcta. Esta tarea es usualmente utilizada para la predicción, evaluación o generalización; se aprende básicamente a asociar un conjunto de datos de entrada con su correspondiente conjunto de salida.

En la etapa de construcción de la ANN, en la primer capa, se asume que Xi = [X1, X2, … Xn], la cual representa las entradas de la red, W_{ij} corresponde a los pesos sinápticos de las neuronas j asociadas a la entrada i, $Y_{h,j}$ es la salida de la neurona j (j=1,2, …q) en la capa h (h = 1, 2, … ,m); b_{hj} sería el bias, donde la ecuación de la primera capa seria:

$$S_{1,j} = \sum_{i=1}^{n} \left(W_{i,j} * X_i + b_{1,j} \right) \ldots \quad (3.\ 16)$$

La sumatoria de las entradas de la neurona, pasan a una función de activación o función de trasferencia no lineal. En este caso se utilizó la función de trasferencia tangente hiperbólica que se satura entre -1 y +1. En la Figura 3. 17 se muestra gráficamente la función de trasferencia.

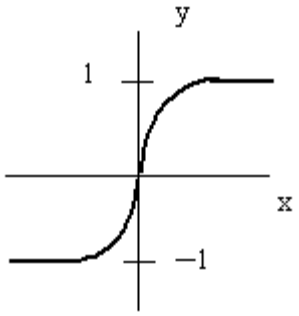


Figura 3. 17 Función de trasferencia sigmoidea tangente.

Por lo cual la salida de la neurona de la primera capa quedaría de la siguiente forma:

$$Y_{1,j} = f(S_{1,j}) = f\left(\sum_{i=1}^{n}(W_{i,j} * X_i + b_{1,j})\right) \ldots \quad (3.\ 17)$$

En la siguiente capa subsecuente, se realiza un proceso semejante donde la salida de la capa anterior serán las entradas de ésta; esto se puede expresar con las siguientes ecuaciones.

$$S_{h,j} = \sum_{i=1}^{m}(W_{k,j} * X_k + b_{h,j}) \ldots \quad (3.\ 18)$$

$$Y_{h,j} = f(S_{h,j}) = f\left(\sum_{i=1}^{m}(W_{k,j} * Y_k + b_{h,j})\right) \ldots \quad (3.\ 19)$$

Donde $k=j^{(h-1)}$ denota a la neurona de la capa anterior (h-1), y $W_{k,j}$ es el peso sináptico asociado entre las neuronas k y j.

El siguiente problema fue tomando de la referencia [26], donde se realizó el entrenamiento de la ANN, el cual se realiza mediante del algoritmo de back-propagation. Como alternativa, se hace ahora el entrenamiento mediante al ABC.

La tarea que se desea para la ANN es la de aproximador de función, donde la función a aproximar es:

$$Z = 4xe^{-4(x^2+y^2)}$$

En la siguiente Figura 3. 18 se muestra esta función graficada tridimensionalmente, dentro de los intervalos x = [-1,+1], y = [-1,+1].

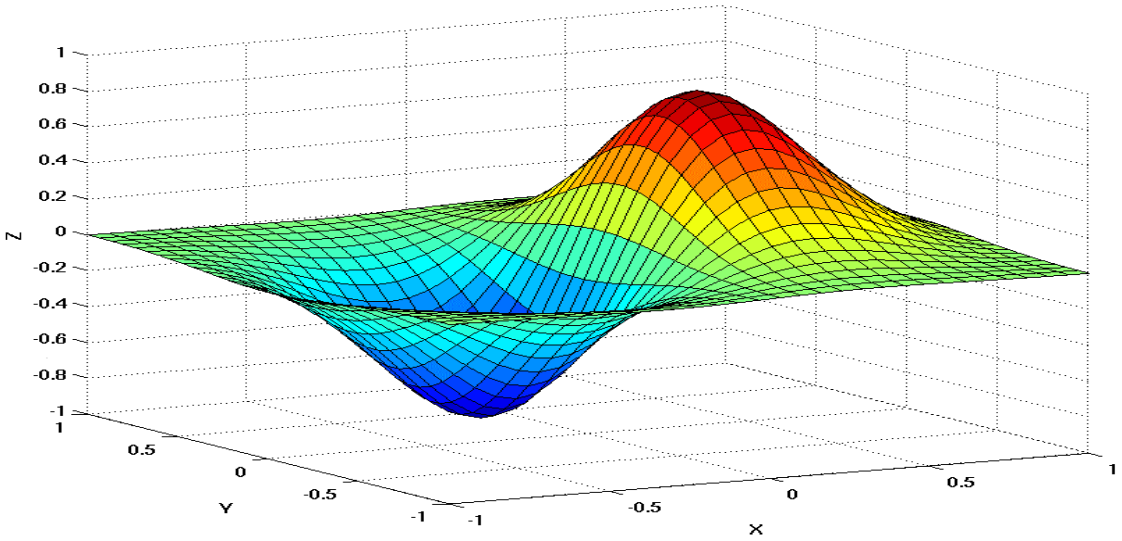


Figura 3. 18 Función aproximar, $Z = 4xe^{-4(x^2+y^2)}$.

La arquitectura propuesta para la ANN, para esta red con dos entradas, fue:

- 5 neuronas en la primera capa.
- 2 en la segunda capa
- 1 en la última capa

Esquemáticamente, esta red se puede representar como el diagrama de la Figura 3. 19.

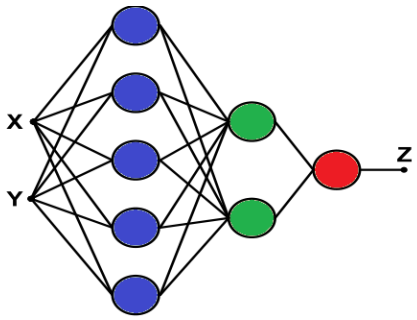


Figura 3. 19 Arquitectura de la ANN 2-5-2-1.

La función objetivo utilizada en el ABC, fue la función de error cuadrático medio (ECM). El algoritmo 1.3 presenta el proceso para la obtención de los pesos sinápticos de la red, que equivale al proceso de entrenamiento.

Algoritmo 1. 9 Pseudocódigo del cuerpo del Algoritmo de Colonia Artificial de Abejas (ABC).

- ~ Inicializar las variables (serán 30 variables, que corresponden a los pesos sinápticos y bias, para la arquitectura propuesto de la ANN), algoritmo 1.4.

- ~ Evaluación (se hace uso de ECM, para saber cuánto se parece la respuesta de la red a la funciona a aproximar)

For Iter = 1 hasta IterMax **do**
Cada modificación a las variables para aproximar más la respuesta de la red, se compara con la función aproximar, esto en todas las fases.

- ~ Fase de Abejas Empleadas, algoritmo 1.5.
- ~ Fase de Abejas Observadoras, algoritmo 1.6.
- ~ Fase de Abejas Exploradoras, algoritmo 1.7.
- ~ Memorización de las mejores soluciones.

End For

Los resultados obtenidos mediante este proceso de entrenamiento y la función a aproximar antes mencionada, se muestran al Figura 3. 20. El gráfico de color rojo con amarillo corresponde a la función a aproximar y el conjunto de puntos corresponde a la respuesta de la ANN. La diferencia entre ambos gráficos corresponde a ECM = 0.05.

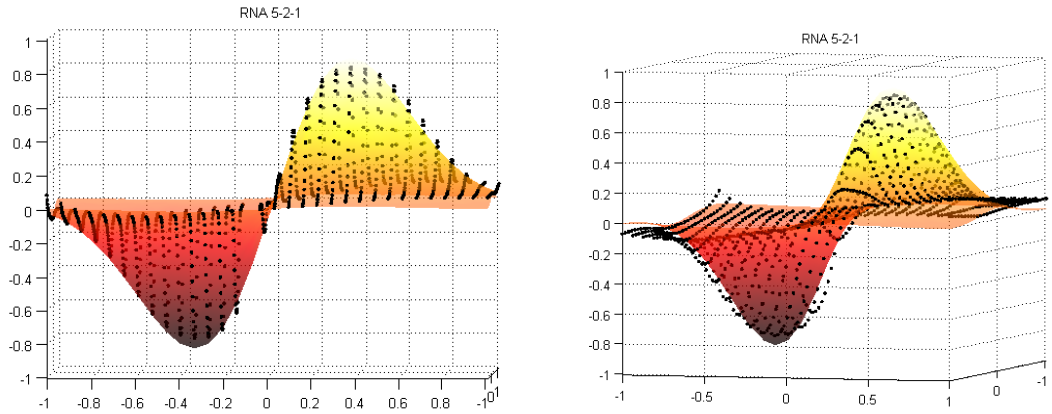


Figura 3. 20 Gráficos de comparación de la función aproximar con la respuesta de la ANN.

3.4 Conclusiones.

En este capítulo se realizó una presentación de las diferentes tareas resueltas por los algoritmos meta-heurísticos, donde se pudo comprobar la eficiencia y versatilidad de estos mismos. Entre las cualidades de los algoritmos meta-heurísticos es que pueden entrar en acción donde los algoritmos de optimización clásica fallan.

La primera tarea resuelta en este trabajo de tesis fue la búsqueda de mascarillas de una CNN para el procesamiento de imágenes binarias. Esta metodología ya fue presentada en [20] donde el algoritmo de optimización utilizado fue el algoritmo simplex; mediante el algoritmo de ABC se logró concluir en los mismos resultados.

Después de ésta se optó por continuar con el procesamiento de imágenes, pero más complejas, por lo cual se buscó realizar segmentación de imágenes, haciendo uso de otro algoritmo de optimización meta-heurísticos que fue el algoritmo de PSO, de los cuales se obtuvieron resultados favorables que se pueden comparar con algoritmo de segmentación más complejos.

También se logró resolver problemas de tipo ICA, pero de una forma supervisada, mediante el algoritmo de ABC. Lo que dio hincapié a continuar explotando las aplicaciones de los algoritmos meta-heurísticos, realizando el entrenamiento de redes neuronales artificiales, de las cuales se logró obtener resultados favorables, pero que es necesario el continuar investigando para mejorar este proceso.

Capítulo 4. Resultados de la síntesis de algoritmos Meta-heurísticos en una FPGA por medio de VHDL.

En este capítulo se describen los aspectos relacionadas con el diseño e implementación de los algoritmos de inteligencia colectiva: Algoritmo de la Colonia Artificial de Abejas (ABC) y Algoritmo de Cúmulo de Partículas (PSO) en un FPGA (Field Programmable Gate Array) dispositivo: XC6SLX45 del kit Spartan-6. En la etapa de diseño, se describen las técnicas utilizadas para poder llevar a cabo la migración de software (descritas en el capítulo anterior) a hardware de los algoritmos de inteligencia colectiva. Por otro lado, en la etapa de implementación se describe la manera y el lenguaje empleado para la programación del FPGA, iniciando con los elementos clave para poder realizar la implementación y después, una descripción de la estructura del PSO y ABC.

Las tareas aplicadas en la FPGA, serían:

- Segmentación de imágenes.- En este caso se utilizaron únicamente números naturales, y se generaron módulos para comunicación (recepción y transmisión), entre el Matlab (R2014a) y la tarjeta FPGA.

- Resolución del problema tipo ICA de forma supervisada.- En este caso se hizo uso de números racionales, mediante el uso de punto fijo.

La síntesis y generación de los modelos de simulación se llevaron a cabo en el programa de ISE de Xilinx, en la versión 14.7 (nt64) y las simulaciones fueron realizadas en el programa ModelSim de ModelTech, versión 10.0c.

Las características principales del FPGA son [27]:

- Bloques lógicos configurables, son los principales recursos para la implantación de circuitos lógicos, tanto síncronos como combinatorios. Cada bloque lógico configurable está conformado por dos pares de capas, cada una contiene un generador de funciones lógicas (LUT, Look-Up Tables), los cuales pueden ser utilizados como generadores de funciones o memorias. Además contiene elementos para almacenamiento, compuertas aritméticas, multiplexoras, etc.

- Bloques de Entrada/Salida (IOB, Input/ Output Blocks), que controlan el flujo de datos entre las terminales y los elementos internos del FPGA.

- Multiplicadores. Estos bloques aceptan como entrada dos números de 16 bits, teniendo como salida un número de 32 bits, que representa el producto de los dos números.

La descripción de los módulos de procesamiento, se pude realizar mediante la descripción de funcionamiento y características, esto debido a la versatilidad y sencillez del lenguaje VDHL, que puede ir desde una descripción algorítmica o estructura lógica según sea necesario.

VHDL (unión de los conceptos: HDL ó Hardware Description Language, y VHSIC ó Very High Speed Integrated Circuit), siendo un lenguaje de descripción de hardware, lo que significa que describe el comportamiento de un circuito electrónico de lógica. La forma de comportamiento lógico específico por diseño es independiente del hardware o plataforma en donde se implementará.

4.1 Elementos estructurales de los algoritmos meta-heurísticos para su implementación en hardware (FPGA).

Algunas de las características que hacen que el algoritmo de PSO y ABC sea apropiado para la implementación sobre FPGA, son:

- Proceso cíclico o iterativo: El PSO y el ABC, se basan en procesos iterativos, por lo cual la implementación en una FPGA acelerará el algoritmo.

- Tamaño de memoria: En la implementación del PSO y del ABC, se hace uso de la memoria de una forma heterogénea, por lo cual en la implementación en hardware se tendrá un manejo de acceso a memoria balanceado.

- Funcionamiento paralelo.- El funcionamiento de FPGA se caracteriza por llevar en tiempo real el proceso de forma paralela, lo que hace disminuir el tiempo de procesamiento de los algoritmos.

La implementación del algoritmo meta-heurístico tiene algunas dificultades, porque:

- Las operaciones demandan gran cantidad de multiplicaciones; requiriendo multiplexar este proceso, para que pueda ser implementado.

- Los algoritmos meta-heurísticos se caracterizan por trabajar con variables aleatorias, por lo cual se utilizan registro de desplazamiento con retroalimentación lineal (LFSR – linear feedback shift register source), para poder conseguir una aleatorización de las variables.

- La cantidad de operaciones, que requieren ser guardadas en una variable, es grande y genera que las LUT puedan llegar a saturarse. Es necesario utilizar una programación de estilo funcional, para disminuir el uso de éstas y poder acelerar el proceso.

A continuación, se hace una breve descripción de los módulos fundamentales y con mayor importancia, en el diseño lógico para la implementación del ABC y PSO.

4.1.1 Multiplexores.

Un multiplexor, es un dispositivo que permite dirigir la información digital procedente de diversas fuentes hacia un destino único por un canal.

Los multiplexores se diseñan describiendo su comportamiento mediante la declaración **with-select-when** o mediante ecuaciones booleanas. En la **Error! Reference source not found.** a) se observa un multiplexor de entradas a, b, c y d. Cada entrada representa una palabra de 2 bits (a1, a0), (b1, b0), (c1, c0) y (d1, d0).

También posee una entrada de selección de datos (s1 y s2), que permiten elegir los datos digitales provenientes de cualquier entrada hacia las líneas de salida (Z1 Y Z2).

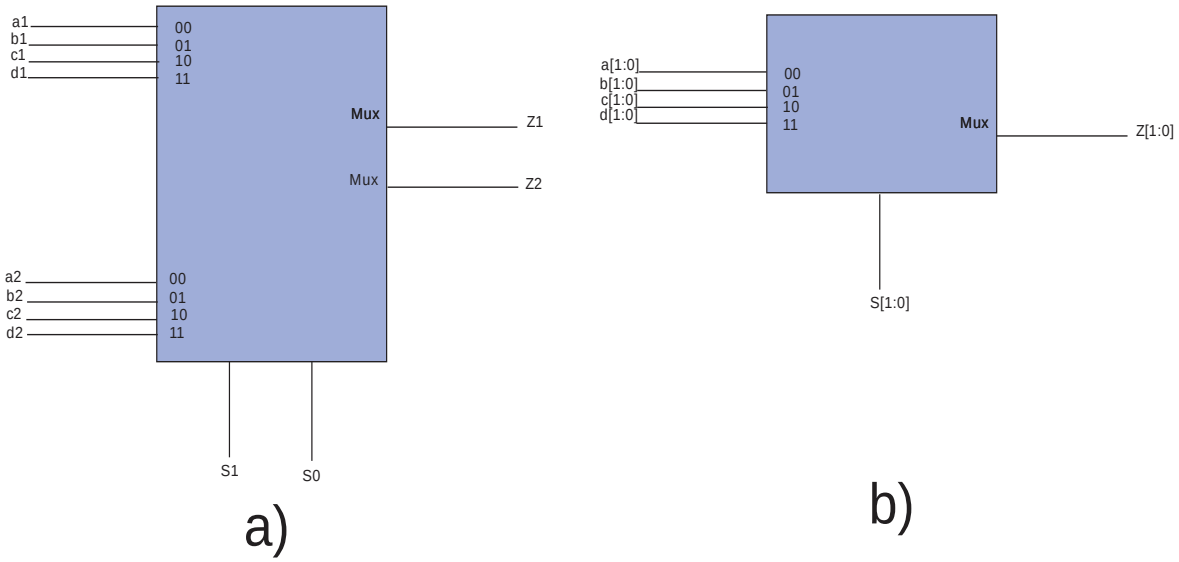


Figura 4. 1 a) Multiplexor de 4 bits. b) Multiplexor de vectores.

En la **Error! Reference source not found.** b), se muestra un diagrama simplificado que resalta la representación mediante vectores de bits. Esta estructura es básica para la implementación de los algoritmos meta-heurísticos para un buen manejo de recursos, como es la memoria y los multiplicadores.

4.1.2 Diseño de sistema secuencial síncrono.

El diseño lógico secuencial síncrono [28], será la base para nuestro programa de implementación en el FPGA de los algoritmo meta-heurísticos, debido a que éstos se encargarán de mantener la secuencia y el control de los elementos de memoria (flip-flops y LUTs).

Un flip-flop es un elemento de memoria, que se puede utilizar tanto en sistemas síncronos como asíncronos. La característica principal de éstos, es el mantener o almacenar un bit de manera indefinida hasta que a través de un puso o una señal cambie de estado.

Se le llama síncrono, debido a que cada elemento de memoria se encuentra conectado a la misma señal de reloj, de tal forma que sólo se producirá un cambio de estado en el sistema, cuando ocurra un flanco de disparo o un pulso en la señal de reloj.

En la implementación se utilizó la estructura de Moore (**Error! Reference source not found.**) debido a que sólo la señal de salida depende del estado en que se encuentra.

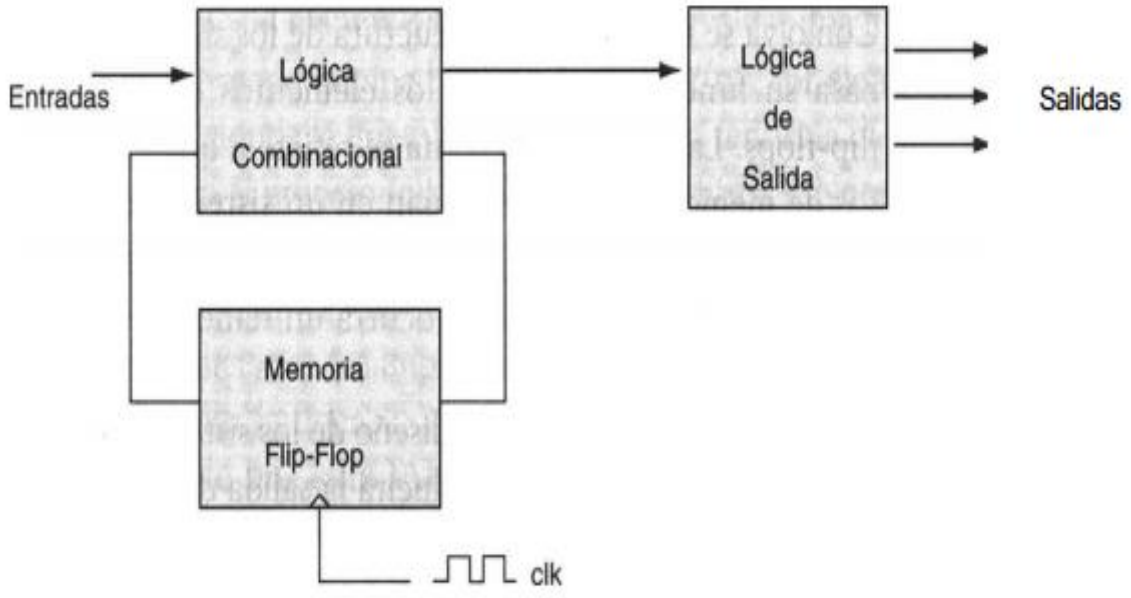


Figura 4. 2 Arquitectura secuencial de Moore

4.1.3 Registro de desplazamiento con retroalimentación lineal, para la generación de números aleatorios.

Uno de los procedimientos para generar números aleatorios es el empleo de registros de desplazamiento con realimentación lineal (linear feedback shift register - LFSR) [29].

Éste proporciona un buen desempeño en la aleatoriedad. Genera una secuencia pseudo-aleatoria de n-bits de 2^n-1 posibles valores, donde n es el número de registros del registro de desplazamiento, donde cada registro del registro de desplazamiento se recorre hacia la derecha una posición en cada ciclo de reloj.

Este bit de entrada al registro de desplazamiento, es el resultado de realizar la operación lógica XOR (o XNOR) entre ciertos y determinados bits del registro de desplazamiento, obteniendo los valores lógicos a la salida de cada registro de desplazamiento y uniéndolos de modo de formar un vector de bits (**Error! Reference source not found.**).

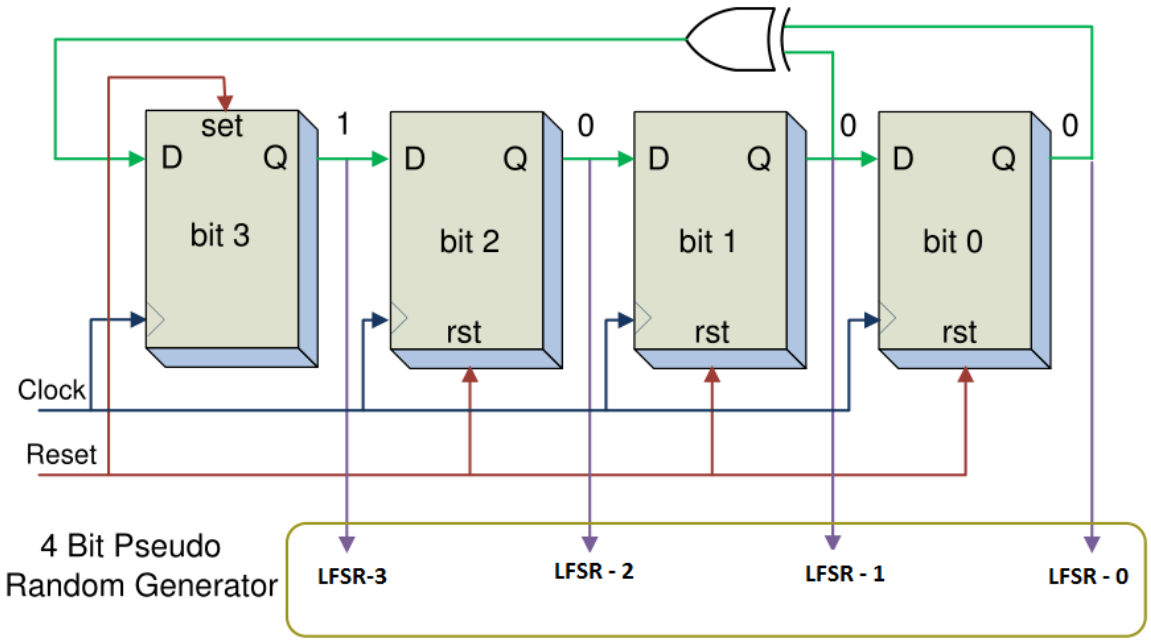


Figura 4. 3 Registros de desplazamiento con realimentación lineal, 4 Bits.

Los resultados de la simulación del código VHDL (Anexo B), que describe un LFSR de 4 bits se puede observar en la Figura 4. 4.

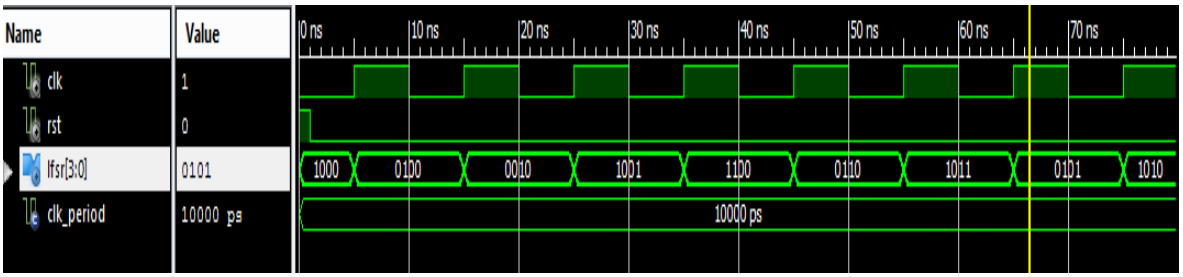


Figura 4. 4 Resultado de la simulación de un LFSR de 4 bits.

4.1.4 Complemento a dos y representación fraccionaria de números binarios en formato de punto fijo.

Se opta por usar el complemento a dos y las operaciones aritméticas en un solo subtema, debido a que las dos están relacionadas para la expresión de números negativos (complemento a 2) y fracciones (operaciones de punto fijo), en una forma binaria para que pueda ser interpretada por el FPGA y por consiguiente desarrollada.

- Complemento a dos:

  La importancia del complemento a 2 de los números binarios, es debido a que nos permite la representación de números negativos.

  Ejemplo.-

Complemento a dos	**Decimal**
0111 $_{Ca2}$	7
0110 $_{Ca2}$	6
0101 $_{Ca2}$	5
0100 $_{Ca2}$	4
0011 $_{Ca2}$	3
0010 $_{Ca2}$	2
0001 $_{Ca2}$	1
0000 $_{Ca2}$	0
1111 $_{Ca2}$	-1
1110 $_{Ca2}$	-2
1101 $_{Ca2}$	-3
1100 $_{Ca2}$	-4
1011 $_{Ca2}$	-5
1010 $_{Ca2}$	-6
1001 $_{Ca2}$	-7

El cálculo de complemento a dos es muy sencillo de realizar mediante compuertas lógicas, donde el bit más significativo tendrá la tarea de representar si el número binario es positivo (0) o negativo (1), por lo cual el rango de valores será diferente al de una representación binaria habitual, la cual podría ser representada como:

$$-2^{n-1} \leq Rango \leq 2^{n-1} - 1$$

Una forma sencilla y rápida para hacer una conversión de números binarios a complemento a dos es comenzar con el bit menos significativo, copiando el número original hacia el bit más significativo hasta encontrar el primer 1; luego de haber encontrado y copiado el primer 1, se niega (cuando es 0 se convierte en 1 o viceversa) los dígitos restantes. Este método es rápido y sin necesidad de usar complemento a 1 y sumarle 1, en el caso de los números positivos binarios, mantendrá su estructura binaria.

Por ejemplo:

Supongamos que tenemos el número binario 11111 (31 decimal) y vamos a usar números binarios con signo, por lo cual es necesario utilizar el complemento a 2. Éste nos dice que si el bit más significativo es 1, entonces estamos hablando de un número negativo, y para saber el valor de este número podemos usar la metodología antes mencionada.

$$11111_{Ca2} \rightarrow 00001 = -1$$

Donde el resultado es que $11111_{Ca2} = -1$.

- Representación fraccionaria de números binarios en formato de punto fijo.

  Hasta ahora se ha visto la representación binaria tanto para números positivos como negativos. Sin embargo, para los algoritmos meta-heurísticos es necesario el poder operar como números fraccionaros por lo cual es necesario trabajar con punto fijo.

  En el formato de punto fijo a utilizar, éste tiene las siguientes características:

  1. El bit más significativo representará si el número es negativo (1) o positivo (0) en complemento a 2.
  2. Se utilizan 4 bits en complemento a 2 para la parte entera de un número.
  3. 11 bits, para la parte fraccional en complemento a 2.

  En la Figura 4. 5 Vector de bit. S representará negativos o positivos binarios; E, números enteros binarios y f la parte fraccionaria.se presenta el vector de bits que forman el número binario con punto fijo y complemento a dos.

Figura 4. 5 Vector de bit. S representará negativos o positivos binarios; E, números enteros binarios y f la parte fraccionaria.

La mínima resolución de este formato sería:

$$\frac{1}{2^{11}} = 0.000488281$$

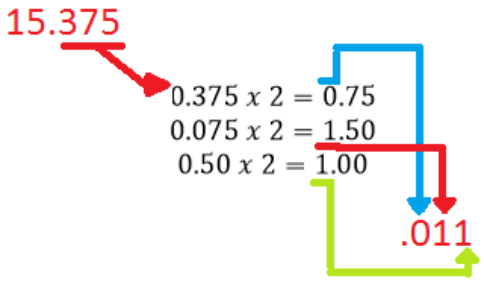
La posición del punto es seleccionada según nuestras necesidades, pero el punto binario es simbólico, ya que no ocupa ningún bit.

Un ejemplo de la conversión de un número decimal fraccionario a binario, seria:

1. Si el número decimal fraccionario es positivo se pasa al paso 2, de lo contrario se obtiene el complemento a 2.
2. Se transforma la parte entera de igual manera que un número decimal entero a binario.

Suponiendo que tenemos el número 15.375, la parte entera correspondería a 15 → 01111_{bin}.

3. La parte fracciona la multiplicamos sucesivamente por 2 hasta llegar a 0 o hasta el número de dígitos del vector de bits.



4. El resultado tanto del número binario correspondiente a enteros como el número binario fraccionario se unirán para formar el vector.

15.375 → 01111.01100000000

El uso de las operaciones aritméticas básicas como suma y resta, no presentan problemas debido a que el vector de bits mantiene su orden de 16 bits, en cambio en la multiplicación se presenta que $B_m \times B_n = B_{m+n}$, donde m y n son los bit del vector, por lo cual el resultado puede y debe ser truncado para poder representarlo

en 16 bits (Figura 4. 6). Debido a esto se debe tener cuidado con la multiplicación, ya que es fácil que ocurra un sobreflujo, donde sólo se tomarán los bits 27 al 12, los cuales corresponderán a nuestro vector de 16 bits.

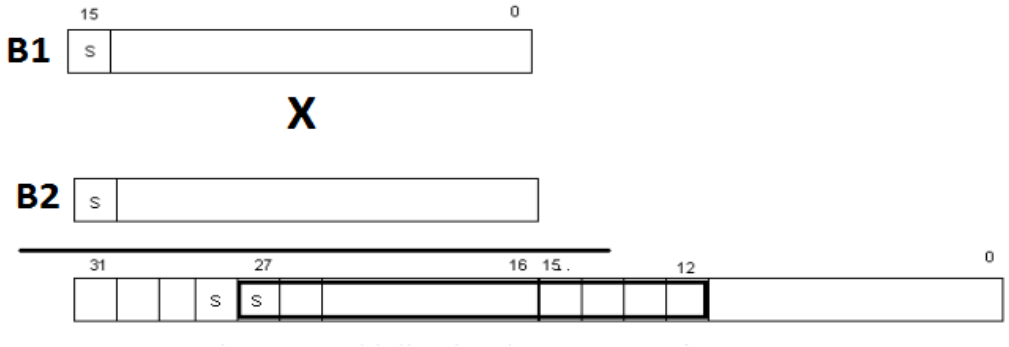


Figura 4. 6 Multiplicación de 2 vectores de 16 bits, en el cual el resultado es un vector de 32 bits, y éste es truncado del bit 27 al 12.

4.2 Algoritmo de PSO orientado a hardware (FPGA) para la tarea de segmentación de imágenes.

Para la implementación exitosa del PSO en un FPGA, es necesario seguir una metodología, la cual nos permite planificar y ordenar el desarrollo de PSO.

En la Figura 4. 7, se muestra un diagrama a bloques de la metodología de diseño que se seguirá para implementar el PSO en un FPGA.

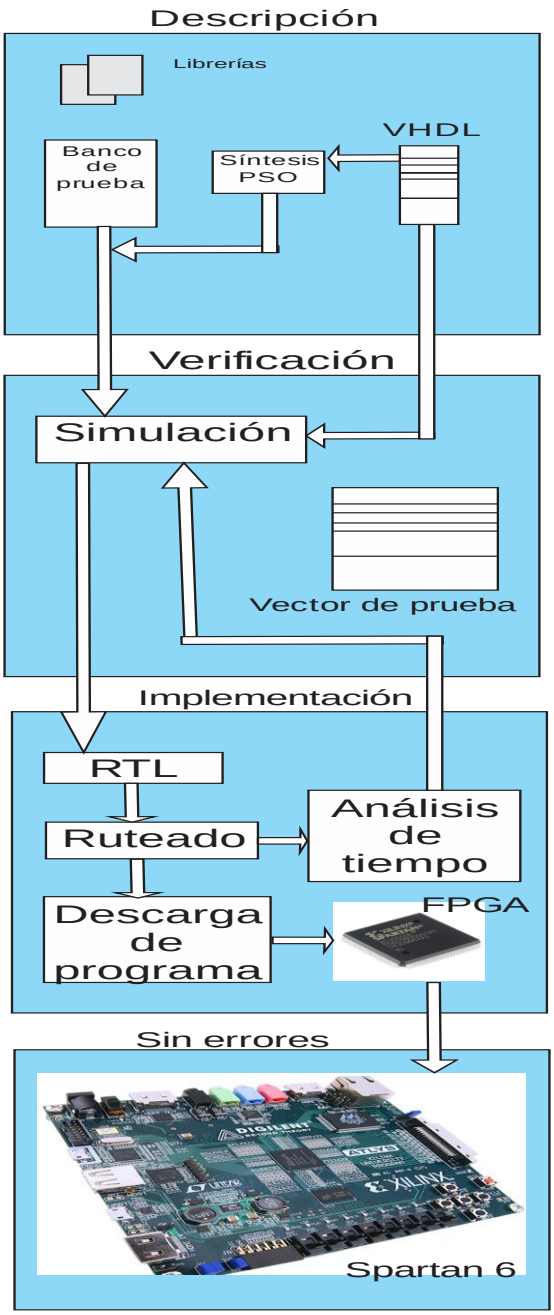


Figura 4. 7 Diagrama a bloques de la metodología para implementación de PSO.

En la síntesis del PSO:

1.- Se generaron bancos de memoria de diferentes histogramas, para poder hacer pruebas del funcionamiento del PSO.

2.- Se estructura una máquina de estados, de tal modo que se pudieran sincronizar los procesos y evitar el desbordamiento de datos.

3.- Verificación en simulación del correcto funcionamiento del PSO.

4.- Generación de comunicación entre Matlab y FPGA, de modo de enviar el histograma de diferentes imágenes y el PSO pueda realizar la umbralización de las imágenes.

La máquina de estado realizada para el PSO se muestra en la Figura 4. 8, donde se observa la rutina completa a seguir y los estados necesarios para realizar correctamente cada proceso.

En este tipo de sistemas secuenciales, la salida no sólo depende de la entrada sino que también del estado interno, por lo cual se podría controlar fácilmente a qué estado tendría que cambiar.

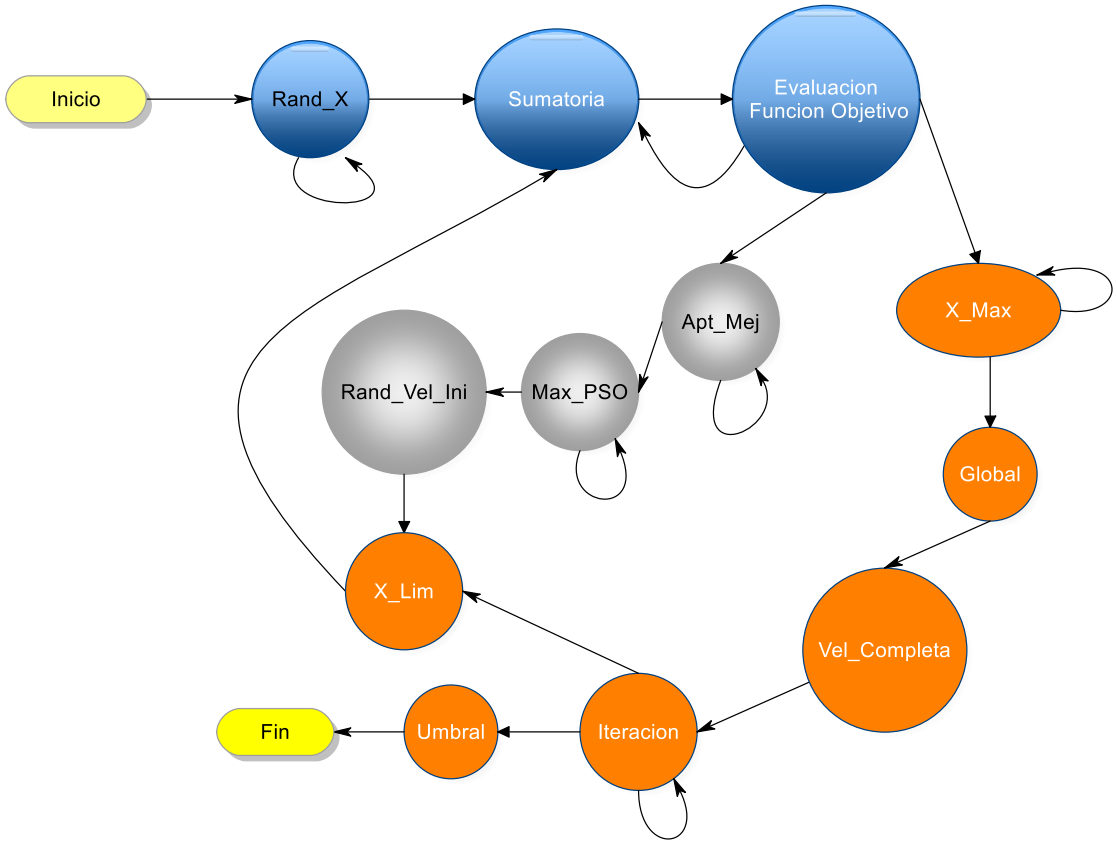


Figura 4. 8 Máquina de Estado PSO.

La máquina de estados mostrada en la Figura 4. 8, fue usada para realizar el algoritmo PSO y hacer pruebas con base de datos fijos. Después de comprobar su correcto funcionamiento, se realizaron dos módulos: el de recepción y el de trasmisión por medio de UART, de los cual se hablará más adelante. En la siguiente Tabla 4.1, se resumen los estados de la máquina de estados antes mencionada.

Tabla 4.1. Resumen de la máquina de estado del PSO.

Estado VHDL	Descripción
Rand_X	En este estado se generan las variables aleatorias correspondientes a cada partícula, usando los registros de desplazamiento con realimentación lineal.
Sumatoria	Aquí se genera la primer parte de la función de error cuadrático medio, la cual sería la sumatoria $\sum_{i=1}^{n}\widehat{(Y_i} - Y_i)^2$.
Evaluación Función Objetivo	Una vez terminandas las sumatorias, se obtiene la ECM completa, que sería nuestra función de fitness. $$\mathrm{ECM} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2.$$ Esto se repite según la cantidad de partículas utilizadas.
Apt_Mej	El estado de Apt_Mej corresponde a uno de los estados iniciales encargados de plantear las condiciones iniciales necesarias para el PSO, donde su funcionamiento es el de guardar los resultados de la función de fitness y las posiciones de las partículas, como la mejor aptitud y la mejor posición.
Max_PSO	Se realiza una comparación entre todas las partículas para obtener la mejor global.
Rand_Vel_Ini	Se genera aleatoriamente el vector de velocidad, correspondiente a cada una de las partículas, la cual se suma al vector de la posición de la i-ésima partícula para generar una nueva posición.
X_Lim	El nuevo vector de la posición es limitado por el valor máximo y mínimo que puede alcanzar la partícula.
X_Max	Una vez teniendo todas las condiciones iniciales, inicia el ciclo iterativo del PSO, donde después de evaluar la función de fitness, se vuelven a comparar los resultados de las partículas pasadas con las nuevas, para ver si mejoró su función de fitness. En el caso de que sí mejoren, se sustituyen los valores anteriores por los mejores actuales.

Global	Se analizan todas las partículas y se selecciona la mejor de la nube con respecto a su función de fitness.
Vel_Completa	Se genera un nuevo vector de velocidad, con la ecuación característica del PSO (ec. 1.3).
Iteración	Este estado, solamente es un contador de iteraciones, el cual tiene la tarea de terminar, el proceso según el número de iteraciones establecidas.
Umbral	Una vez alcanzado el número de iteraciones, se arroga el mínimo global que corresponderá al Umbral de la imagen a segmentar.

En la Figura 4. 9, se pueden apreciar los resultados del algoritmo PSO, usando como imagen a umbralizar la Figura 3.21, donde el vector de "X" corresponde a las partículas de la nube, "d_bus" a los estados de la máquina de estado y el resultado buscado será el umbral global que tiene un resultado de 39, el cual es igual a los resultados obtenidos en software, para esta figura.

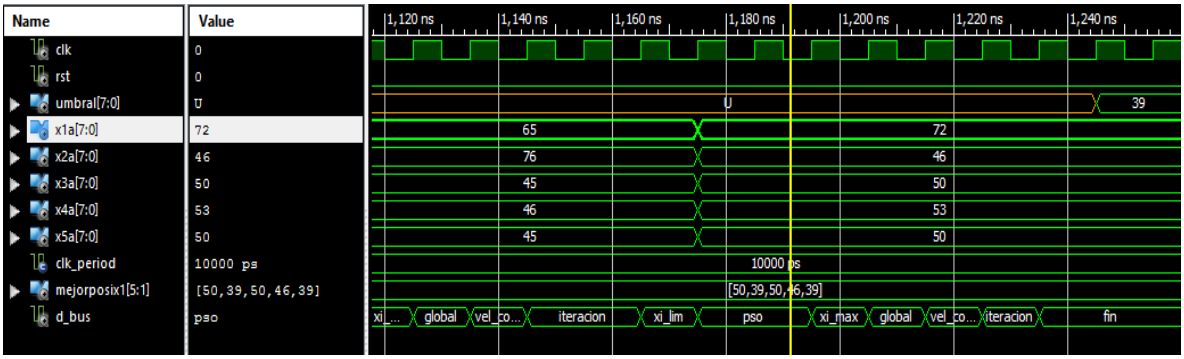


Figura 4. 9 Simulación PSO.

Diseño de UART.

Realizada la implementación correcta y sin errores del algoritmo de PSO, se generan la comunicaciones entre Matlab mediante el uso de UART (Universal Asynchronous Receiver and Transmitter), el cual es un dispositivo (RS-232) que envía datos paralelos sobre una línea serie.

El RS-232 (Recommended Standard 232, también conocido como Electronic Industries Alliance RS-232C), es una interfaz que tiene como función el intercambio de una serie de datos binarios entre equipo terminal de datos y un equipo de comunicación de datos. En la tarjeta SPARTAN 6, el puerto correspondiente al UART, se encuentra en una terminal micro USB (Figura 4. 10), ésta permite la comunicación con el puerto COM de una PC.

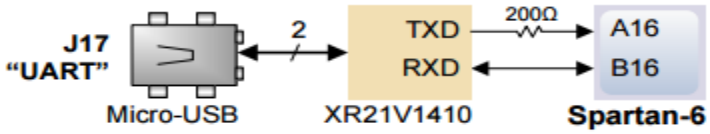


Figura 4. 10 USB-UART (puerto serial), Spartan- 6.

El UART incluye un trasmisor y un receptor de datos (Figura 4. 11), donde el trasmisor es un registro de corrimiento que carga los datos a trasmitir en paralelo. Estos datos son trasmitidos uno a uno, a una velocidad determinada (baud rate), empezado por el bit menos significativo.

La velocidad de trasferencia y recepción, entre Matlab y FPGA, tiene que ser igual tanto en el receptor como en el trasmisor (115,200 bauds por segundo, velocidad máxima permitida por la FPGA), pero esto se realiza de una forma asíncrona, debido que no hay una relación de tiempo entre ambas. La razón de transmisión (baud rate) antes mencionada generará un pulso de 16 veces; esta señal no se comporta como un reloj sino como un habilitador, porque valdrá "1" durante todo el ciclo de reloj y luego cambiará su valor a cero.

Otra consideración a tomar es que Matlab trabaja con código ASCII, por lo cual se tiene que convertir a binario los datos que se quieran trasmitir y que puedan ser interpretados por la FPGA. Por lo tanto, se debe hacer un decodificador de Binario a ASCII para que Matlab pueda definir la trama recibida por la FPGA.

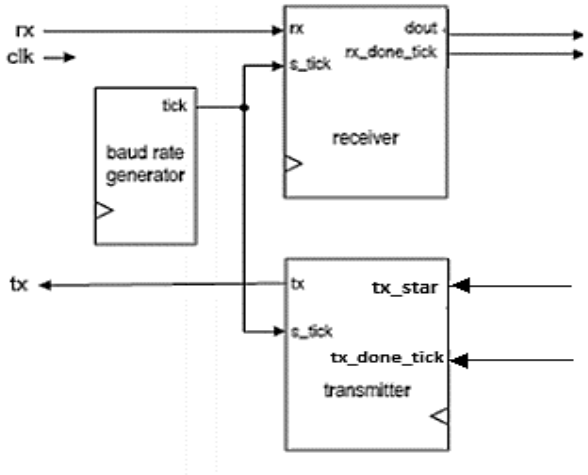


Figura 4. 11 Diagrama a bloques del UART.

Módulo de recepción.

La implementación de este módulo en VHDL es muy sencilla. Se puede realizar mediante una máquina de estados, que cuando tenga presente un evento cambie al siguiente bit hasta generar la trama de 8 bits y volver a iniciar una nueva trama. El receptor recibe los bits uno a uno y los re-ensambla en un vector de bits, hasta encontrar el bit de paridad para detenerse (Figura 4. 12).

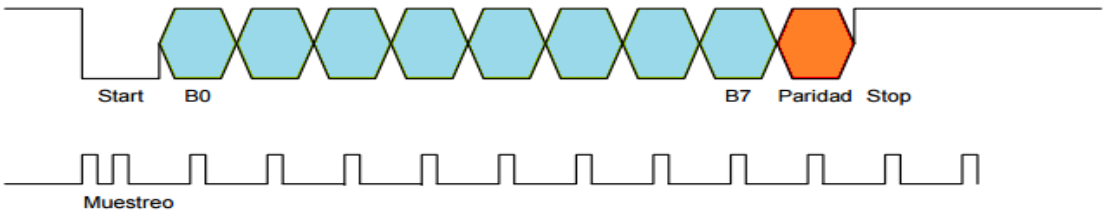


Figura 4. 12 Recepción y trasmisión de Bits.

En la Figura 4. 13, podemos observar los bloques pertenecientes al módulo de receptor, donde se puede apreciar lo siguiente:

- La máquina de estado del receptor, la cual se encarga de secuenciar los procesos propios de la recepción.
- Contador de Baudios, el cual será el encargado de ajustar la frecuencia de recepción.
- Contador de bits que llevará la lógica y secuencia de los bit recibidos.

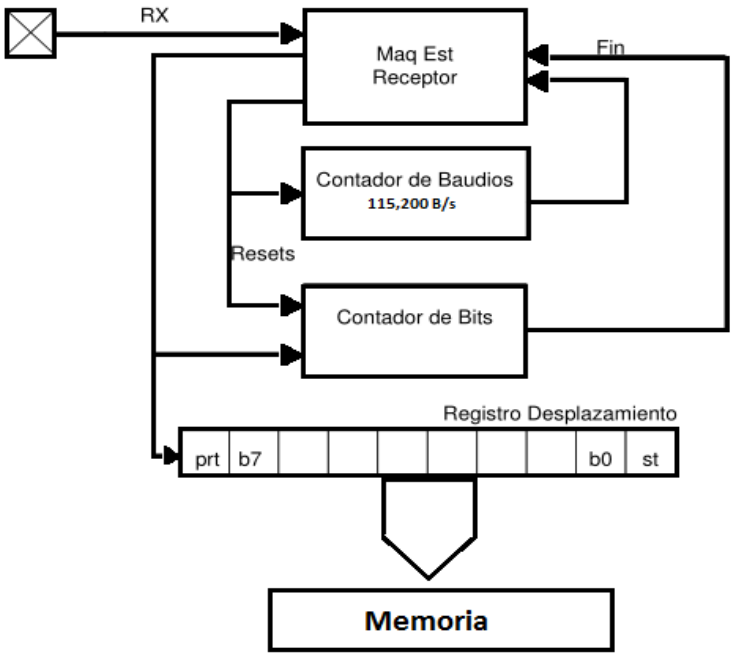


Figura 4. 13 Diagrama de bloques de una arquitectura del receptor.

Su funcionamiento básico, sería:

- Se recibe el bit de inicio (Start), en el que se pone la línea RX a cero.
- El contador se pone a cero y utiliza una señal de reloj en alto para muestrear la recepción.
- Se han de contar 9 bits (8 + paridad).
- Se realiza este proceso según sea necesario.

Módulo de trasmisión.

La sistematización del módulo de trasmisión es similar al módulo de recepción, el cual consiste en:

- Un Registro de Corrimiento que envía hacia afuera uno a uno los bits a una determinada velocidad (baud rate).
- Como aquí no hay muestreo de bit, la frecuencia de trasmisión suele ser más lenta que la de recepción.
- Usa el mismo generador de baudios que el receptor, con la diferencia que se le agrega un contador de forma que cada bit es desplazado cada 16 flancos.

La Trasmisión comienza con un bit de inicio (‘0’), seguido por el vector de 8 bits y finaliza con el bit de finalización (‘1’) (Figura 4. 12). Éste trabaja de una forma semejante al módulo de recepción.

El siguiente listado, describe los módulos necesarios para realizar la trasmisión.

- Contador de bits.
- Contador de baudios.
- Máquina de estados que controla la transmisión.
- Multiplexor de selector de bit.

4.3 Algoritmo de ABC orientado a Hardware (FPGA), con la tarea de aproximador de funciones.

El propósito de esta sección, es el describir el proceso realizado para la descripción e implementación del ABC. En este caso, no se logró llegar a la comunicación por UART, pero se logró realizar la implementación del ABC para resolver problemas de aproximador de funciones supervisadas, como se describió en el subtema 3.3 Aproximador de funciones, usando ABC.

En la implementación del ABC, se hizo uso de formato de punto fijo (16 bits, el bit más significativo corresponde al signo, los siguientes 4 bits serán las parte entera y los últimos 11 bits serán la parte fraccionaria, como se explicó en previamente), ya que era necesario el tener una evolución y aleatorización de los números de una forma fraccionaria para aumentar la exactitud de la matriz inversa, y al igual de en software, se cuenta con 2 señales objetivo ($S1 = X * \mathrm{Sin}(2\pi * 1000 * \mathrm{tiempo})$ y $S2 = X * \mathrm{Sin}(2\pi * 800 * \mathrm{tiempo})$) y la mezcla de las señales (X1 = (S1+ S2*0.5) y X2 = (S2+ S1*0.5)) (Figura 4. 14).

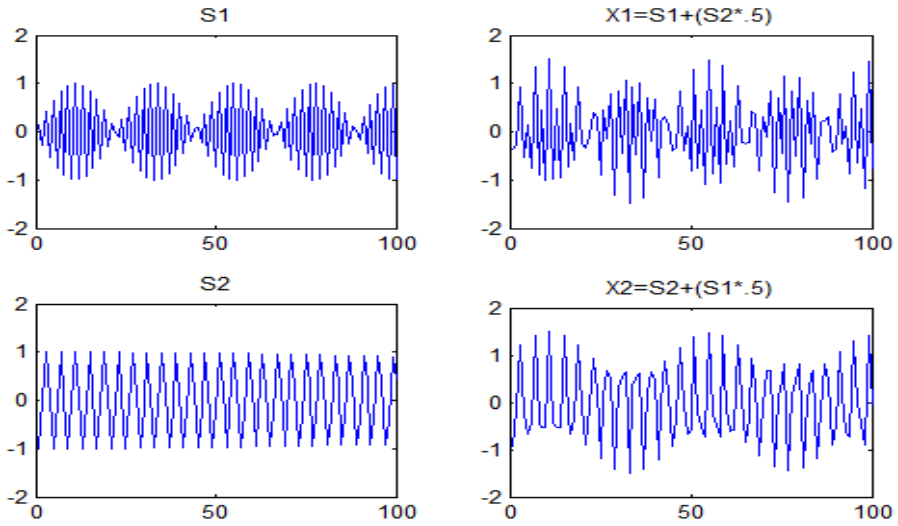


Figura 4. 14 Señales meta y mezclas, utilizadas para ABC.

Buscando poder recuperar las señales objetivo y por consiguiente obtener la matriz inversa que resuelve este coctel de señales, como función objetivo se volvió a hacer uso de la función de error cuadrático medio (Ec. 3.20). La implementación se realizó con una colonia de abejas reducida, para verificar su correcto funcionamiento, la cual está conformada por 5 abejas, 100 ciclos de error y 500 iteraciones, las cuales fueron suficientes para obtener la matriz inversa resultante; ésta nos da valores muy parecidos a los obtenidos en software.

Para la implementación exitosa del ABC en un FPGA, es necesario seguir una metodología, la cual nos permite planificar y ordenar el desarrollo de ABC, la cual se muestra en el siguiente diagrama de bloques (Figura 4. 15).

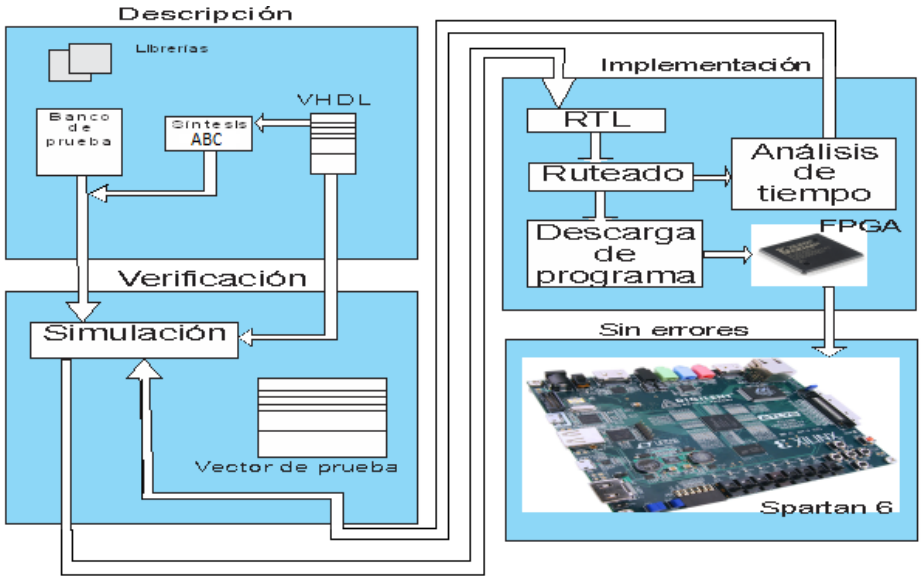


Figura 4. 15 Diagrama a bloques de la metodología para implementación de ABC.

En la síntesis del ABC:

1.- Se generaron un banco de prueba correspondiente a S1, S2, X1 y X2.

2.- Se estructura una máquina de estados, de tal modo de poder sincronizar los procesos y evitar el desbordamiento de datos.

3.- Verificación en simulación del correcto funcionamiento del ABC.

La máquina de estados realizada para el ABC, se muestra en la Figura 4. 16**Error! Reference source not found.**, donde se observa la rutina completa a seguir y, los estados necesarios para realizar correctamente cada proceso.

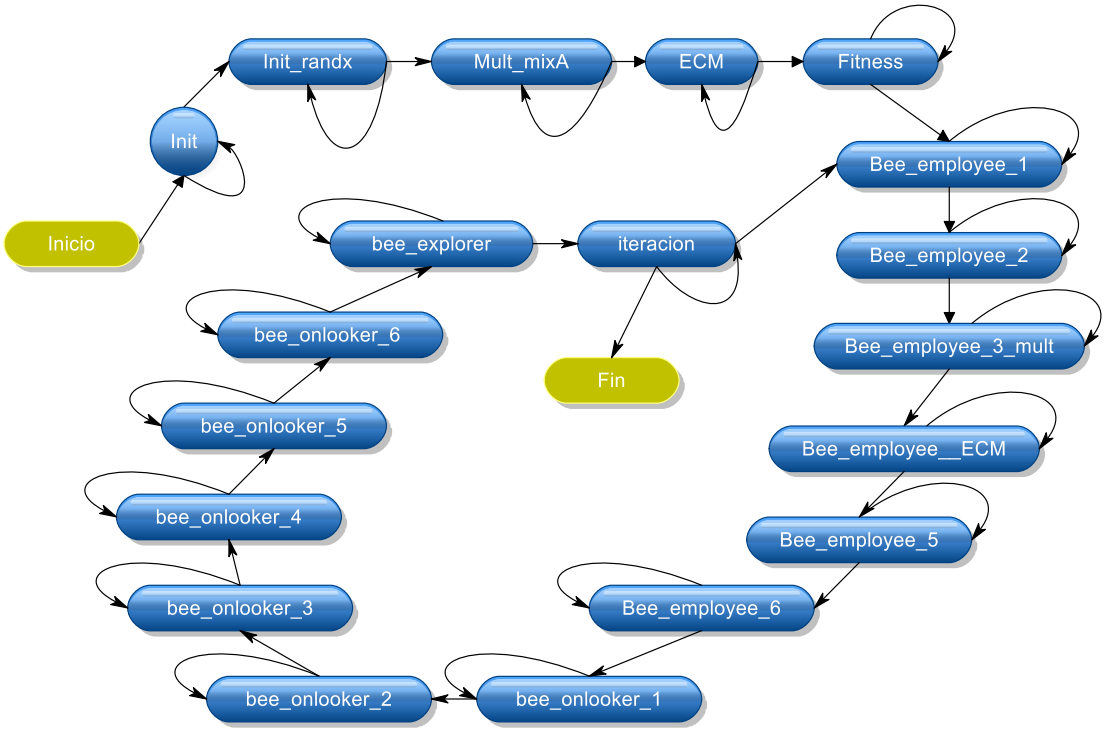


Figura 4. 16 Máquina de estados del ABC.

En la Tabla se resumen los estados de la máquina de estados del ABC.

Tabla 4.2. Resumen de la máquina de estados del ABC.

Estado VHDL	Descripción
Init	Se inicializan las variables y a su vez funciona como reset-
Init_randX	En este estado se generan las variables aleatorias correspondientes a la posición de las fuentes de alimento, usando los registros de desplazamiento con realimentación lineal.
Mult_mixA	Se multiplican las señales mezcladas por las fuentes de alimento propuestas, que corresponde a la matriz inversa de solución.
ECM	Aquí se genera la parte de la función de error cuadrático medio, la cual sería la sumatoria de la resta, $\hat{Y}_i$ corresponde al valor de n predicciones y Yi es el vector de los valores objetivo ($\sum_{i=1}^{n}(\widehat{Y}_i - Y_i)$).
Fitness	Se busca el ECM mínimo entre la colonia de abejas propuesta, donde su funcionamiento es el de guardar la fuente de alimento más abundante.
Bee_employee_1	- Se generan variables aleatorias que puedan ser usadas en la etapa posterior, para modificar la posición de búsqueda en las fuentes de alimento. - También se encargan de generar aleatoriamente los índices que corresponden a K y J.
Bee_employee_2	Modifica las fuentes de alimento, según los índices K y J, con las variables aleatorias generadas en la etapa posterior.
Bee_employee_3 _mult	Se vuelve a iniciar el proceso de evaluación de las fuentes de alimento. En esta etapa se multiplica la nueva matriz inversa generada en la etapa anterior, con la mezclas X1 y X2.
Bee_employee_ECM	Se vuelve aplicar la función de error cuadrático medio, para los nuevos resultados de las multiplicaciones de la etapa anterior.
Bee_employee_5	Compara los resultados ECM de cada fuente y si mejoran los sustituye de los valores anteriores.
Bee_employee_6	Compara la mejor fuente de alimento encontrada anteriormente con las nuevas fuentes de alimento. Si encuentra una fuente de alimento más rica en néctar, se sustituye la fuente de alimento anterior por la nueva.
bee_onlooker_1	- Se generan variables aleatorias que puedan ser usadas en la etapa posterior, para modificar la posición de búsqueda en las fuentes de alimento.

	- También se encargan de generar aleatoriamente los índices que corresponden a K, J y un criterio de selección de fuentes de alimento para una nueva exploración.
bee_onlooker_2	Modifica las fuentes de alimento, según los índices K, J y el criterio de selección de las fuentes que se volverán a explorar.
bee_onlooker_3	Se vuelve a iniciar el proceso de evaluación de las fuentes de alimento. En esta etapa se multiplica la nueva matriz inversa generada en la etapa anterior, con la mezclas X1 y X2.
bee_onlooker_4	Se vuelve a aplicar la función de error cuadrático medio, para los nuevos resultados de las multiplicaciones de la etapa anterior.
bee_onlooker_5	Compara los resultados ECM de cada fuente y si mejoran, los sustituye de los valores anteriores.
bee_onlooker_6	Compara la mejor fuente de alimento encontrada anteriormente con las nuevas fuentes de alimento. Si encuentra una fuente de alimento más rica en néctar, se sustituye la fuente de alimento anterior por la nueva.
bee_explorer	Se compara el número de viajes o ciclos de errores, donde las fuentes de alimento no mejoraron. Si se llega al criterio propuesto, se abandona la fuente de alimento y aleatoriamente se propone una nueva, esto para cada fuente de alimento.
Iteración	Este estado, solamente es un contador de iteraciones. El cual tiene la tarea de terminar el proceso según el número de iteraciones establecidas.

En la Figura 4. 17 se pueden apreciar los resultados obtenidos en simulación del ABC con la tarea de aproximador de funciones.

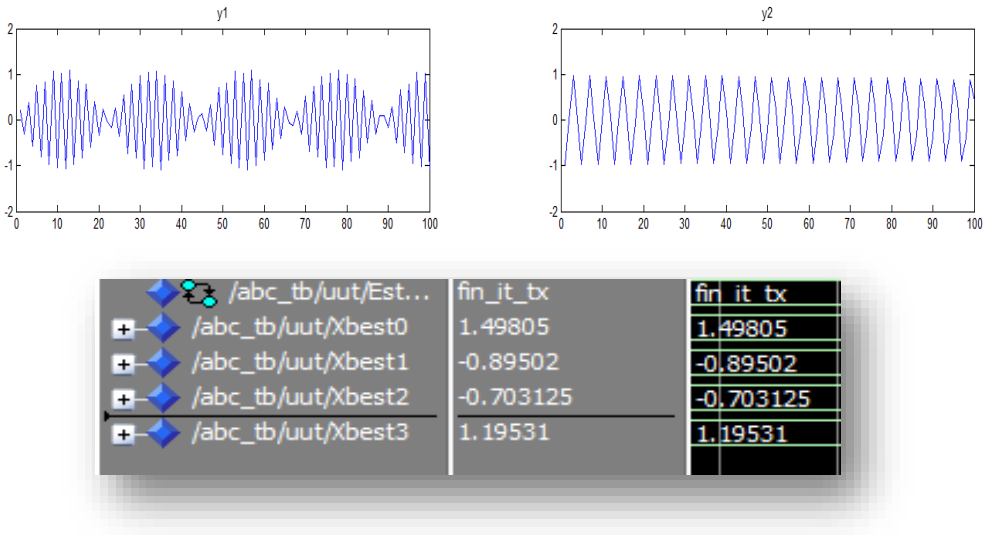


Figura 4. 17 Resultados de simulación ABC, con la tarea de aproximador de función.

4.4 Conclusiones.

La implementación de los algoritmos de ABC y PSO, busca como finalidad el disminuir los tiempos de procesamiento, esto debido al funcionamiento de paralelismo de hardware (FPGA). Teniendo como ventajas, el diseño hardware por medio de código, el que se puede editar simular, modificar y finalmente sintetizar. También cuenta con una flexibilidad física, que nos permite tener en una sola placa distintos elementos como serian UART, temporizadores, multiplexores, etc. Por lo cual se pueden crear prototipos muy rápidamente.

Los algoritmos meta-heurísticos antes mencionados, tienen como base, que son algoritmos cíclicos y que mantiene una estructura fija con respecto al uso de variables. Esto permite que puedan ser implementables. La programación mediante VHDL, nos permite modelar y simular sistemas desde un alto nivel de abstracción hasta el nivel más bajo (compuertas, etc.).

En este trabajo de tesis, se logró realizar correctamente la implementación de ambos algoritmos meta-heurísticos, obteniendo resultados aceptables y parecidos a los obtenidos en software, pero faltaría el seguir analizando los resultados obtenidos por estas implementaciones y seguir evaluando su correcto funcionamiento. Así como realizar una optimización y depuración en el código para aumentar su velocidad de procesamiento, y por consiguiente, lograr obtener resultados más óptimos.

Capítulo 5. Conclusiones y Trabajo Futuro.

En este capítulo se hace una presentación de los resultados conseguidos en este trabajo de tesis. El objetivo planteado en este trabajo de tesis, era el poder obtener las mascarillas mediante Algoritmo de Optimización Meta-Heurísticos para una CNN, con la tarea de procesamiento de imágenes y el explorar el potencial del ABC y el PSO, con diferentes tareas y realizar la implementación en FPGA de algunas de ellas.

Así como también se describieron las dificultades en la realización de este trabajo, para terminar con un apartado de las posibles extensiones futuras de este tema.

5.1 Análisis Global de Resultados.

Se considera que los objetivos planteados en este tema de tesis se han cumplido, debido a que se lograron obtener las mascarillas para la CNN, en procesamiento de imágenes para diferentes tareas como fueron las de removedor de ruido, detección de bordes, etc. Entre las dificultades encontradas en este tema están los tiempos de procesamiento de imágenes mayores a 50 x 50 pixeles en la CNN.

Entre otros propósitos, se plantea el uso del PSO para la segmentación de imágenes, lo cual fue logrado, ya que se logró tanto la implementación en software como en hardware, obteniendo resultados correctos y por consiguiente poder explorar en una gama amplia de diferentes umbrales de una imagen para poder segmentar diferentes objetos y fondos esta misma, pero faltaría el seguir haciendo pruebas con diferentes imágenes en la FPGA, así como conocer sus tiempos de procesamiento, realizar comparaciones con sistemas semejantes a éste y, conocer su margen de error con respecto a sistemas más sofisticados.

El Aproximador de Funciones usando ABC, es otro logro alcanzado con éxito debido a que se logró resolver sistemas básicos de tipo ICA de una forma supervisada tanto en software como en hardware. También los tiempos de procesamiento disminuyen con esta metodología a comparación con el uso de Redes Neuronales Artificiales. Por cuestiones de tiempo en la implementación no se logró llegar a las comunicaciones entre PC y FPGA, así como también faltaría hacer más pruebas en la implementación del algoritmo para verificar el correcto funcionamiento para diferentes señales.

En la metodología propuesta para el entrenamiento de Redes Neuronales Artificiales en software, se logró entrenar redes neuronales chicas con una gran eficiencia, pero la Red Neuronal Artificial usada en esta tesis, debido a su complejidad, no se consiguióun entrenamiento totalmente satisfactorio, así como también se encontró que requiere de mayor tiempo de entrenamiento a comparación con el entrenamiento por back-propagation [26] .

Como conclusión general, podríamos decir que los Algoritmos de Optimización Meta-Heurísticos demostraron obtener resultados de muy alta calidad y determinada exactitud,

esto ya que han sido evaluados en diferentes tareas y en su mayoría se han obtenido resultados óptimos en tiempos menores a los algoritmos clásicos de optimización u otras metodologías para las tareas presentadas en este trabajo de tesis.

5.2 Trabajo Futuro

Entre las posibles mejoras a este trabajo de tesis, se propone el realizar más pruebas a las diferentes implementaciones en FPGA, para poder afinar los resultados obtenidos. Esto resultaría especialmente importante para conocer sus tiempos de procesamiento en hardware y para determinar si presentan una ventaja para poder llevarlos a un diseño de algún circuito integrado.

Por otra parte, sería adecuado el seguir implementando diferentes tareas a los Algoritmos de Optimización Meta-heurísticos, para seguir explotando esta herramienta que facilita y acelera el procesamiento de optimización.

También se podrían utilizar los Algoritmos de Optimización Meta-heurísticos en entrenamiento de Redes Neuronales Artificiales, pero con la modalidad de ir expandiendo las capas ocultas de la red de forma automática con un criterio de optimización, el cual también estaría controlado por algún Algoritmo de Optimización Meta-heurístico.

Existen diferente modificaciones a los Algoritmos de Optimización Meta-heurísticos, los cuales permiten una convergencia más rápida y exacta, y que además también se podrían implementar en hardware para aumentar la exactitud de los resultados y disminuir los tiempos de procesamiento.

Bibliografía

[1] K. Deb, Current trends in evolutionary multi-objective optimization., International Journal for Simulation and Multidisciplinary Design Optimization, pp. 1 - 8 , 2007.

[2] J. G. Nieto, *Algoritmos Basados en Inteligencia Colectiva para la Resolución de Problemas de Bioinformática y Telecomunicaciones,* Universidad de Málaga, 2007.

[3] X.-S. Yang, Nature-inspired Metaheuristic Algorithms, Luniver Pess, 2010.

[4] E. Talbi, Metaheuristics: From Design to Implementation, New Jersey: John Wiley & Sons, 2009.

[5] K. Deb, Optimization for Engineering Design Algorithms and Examples., Prentice Hall of India, 2000.

[6] X.-S. Yang, Swarm Intelligence and bio-inspired computation theory and applications, Elsevier Inc., 2013.

[7] J. Kennedy, The Particle Swarm: Social Adaptation Knowledge, IEEE International Conference on Evaluationary Computation, 1997, pp. 303-308.

[8] J. Kennedy, R. Eberhart y Y. Shi, Swarm Intelligence., San Francisco: Morgan Kaufmann Publishers, 2011.

[9] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Kayseri/Türkiye: technical report-tr06,, octubre, 2005.

[10] D. Karaboga, Algorithm, On the performance of artificial bee colony (ABC), Applied Soft Computing , pp 687-697, Enero 2008.

[11] D. a. C. O. Karaboga, A novel clustering approach: Artificial Bee Colony (ABC) algorithm., 2011.

[12] J. E. R. Kennedy, Particle swarm optimization. In: the 1995 IEEE International Conference on Neural Networks, Piscataway, New Jersey, pp. 1942 - 1948 (1995).

[13] L. O. Chua y L. Yang, Cellular Neural Networks: Theory, vol. 35, IEEE Trans. on Circuits and Systems, 1988, pp. 1257-1272.

[14] T. Roska, J. Hámori, E. Lábos, K. Lotz, L. Orzó, J. Takács, P. Venetianer, Z. Vidnyánszky y A. Zarándy, The use of CNN Models in the Subcortical Visual Pathway, vol. 40, IEEE Trans. Circ. Syst. I Fund. Theory Appl, 1993, pp. 182-195.

[15] S. Moser y E. Pfaffhauser, «Cellular Neural Network Simulator,» Julio 1998. [En línea]. Available: http://www.isiweb.ee.ethz.ch/haenggi/CNN_web/CNNsim_adv_manual.html. [Último acceso: Agosto 2014].

[16] M. Hänggi y G. S. Moschytz, "An Exact and Direct Analytical Method for the Design of Optimally Robust CNN Templates", IEEE Transactions on Circuits and Systems, pp. 304 - 311, 1999.

[17] A. Ushida, y K. Nakai, “Design Technique on Cellular Neural Network”, vol. 78, Japan: Electronics and Comunications in Japan., pp. 97 - 107, 1995.

[18] I. Fajfar y F. Bratkovic, Design of Monotonic Binary-Valued Cellular Neural Networks, SEVILLE: CNNA’96, Fourth IEEE International Workshop on Cellular Neural Networks and their Applications, 1996.

[19] C. Steven Chapra y P. Raymond Canale, Métodos numéricos para ingenieros, Mexico, D.F.: Mc Graw-Hill, 1988.

[20] J. E. M. Solis, Circuito integrado analogico CMOS con arquitectura de red neuronal celular, Mexico, D.F., 2002.

[21] M. Hänggi, «Martin Hänggi's Homepage,» 1999. [En línea]. Available: http://www.isiweb.ee.ethz.ch/haenggi/. [Último acceso: Julio 2015].

[22] J. J. Esqueda Elizondo y L. E. Palafox Maestre, Fundamentos de procesamiento de imagenes, Universidad Autonoma de Baja California, 2005.

[23] F. Hamdaoui, A. Khalifa, A. Sakly y A. Mtibaa, Real Time Implementation of Medical Images Segmentation Based on PSO, Monastir Tunisie, 2013.

[24] R. Benitez, G. Escudero, S. Kanaan y D. Masip Rodo, Inteligencia artificial avanzada, Barcelona: UOC, 2014.

[25] M. T. Hagan, H. B. Demuth y M. Beale, Neural Network Design, International Thomson Publishing Company, 1996.

[26] A. Pérez-García, G. Tornez-Xavier, L. Flores-Nava, F. Gómez-Castañeda y J. Moreno-Cadenas, Multilayer perceptron network with integrated training algorithm in FPGA, Mexico D.F., Mexico : Department of Electrical Engineering, CINVESTAV-IPN., 2014.

[27] P. P. Chu, FPGA prototyping by VHDL examples, New Jersey: John Wiler and Sons, Inc., 2008.

[28] D. G. Maxinez y J. Alcalá Jara, VHDL. El arte de programar sistemas digitales., Mexico: Grupo Editorial Patria, 2010.

[29] E. Aguillón Martínez y M. J. López Barrientos, NAM - Facultad de Ingeniería, División de Ingeniería Eléctrica., 2012. [En línea]. Available: http://redyseguridad.fi-p.unam.mx/proyectos/criptografia/criptografia/index.php/3-gestion-de-claves/33-generadores-y-distribucion-de-claves/332-generadores-pseudoaleatorios?showall=&start=1. [Último acceso: 27 07 2015].

Anexo A.

Segmentación de imágenes, Algoritmo de PSO - Script (Matlab)

```
%%Segmentación
clc; clear; close all;

im=imread('F:\Respaldo\Matlab\Segmentacion\ABC\PSO_segmentation\digital
10.png');
ImagenGris=rgb2gray(im);
[di,dj]=size(ImagenGris); h=zeros(1,256);
t=0;
for i=1:di %% Histograma
     for j=1:dj
         k = ImagenGris(i,j);
         h(k+1) = h(k+1)+1;
     end
end

Xim=randi([31 75],1,256);
[PSO,t,Aptitud_mejorposg]=PSO1B(Xim,h);
Xim=randi([76 150],1,256);
[PSO1,t1,Aptitud_mejorposg1]=PSO1B(Xim,h);
Xim=randi([30 150],1,256);
[PSO3,t3,Aptitud_mejorposg3]=PSO1B(Xim,h);

figure
subplot(2,3,1);imshow(im);title('Imagen Original');
subplot(2,3,2);imshow(ImagenGris);title('Imagen en escala de grises');
subplot(2,3,3); plot(h);title('Histogramas');
subplot(2,3,4);imshow(PSO);title('Region I 31-75');
subplot(2,3,5);imshow(PSO1);title('Region II 76-150');
subplot(2,3,6);imshow(PSO3);title('Region III Global 30-150');

umbrales=[t,t1,t3];
 display(umbrales)
```

Segmentación de imágenes, Algoritmo de PSO – Función (Matlab)
PSO1B

```
function [PSO,mejorposg,Aptitud_mejorposg]=PSO1B(Xim,h)
%%%%%%%%%%%%%%%% Inicialización
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

P=80; %Numero de partículas swarm.
G=5; %Defino el número de partículas vecinas------- En este caso serían
4 grupos de 5 partículas c/u
Ng=P/G; % Calculo el número de grupos que voy a utilizar--ojo!!!! debe
dar un numero entero.
Dim=256; %Dimensiones del problema (# de landas).
inermax=0.9; %inercia inicial.
inerfin=0.4; %inercia final.
tmax=100; %Numero máximo de iteraciones.
c1=.5; %Control componente cognitivo.
c2=.5; %Control componente social.
w=.5;
Vmax=256; %Valor máximo de velocidad donde -Vmax<V<Vmax.
Vmin=1;
Ximax=18;Ximin=0;
E=.3; %Error global, tolerancia o criterio de salida del proceso
iterativo.
t=0; %Contador de iteraciones.
Vim=rand(1,256);
s=0;
im=imread('F:\Respaldo\Matlab\Segmentacion\ABC\PSO_segmentation\digital
10.png');
ImagenGris=rgb2gray(im);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
S=0;
for p=1:256
Ii=h(p);
s=Ii-Xim(p);

        if s==0
            Sk=1;
        else
            Sk=0;
        end
 S=S+Sk;
 p3=p+30;
Aptitud_Xi(p)=testfunction1B(p,S); % cálculo de fitness
end
s=0; S=0; p3=0;

Aptitud_mejorposi=Aptitud_Xi; %En este caso la mejor posición de la
particula es la que se acaba de calcular.
mejorposi=Xim; %La mejor posición de las partículas es la misma que se
acabó de obtener.
[Aptitud_mejorposg,I]=max(Aptitud_mejorposi); %Se obtiene el valor del
mejor valor hallado hasta el momento de la nube.
```

```
mejorposg=mejorposi(I);%Se halla la mejor posición de la nube de
partículas


% %  SE ACTUALIZA EL VALOR DE LA POSICION SUMANDOLE LA VELOCIDAD
HALLADA.
for p=1:256
Xim(p)=Xim(p)+Vim(p);
end

%%%%%%%%%%%%%%%%%%%%%%%%  PSO %%%%%%%%%%
m=0;
Aptitud_Xi=zeros(1,P);%Inicializo los valores de la suma de la función
de costo para ser evaluados.
while m==0 && t<tmax%(Aptitud_mejorposg>=1)
t=t+1;
    s=0; S=0;
      p3=0;
%  CON LA NUEVA POSICION SE CALCULA Y SE ACTUALIZA Aptitud_Xi
    for p=1:1:Dim
        Ii=h(p);
            s=Ii-Xim(p);

        if s==0
            Sk=1;
        else
            Sk=0;
        end
             S=S+Sk;
             p3=p+30;
    Aptitud_Xi(p)=testfunction1B(p,S); % cálculo de fitness

%  SE EVALUA SI ES MEJOR QUE LA MEJOR APTITUD HALLADA Y SE ACTUALIZA
%Aptitud_Xi = fitness   Aptitud_mejorposi=pbest
       if Aptitud_Xi(p)>Aptitud_mejorposi(p);
            Aptitud_mejorposi(p)=Aptitud_Xi(p);
            mejorposi(p)=Xim(p);
       end
          TH=max(Aptitud_mejorposi);

         if TH>Aptitud_mejorposg
        Aptitud_mejorposg=TH;
        mejorposg=mejorposi(p);
         else

            if Aptitud_mejorposg>=.001%.05
               m=1;
            elseif Aptitud_mejorposg==0
                %t=1;
                m=0;


            end
```

```
         Vim(p) = w*Vim(p) + c1*rand(1)*(mejorposi(p)-Xim(p)) +
c2*rand(1)*(mejorposg-Xim(p));
        Xim(p)=Xim(p)+Vim(p); % Se actualizan los valores de la
posición.
end
end
end

[d c]=size(ImagenGris);
for i=1:d
for j=1:c
 if  ImagenGris(i,j)>mejorposg
     ImagenGris2(i,j)=1;
 else
     ImagenGris2(i,j)=0;
 end
end
end

PSO=ImagenGris2;
end
```

Aproximador de función tipo ICA, Algoritmo de ABC - Script (Matlab)

```
clear all; close all; clc
global s  % señales de entrada originales
global xt % Señales multiplicadas por una A aleatoria

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%   Frecuencia modulada %%%%%%%%%%%%%%%%%%%%%%%%%%%%
n=(1:1:100);
s1=sin(3*n);
s2=sin(11*n);
s=[s1;s2];
  s1_xt=s1+(s2*.5);
  s2_xt=s2+(s1*.5);
  xt=[s1_xt;s2_xt];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

CS = 30; %Tamaño de la colonia
D = 4; %Dimensiones del problema

L = (CS*D)/2;%Limite para Scout
It = 500;%Máximo número de iteraciones

Xmin = -1;%Límite inferior de las variables
Xmax = 1;%Límite superior de las variables
%Crea los puntos iniciales con distribución uniforme
x = Xmin + (Xmax-Xmin).*rand(CS/2,D);

fx = zeros(CS/2,1);%Vector de resultados
fitx = zeros(CS/2,1); %Vector de los fitness
```

```
TrialCounter = zeros(CS/2,1); %Vector de TrialCounter para la fase de
ScoutBees
fv = fx; fitv = fitx; v = x; p = fx; %Vector de los restados del
employed bees
bestXmin = zeros(D,2); fitm = 0;
tic
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Initial= Xmin + (Xmax-Xmin).*rand(CS/2,D);
x=Initial;
x(1)=1;x(1,2)=0;x(1,3)=0;x(1,4)=1;
disp(['Iteraciones = ' num2str(It) '  Abejas = ' num2str(CS)]);
disp(['X inicial= ' num2str(x(1,1:4))]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[fx fitx]=Eval_inicial(CS,x);%% Evaluación de variables iniciales
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for n=1:It
%%  Emplyed Bees Phase

   for i=1:(CS/2)
        k = randi([1,D],1);
        %Comprueba que k sea diferente de i
        while (k == i)
            k = randi([1,D],1);
        end
            j = randi([1,D],1);
            phi = rand;
            v(i,j) = x(i,j) + phi*(x(i,j)-x(k,j));
            fv(i,:) = funObjetivo(v(i,:));
            fitv(i,1) = fitness(fv(i,1));
            %Aplica Greedy
            if (fitv(i,1) > fitx(i,1))
                x(i,:) = v(i,:);
                fx(i,1) = fv(i,1);
                fitx(i,1) = fitv(i,1);
                TrialCounter(i,1) = 0;
            else
                TrialCounter(i,1) = TrialCounter(i,1) + 1;
            end
    end
%% Calculo de probabilidad
fitT = 0;

    for i=1:(CS/2)
        fitT = fitT + fitx(i,1);
    end
    for i=1:(CS/2)
        p(i,1) = fitx(i,1)/fitT;
    end

%%  Onlookers Bees Phase
t = 0;  i = 1;
    while (t < (CS/2))
        if (rand < p(i,1))
            t = t + 1;
```

```
            k = randi([1,D],1);
            %Comprueba que k sea diferente de i
            while (k == i)
                k = randi([1,D],1);
            end
            %Indice j
            j = randi([1,D],1);
            %Valor de phi en [0,1]
            phi = rand;
            v(i,j) = x(i,j) + phi*(x(i,j)-x(k,j));
            fv(i,:) = funObjetivo(v(i,:));
            fitv(i,1) = fitness(fv(i,1));
            %Aplica Greedy
            if (fitv(i,1) > fitx(i,1))
                x(i,:) = v(i,:);
                fx(i,1) = fv(i,1);
                fitx(i,1) = fitv(i,1);
                TrialCounter(i,1) = 0;
            else
                TrialCounter(i,1) = TrialCounter(i,1) + 1;
            end
        end
        i = i + 1;
        if (i == (CS/2 + 1))
            i = 1;
        end
    end

    %Memoriza el mejor


%% -- Scout Bees Phase
 for i=1:(CS/2)
        if(TrialCounter(i,1) > L)
            x(i,:) = Xmin + (Xmax-Xmin).*rand(1,D);
        end
    end
%% Se memoriza las mejores solucionesç
    mejIn = bestXmin;
    fitmIn = fitm;
    [bestXmin,fitm] = best(x,fitx,mejIn,fitmIn);
   % fbest(It)=fitx;

end
toc
 inv_1_a=[bestXmin(1),bestXmin(2);bestXmin(3),bestXmin(4)];
 y1=inv_1_a*xt;
 disp(['A= ' num2str(bestXmin)]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%   Frecuencia modulada %%%%%%%%%%%%%%%%%%%%%%%%%%
 subplot(3,2,1);plot(s(1,:));title('S1');axis([0 100 -2 2]);
 subplot(3,2,2);plot(s(2,:));title('S2');axis([0 100 -2 2]);
 subplot(3,2,3); plot(xt(1,:));title('x1');
 subplot(3,2,4); plot(xt(2,:));title('x2');

 subplot(3,2,5); plot(y1(1,:));title('y1');axis([0 100 -2 2]);
```

```
 subplot(3,2,6); plot(y1(2,:));title('y2');axis([0 100 -2 2]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure;
fitx_1=sort((fx(:)),'descend');
 plot(fitx_1,'LineWidth',2);
 xlabel('Iteration');
 ylabel('Y');
```

Aproximador de función tipo ICA, Algoritmo de ABC – Función (Matlab).
best

```
%Encuentra el mejor valor buscado y lo guarda
function [mej,fitm] = best(x, fitx,mejIn,fitmIn)
[tfitx,ind] = max(fitx);
if(tfitx > fitmIn)
    fitm = tfitx;
    mej = x(ind,:);
else
    mej = mejIn;
    fitm = fitmIn;
end
```

Aproximador de función tipo ICA, Algoritmo de ABC – Función (Matlab).
Eval_inicial

```
function [fx fitx]=Eval_inicial(CS,x)

for i=1:(CS/2)
        %Evalua la función objetivo
        fx(i,1) = funObjetivo(x(i,:));


        %funcion de fitness
        fitx(i,1) = fitness(fx(i,1));
end
```

Aproximador de función tipo ICA, Algoritmo de ABC – Función (Matlab).
fitness

```
function fit = fitness(z)
if (z < 0)
    fit = 1+abs(z);
else
    fit = 1/(1+z);
end
```

Aproximador de función tipo ICA, Algoritmo de ABC – Función (Matlab).

funObjetivo

```
function z = funObjetivo(x)

global s % señales de entrada originales
global xt % Señales multiplicadas por una A aleatoria

inv_1_a=[x(1),x(2);x(3),x(4)];
y=inv_1_a*xt;

sum1=0;
ECM_1=0;
for i=1:1:2
    for j=1:100
        sum1=sum1+((y(i,j)-s(i,j))^2);%s1(i));
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%   Frecuencia modulada %%%%%%%%%%%%%%%%%%%%%%%%%%
z=(1/100)*(sum1); %ECM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Entrenamiento de Red Neuronal Artificial, Algoritmo de ABC – Script (Matlab).

```
%/* ABC algorithm

clear all
close all
clc

global p1
global appro
%   %% Input X y Y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%  Red  Neuronal 2-5-2 %%%%%%%%%%%%%%%%%%%%%%%%%%

 load datos.dat;

x1 = datos(:,1);
P1 = x1';
y = datos(:,2);
P2 = y';
p1=[P1;P2];
appro1=datos(:,3);
appro=appro1';
%/* Control de parámetros del algoritmo ABC */
NP=50; %/* Numero de colonia (employed bees + onlooker bees)*/
FoodNumber=NP/2; %/* número de fuentes de alimento */
maxCycle=2500; %/* Numero de Iteraciones */

objfun='nntrainxor221b'; % Funcion Objetivo
D=30; %/* Numero de parámetros del problema de optimización */
hidden=5;
```

```
ub=ones(1,D)*3; %/* limite menor del parametro */
lb=ones(1,D)*(-3);%/* limite mayor del parametro */
limit=FoodNumber*D; % límite de intentos para abandonar las fuentes de
laimentos */
runtime=1;%/* Numero de ciclos completos del algoritmo */



GlobalMins=zeros(1,runtime);
MR=1;
for r=1:runtime
  tic

Range = repmat((ub-lb),[FoodNumber 1]);
Lower = repmat(lb, [FoodNumber 1]);
Foods = rand(FoodNumber,D) .* Range + Lower;
ObjVal=feval(objfun,Foods,hidden);
Fitness=calculateFitness(ObjVal);

% inicializar el conteo de trial
trial=zeros(1,FoodNumber);

%/* la mejor fuente de alimento memoria */
BestInd=find(ObjVal==min(ObjVal));
BestInd=BestInd(end);
GlobalMin=ObjVal(BestInd);
GlobalParams=Foods(BestInd,:);
iter=1;
while ((iter <= maxCycle)),

%%%%%%%%% EMPLOYED BEE PHASE %%%%%%%%%%%%%%%%%%%%%%%%%
    for i=1:(FoodNumber)

        Param2Change=fix(rand*D)+1;
        neighbour=fix(rand*(FoodNumber))+1;

        %/* Comprueba que K es diferente de i */
            while(neighbour==i)
                neighbour=fix(rand*(FoodNumber))+1;
            end;

       sol=Foods(i,:);
       %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_{kj}-x_{ij}) */
      Rj= rand;
      if Rj<MR
       sol(Param2Change)=Foods(i,Param2Change)+(Foods(i,Param2Change)-
Foods(neighbour,Param2Change))*(rand-0.5)*2;
      else
          sol(Param2Change)=Foods(i,Param2Change);
      end
        ind=find(sol<lb);
        sol(ind)=lb(ind);
        ind=find(sol>ub);
        sol(ind)=ub(ind);
        % Evaluacion de parametros
```

```
         ObjValSol=feval(objfun,sol,hidden);
         FitnessSol=calculateFitness(ObjValSol);

        if (FitnessSol>Fitness(i))
             Foods(i,:)=sol;
             Fitness(i)=FitnessSol;
             ObjVal(i)=ObjValSol;
             trial(i)=0;
             prob1(i)=1;
         else
             trial(i)=trial(i)+1;
             prob1(i)=0;
        end;



    end;

%%%%%%%%%%%%%%%%%%%%%%%%  Calculo de probabilidad %%%%%%%%%%%%%%%%

  prob=(0.9.*Fitness./max(Fitness))+0.1;

%%%%%%%%%%%%%%%%%%%%%%%% ONLOOKER BEE PHASE %%%%%%%%%%%%%%%%%%%%%%%%

i=1;
t=0;
while(t<FoodNumber)


     if(rand<prob(i))
      t=t+1;
        Param2Change=fix(rand*D)+1;
        neighbour=fix(rand*(FoodNumber))+1;
             while(neighbour==i)
                 neighbour=fix(rand*(FoodNumber))+1;
             end;

       sol=Foods(i,:);
       %  /*v_{ij}=x_{ij}+\phi_{ij}*(x_{kj}-x_{ij}) */
       sol(Param2Change)=Foods(i,Param2Change)+(Foods(i,Param2Change)-
Foods(neighbour,Param2Change))*(rand-0.5)*2;
        ind=find(sol<lb);
        sol(ind)=lb(ind);
        ind=find(sol>ub);
        sol(ind)=ub(ind);

        % Evaluacion de las nuevas soluciones
ObjValSol=feval(objfun,sol,hidden);
        FitnessSol=calculateFitness(ObjValSol);

       if (FitnessSol>Fitness(i))
             Foods(i,:)=sol;
             Fitness(i)=FitnessSol;
             ObjVal(i)=ObjValSol;
```

```
            trial(i)=0;
        else
            trial(i)=trial(i)+1;
       end;
    end;

    i=i+1;
    if (i==(FoodNumber)+1)
        i=1;
    end;
end;


%/* Memorización de la mejor fuente de alimento */
         ind=find(ObjVal==min(ObjVal));
         ind=ind(end);
         if (ObjVal(ind)<GlobalMin)
         GlobalMin=ObjVal(ind);
         GlobalParams=Foods(ind,:);
         end;

%%%%%%%%%%% SCOUT BEE PHASE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ind=find(trial==max(trial));
ind=ind(end);
if (trial(ind)>limit)
    Bas(ind)=0;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     sol=(ub-lb).*rand(1,D)+lb;
    ObjValSol=feval(objfun,sol,hidden);
    FitnessSol=calculateFitness(ObjValSol);
    Foods(ind,:)=sol;
    Fitness(ind)=FitnessSol;
    ObjVal(ind)=ObjValSol;

end;

disp([' Ýter= ' num2str(iter) ' ObjVal= ' num2str(GlobalMin) ' r = '
num2str(r)]);
iter=iter+1;
end %
GlobalMin1(r)=GlobalMin;
GlobalParams1(r,:)=GlobalParams;
end;

for r=1:runtime
a31= Red_neuronal_ABC(GlobalParams1(r,:),hidden);
%%%%%%%%%%%%%%%%%%% grafica %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 figure(r)
 x_input = linspace (-1,1,32);
 y_input = linspace (-1,1,32);

 [X1,Y2]=meshgrid(x_input,y_input);
Z = 4*X1.*(exp(-4*(X1.^2+Y2.^2)));
surf(X1,Y2,Z);title('RNA 5-2-1');
```

```
shading interp
colormap(hot)
alpha(0.5);
hold on
plot3 (P1,P2,a31,'k.')
hold off
min=toc/60;
disp([' Minutos= ' num2str(min)]);
GlobalParams;
end
```

Entrenamiento de Red Neuronal Artificial, Algoritmo de ABC – Función (Matlab). ObjVal

```
%   /*
%   Entrenamiento de Red Neuronal Artificial.

function ObjVal = nntrainxor221b(Chrom,hidden)
global p1
global appro
[Nind Nvar]=size(Chrom);
trin=p1trout =approinp=size(trin,1);
out=size(trout,1);
[R,Q] = size(p1);
 P2 = p1;
  [R,Q2] = size(P2);
for i=1:Nind
x=Chrom(i,:);

    W1 = reshape(x(1:10),5,2);
    b1 = reshape(x(11:15),5,1);
    W2 = reshape(x(16:25),2,5);
    b2 = reshape(x(26:27),2,1);
    W3 =reshape(x(28:29),1,2);
    b3 =reshape(x(30),1,1);

    y =sigmoid(W3*(sigmoid(
W2*(sigmoid(W1*P2+b1*ones(1,Q2)))+b2*ones(1,Q2)))+b3*ones(1,Q2));
sum1=0;
    for j=1:1024
        sum1=sum1+((y(1,j)-trout(1,j))^2);
    end
error=(1/1024)*(sum1); %ECM
ObjVal(i)=sqrt(error);
end;
```

Entrenamiento de Red Neuronal Artificial, Algoritmo de ABC – Función (Matlab). calculateFitness

```
function fFitness=calculateFitness(fObjV)
fFitness=zeros(size(fObjV));
ind=find(fObjV>=0);
fFitness(ind)=1./(fObjV(ind)+1);
ind=find(fObjV<0);
fFitness(ind)=1+abs(fObjV(ind));
```

Entrenamiento de Red Neuronal Artificial, Algoritmo de ABC – Función (Matlab). Red_neuronal_ABC

```
function y = Red_neuronal_ABC(Chrom,hidden)
global p1
global appro
[Nind Nvar]=size(Chrom);
trin=p1;
trout =appro;
inp=size(trin,1);
out=size(trout,1);

[R,Q] = size(p1);
 P2 = p1;
  [R,Q2] = size(P2);
for i=1:Nind

x=Chrom(i,:);

    W1 = reshape(x(1:10),5,2);
    b1 = reshape(x(11:15),5,1);
    W2 = reshape(x(16:25),2,5);
    b2 = reshape(x(26:27),2,1);

    W3 =reshape(x(28:29),1,2);
    b3 =reshape(x(30),1,1);


    y =sigmoid(W3*(sigmoid(
W2*(sigmoid(W1*P2+b1*ones(1,Q2)))+b2*ones(1,Q2)))+b3*ones(1,Q2));


end;
```

Entrenamiento de Red Neuronal Artificial, Algoritmo de ABC – Función (Matlab). sigmoid

```
function fy= sigmoid (a)
y_1=a;
[i,j]=size(a);

f=zeros(i,j);
```

```
for ii=1:i

  for jj=1:j
    f(ii,jj)=(exp(y_1(ii,jj))-exp(-y_1(ii,jj)))/(exp(y_1(ii,jj))+exp(-y_1(ii,jj)));
  end
end

fy=f;
```

Entrenamiento de Red Neuronal Artificial, Algoritmo de ABC – Función (Matlab). funObjetivo

```
function Y1 = funObjetivo(x)

global appro % señales de entrada originales


Z1= Red_neuronal_ABC_24_04(x);

sum1=0;
ECM_1=0;

    for i=1:1024
        sum1=sum1+((Z1(1,i)-appro(i))^2);
    end

Y1=(1/1024)*(sum1); %ECM_1
```

Anexo B.

LFSR de 4-bits - Código VHDL (ISE - Xilinx)

```
library ieee;
use ieee.std_logic_1164.all;

entity lfsr_1 is
 generic(initial_value: std_logic_vector(3 downto 0):= "1000";
   lfsr_width   : integer := 4);
   port(     clk      : in std_logic;
            rst      : in std_logic;
            LFSR: out std_logic_vector(lfsr_width-1 downto 0)
            );
end lfsr_1;

architecture Random of lfsr_1 is
signal q_lfsr: std_logic_vector(lfsr_width-1 downto 0);
signal serial_in  : std_logic;

begin
serial_in <= q_lfsr(1) xor q_lfsr(0);
lfsr_cnt_proc: process(rst, clk)

begin  if(rst= '1' ) then
q_lfsr <= initial_value;
elsif (rising_edge(clk)) then

q_lfsr <= serial_in & q_lfsr(q_lfsr'high downto q_lfsr'low+1);

end if;
end process lfsr_cnt_proc;
LFSR <= q_lfsr;
end architecture Random;
```

Paquete de Componentes para el Algoritmo PSO – PSO_1.vhd (ISE - Xilinx)

```
----------------------------------------------------------------------------------
-- Module Name:    PSO_1 - Behavioral
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.PSO_PACK.all;
entity PSO_1 is
        port(   clk: in std_logic;
                        rst,rst_a: in std_logic;
                        rx: in std_logic;
                        tx : out  STD_LOGIC
                        );
end PSO_1;

architecture Behavioral of PSO_1 is

signal ce_rx1, on_tx: std_logic;
signal inRS232: std_logic_vector(7 downto 0) := (others => '0');
SIGNAL W_full : std_logic_vector(23 downto 0);
signal L1 : integer range 1 to 255;
signal Umbral_0,Umbral_1,Umbral_2,Umbral_3,Umbral_4,Umbral_5 :
STD_LOGIC_VECTOR(7 DOWNTO 0);
begin
RECEPTOR: receptorRS232 port map (clk => clk, rst => rst, rx => rx, ce_rx => ce_rx1,
inRS232 => inRS232);
Histograma_banco:   Histograma port map ( clk=>clk,rst => rst, ce_rx => ce_rx1,inRS232
=> inRS232 ,W_full => W_full, L1=> L1);
PSO:    Pso_main    port    map    (clk    =>clk,rst    =>    rst,W_full=>W_full,
L1=>L1,on_tx=>on_tx,Umbral_0=>Umbral_0,Umbral_1=>Umbral_1,
        Umbral_2=>Umbral_2,Umbral_3=>Umbral_3,Umbral_4=>Umbral_4,Umbral_5=>
Umbral_5);
Transmision:    Trans    port    map
(clk=>clk,rst_a=>rst_a,on_tx=>on_tx,Umbral_0=>Umbral_0,Umbral_1=>Umbral_1,Umb
ral_2=>Umbral_2,Umbral_3=>Umbral_3,Umbral_4=>Umbral_4,Umbral_5=>Umbral_5,t
x=>tx);
end Behavioral;
```

Módulo de recepción para el Algoritmo PSO – receptorRS232.vhd (ISE - Xilinx)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity receptorRS232 is
	port(
	clk: in std_logic;
	rst: in std_logic; -- Reset general
	rx: in std_logic; -- Entrada de la recepción RS232
	ce_rx: out std_logic := '0'; -- Habilitador del receptor
	inRS232: out std_logic_vector(7 downto 0)  -- Salida que contiene el dato recibido
			);
end receptorRS232;

architecture arq_receptor of receptorRS232 is

signal erx: std_logic := '0'; -- Señal temporal de recepción
signal regrx: std_logic_vector(7 downto 0); -- Registro temporal que almacena el dato recibido
signal cntrx: std_logic_vector(8 downto 0);  -- Contador auxiliar de tiempo
signal trx: std_logic_vector(4 downto 0);-- Contador temporal para los bits del dato recibido
signal temDato: std_logic_vector(7 downto 0); -- Dato temporal

constant baudrx: std_logic_vector(8 downto 0) := "100100001";  -- Constante para la velocidad de 115200 bauds

begin
-- Proceso de detección del bit de start
process(rst, rx, trx)
begin
	if(rst = '1')then
	-- Reset general, inicializa
		erx <= '0';
		ce_rx <= '0';
	else
		if(rx = '0')then		-- Se detectó el bit de start
			erx <= '1';		-- Habilita la recepción
			ce_rx <= '1';
```

```
                else
                        if(trx = "00000")then  -- Se mantiene deshabilitada la recepción
                                erx <= '0';       -- Ya que se recibió un '1' y no es el de Start
                                ce_rx <= '0';
                        elsif(trx >= "11011")then -- Deshabilita recepción ya que se han
leido
                                erx <= '0';       -- todos los bits del registro
                                ce_rx <= '0';
                        else                      -- Sigue habilitada la recepción
                                erx <= '1';
                                ce_rx <= '1';
                        end if;
                end if;
        end if;
end process;

-- Proceso de contador de bits recibidos
process(clk, rst, cntrx, trx, erx)
begin
        if(rst = '1')then          -- Reset general
                cntrx <= (others => '0');
                trx <= (others => '0');

        else
                if(erx = '0')then -- Deshabilitada la recepción
                        cntrx <= (others => '0');
                        trx <= (others => '0');
                else              -- Habilitada la recepción
                        if(clk'event and clk = '1')then
                                cntrx <= cntrx + 1;      -- Contador de tiempo transcurrido
                                if(cntrx = baudrx)then
                                        trx <= trx + 1;
                                        cntrx <= (others => '0');
                                        --i<=i+1;
                                end if;
                        end if;
                end if;
        end if;
end process; -- Procesado de almacenamiento de bits recibidos
process (clk,rst,trx,rx,regrx)
begin
  if rst='1' then                                                                -
- Reset general
                regrx <= (others=>'0');
```

```
        elsif (clk'event and clk='1') then
                case (trx) is     -- Va almacenando el bit recibido
                        when "00100" =>  regrx(0) <=          rx;
                        when "00111" =>  regrx(1) <=          rx;
                        when "01010" =>  regrx(2) <=          rx;
                        when "01101" =>  regrx(3) <=          rx;
                        when "10000" =>  regrx(4) <=          rx;
                        when "10011" =>  regrx(5) <=          rx;
                        when "10110" =>  regrx(6) <=          rx;
                        when "11001" =>  regrx(7) <=          rx;
                        when others => null;
                end case;
         end if;
end process;

-- Proceso de salida del dato recibido
process(erx, rst, regrx, cntrx, temDato,clk)
begin
        if(rst = '1')then          -- Reset general
                inRS232 <= "00000000";
                temDato <= "00000000";
        --      i<=0;
        else
                if(clk'event and clk = '1')then
                        if(trx = "11010")then    -- Actualiza el dato después de recibir el
                                inRS232 <= regrx;       -- el 8vo bit
                                temDato <= regrx;
                        else
                                inRS232 <= temDato;
                                temDato <= temDato;
                        end if;
                end if;
        end if;
end process;
end arq_receptor;
```

Módulo de banco de histograma del Algoritmo PSO – Histograma.vhd (ISE - Xilinx)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

```
entity Histograma is
   Port ( clk : in  STD_LOGIC;
        rst : in  STD_LOGIC;
        ce_rx : in  STD_LOGIC;
        inRS232: in std_logic_vector(7 downto 0);
        W_full : out  STD_LOGIC_VECTOR (23 downto 0);
        L1: out integer range 0 to 255
                              );
end Histograma;

architecture Behavioral of Histograma is
signal contReg: std_logic_vector(7 downto 0):=(others => '0');
signal W_full_1,W_full_2:std_logic_vector(7 downto 0);

signal L: integer range 0 to 255:=0 ;
signal t1: integer range 1 to 3:=1;
signal tem_ce_rx: std_logic;
begin

-- Pulsos de sincronia
Pulso_sic:process(clk, rst, ce_rx)
begin
          if(rst = '1')then
                    tem_ce_rx <= '0';
          else
                    if(clk'event and clk = '1')then
                              tem_ce_rx <= ce_rx;
                    end if;
          end if;
end process;
-- Contador de Registros
Control_reg:process(rst, tem_ce_rx)
begin
          if(rst = '1')then
                    contReg <= (others => '0');
          else
                    if(tem_ce_rx'event and tem_ce_rx = '0')then
                              contReg <= contReg + 1;
                    end if;
          end if;
end process;

-- Guarda Datos
process(rst,contReg, tem_ce_rx, inRS232,t1)
```

```
begin
        if (rst = '1')then
                W_full_1<=X"00";
                W_full_2<=X"00";
                W_full<=(X"000000");
                t1<=1;
                L<=0;
        elsif(tem_ce_rx'event and tem_ce_rx = '0')then

                                if t1=1 and contReg<=X"FF"   then
                                W_full_1<=  inRS232 ;
                                t1<=2;
                                elsif t1=2 and contReg<=X"FF"     then
                                W_full_2<=  inRS232  ;
                                t1<=3;
                                elsif t1=3 and contReg<=X"FF"     then
                                W_full<=inRS232 & W_full_2 & W_full_1;
                                L<=L+1;
                                L1<=L;
                                t1<=1;
                                end if;
        end if;

end process;
end Behavioral;
```

Módulo de el Algoritmo PSO – Pso_main.vhd (ISE - Xilinx)

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
USE work.PSO_PACK.ALL;

entity Pso_main is
generic (N:positive range 1 to 5 :=5);
   Port ( clk : in  STD_LOGIC;
        rst: in  STD_LOGIC;
       W_full : in  STD_LOGIC_VECTOR (23 downto 0);
        L1: in integer range 0 to 255;
        Umbral:out std_logic_vector(7 downto 0):=X"01";
```

```
        Umbral_0,Umbral_1,Umbral_2,Umbral_3,Umbral_4,Umbral_5:inout
STD_LOGIC_VECTOR(7 DOWNTO 0);
        on_tx : out  STD_LOGIC
                              );
end Pso_main;

architecture Behavioral of Pso_main is

constant w: std_logic_vector(7 downto 0) :=X"01";
constant c1: std_logic_vector(7 downto 0):=X"01";
constant c2: std_logic_vector(7 downto 0):=X"01";
constant P: integer:= 5;-- Numero de particulas

---------- Señales de trabajo  ----------

          signal feedback_0,feedback_1,feedback_2,feedback_3,feedback_4,feedback_5 :
std_logic:='0';


          type X2 is array (5 downto 1) of std_logic_vector(7 downto 0);
          signal X1: X2;

          type aptitud is array (P downto 1) of std_logic_vector(19 downto 0);
          signal Aptitud_Xi:aptitud;

          type count1a1 is array (P downto 1) of std_logic_vector(7 downto 0);
          signal mejorposiX1 : count1a1;

          type fun2 is array (P downto 1) of std_logic_vector(19 downto 0);
          signal Aptitud_mejorposi:fun2;

          signal Aptitud_mejorposg :std_logic_vector(19 downto 0);

          signal X1best : std_logic_vector(7 downto 0):="00000000";

          type X3_1 is array (5 downto 1) of std_logic_vector(7 downto 0);
          signal vel1 :X3_1;

          signal i: integer range 0 to 20:=0 ;

          signal rand1_1,rand2_1: std_logic_vector(7 downto 0);

          signal Control: std_logic_vector(1 downto 0);
          signal Cont: std_logic;
```

```
        signal ciclo,ciclo_2: std_logic:='0';
        signal it: integer range 0 to 20 :=0;
        type registro_total is array (255 downto 0) of std_logic_vector(19 downto 0);
        signal W_full_1 : registro_total:= (others => X"00000");
-----------------------------------------------------
Type mis_estados is
(fin_transmision,Rand_X,sumatoria,Funcion,Apt_mej,Max_pso,Vel_inicial,Xi_lim,Xi_Ma
x,Global,Vel_completa, iteracion,rango,fin);
signal Estado_Actual,Estado_Siguiente: mis_estados;
signal Ii : integer range 1 to 255 :=1;
signal Ij : integer range 1 to 6 :=1;

Signal Up: std_logic:='0';
signal U: integer range 0 to 7:=0;
signal U_1: integer range 0 to 3:=0;
signal Xmin: std_logic_vector (7 downto 0):=X"1D";
signal Xmax: std_logic_vector (7 downto 0):=X"3A";

begin
---------------------- PILA HISTROGRAMA----------------
process (clk, rst, L1,W_full)
begin
        if rst='1' then
                W_full_1<= (others =>X"00000");

  elsif (clk'event and clk = '1') then

      W_full_1(L1)<=W_full(19 downto 0);

   end if;

end process;
process (L1,rst)
begin
         if rst= '1' then
                Up<='0';
         elsif L1= 255 then
                Up<='1';
        else
                Up<='0';
         end if;
end process;
------------- Registro de estado  ----------------------
Secuanciador:process (clk,rst)
```

```
begin

        if rst = '1' then
    Estado_Actual<=fin_transmision;
  elsif (clk'event and clk = '1') then
                Estado_Actual<=Estado_Siguiente;
   end if;

end process Secuanciador;

------------ Logica del estado Siguiente ----------
process (Estado_Actual,Estado_Siguiente,Up,Cont,ciclo,ciclo_2)
begin

case (Estado_Actual) is
    when fin_transmision =>
                        if Up='1' then
                                Estado_Siguiente<=Rand_X;
                        else
                                Estado_Siguiente<=fin_transmision;
                        end if;
    when Rand_X =>
                        if Cont='1'  then
                                Estado_Siguiente<=   sumatoria;
                        else
                                Estado_Siguiente<=Rand_X;
                        end if;
    when sumatoria=>

                        if Cont='0' and ciclo='0' then
                        Estado_Siguiente<=sumatoria;

                        else
                        Estado_Siguiente<=Funcion;
          end if;
                when Funcion =>
                        if   ciclo='0' and Cont='0' then
                                Estado_Siguiente<=sumatoria;
                        elsif ciclo='0' and Cont='1'  then
                                Estado_Siguiente<=Funcion;
                        elsif ciclo_2='1' and ciclo='1' and Cont='0'  then
                                Estado_Siguiente<=Xi_Max;
                        else
                                Estado_Siguiente<=Apt_mej;
```

```
                end if;
            when Apt_mej =>
                        if Cont='0' then
                        Estado_Siguiente<=Max_pso;
                        else
                        Estado_Siguiente<=Apt_mej;
                        end if;
              when Max_pso=>
                        if Cont='1' then
                                Estado_Siguiente<=Vel_inicial;
                        else
                                Estado_Siguiente<= Max_pso;
                        end if;
              when Vel_inicial=>
                        if Cont='1' then
                                Estado_Siguiente<=Xi_lim;
                        else
                                Estado_Siguiente<=Vel_inicial;
                        end if;
              when Xi_lim=>
                        if Cont='0' and ciclo='0' and ciclo_2='1' then
                                Estado_Siguiente<=sumatoria;
                        else
                                Estado_Siguiente<=Xi_lim;
                        end if;
              when Xi_Max=>
                        if Cont='1' then
                                Estado_Siguiente<=Global;
                        else
                                Estado_Siguiente<=Xi_Max;
                        end if;
              when Global=>
                  if Cont='1' then
                                Estado_Siguiente<=Vel_completa;
                  else
                                Estado_Siguiente<=Global;
                  end if;
              when Vel_completa=>
                  if Cont='1' then
                                Estado_Siguiente<=iteracion;
                  else
                                Estado_Siguiente<=Vel_completa;
                  end if;
              when iteracion =>
```

```
                        if Cont='1' and ciclo='1' then
                                Estado_Siguiente<=Xi_lim;
                        elsif Cont='0' and ciclo='0' then
                                Estado_Siguiente<=rango;
                        else
                        Estado_Siguiente<=iteracion;
                        end if;
                when rango =>
                        if Cont='1' and ciclo='1' and ciclo_2 ='1' then
                                Estado_Siguiente <=fin ;
                        elsif Cont='0' and ciclo='0' and ciclo_2 ='1' then
                                Estado_Siguiente <= rango;
                        else
                                Estado_Siguiente <=fin_transmision ;
                        end if;
                when fin=>
                        Estado_Siguiente <= fin;
  end case;
end process;

process (Estado_Actual,Control,clk)
variable sum1: std_logic_vector(19 downto 0);
variable suma: std_logic_vector(19 downto 0);
variable Vel1a_1,Vel1a_2,Vel1a_3,Vel1a_4,Vel1a_5: std_logic_vector (27 downto 0);

begin
if(rising_edge(clk))then
  case Estado_Actual is
    when fin_transmision =>
     Umbral <=X"02";
                 X1best<=X"00";
                 Control<="00";
                        X1<= (others=>Xmin);
                        Aptitud_mejorposi<=(others=>X"00000");
                        mejorposiX1<=(others=>X"00");
                        Aptitud_mejorposg<=X"00000";
                        it<=0;
                        Ii<=1;  Ij<=1;
                        ciclo<='0';
                        Cont<='0';
                        ciclo_2<='0';
                        on_tx<='0';
```

```
    when Rand_X =>

                                X1(1)<=Xmin+X"0A";
                                        X1(2)<=Xmin+X"04";
                                        X1(3)<=Xmin+X"07";
                                        X1(4)<=Xmin+X"0B";
                                        X1(5)<=Xmin+X"0C";
                        Control<=Control+1;
                                 if control = "11" then
                                        Cont<='1';
                                        Control<="00";
                                        sum1:=X"00000";
                                        suma:="00000000000000000000";
                                else
                                        Cont<='0';
                                end if;
---------------------  Proceso de evalucion de funcion objetivo-----------------
        when sumatoria=>
                                sum1:=W_full_1(Ii)-X1(Ij);
                                                if sum1= 0 then
                                                  sum1:=X"00001";
                                                else
                                                        sum1:=X"00000";
                                                end if;
                                                        suma:=suma+sum1;
                         if  Ii= 255 then
                                        cont<='1';
                                        ciclo<='0';
                         else
                                Ii <= Ii+1;
                                cont<='0';
                                ciclo<='0';
                         end if;
        when Funcion =>
                        Aptitud_Xi(Ij)<=divide (suma,Xmax);-- SE DIVIDE ENTRE 75
USANDO UN RANJO DE 30 A 75
Control<=Control+1;
                         if control = "11" and Ij<5 then
                                Ij<=Ij+1;
                                Ii<=1;
                                cont<='0';
                                ciclo<='0';
                                Control<="00";
                                sum1:=X"00000";
```

```
                        suma:="00000000000000000000";
                elsif control = "11" and Ij=5 then

                        cont<='0';
                        ciclo<='1';

                end if;
        when Apt_mej =>
                                Aptitud_mejorposi(1)<=Aptitud_Xi(1);
                                mejorposiX1(1)<=X1(1);

                                Aptitud_mejorposi(2)<=Aptitud_Xi(2);
                                mejorposiX1(2)<=X1(2);

                                Aptitud_mejorposi(3)<=Aptitud_Xi(3);
                                mejorposiX1(3)<=X1(3);

                                Aptitud_mejorposi(4)<=Aptitud_Xi(4);
                                mejorposiX1(4)<=X1(4);

                                Aptitud_mejorposi(5)<=Aptitud_Xi(5);
                                mejorposiX1(5)<=X1(5);

                                if mejorposiX1(5)> X"00" then
                                        cont<='0';
                                else
                                        cont<='1';

                                end if;
----
        when Max_pso =>
                                if Aptitud_mejorposi(2)> (Aptitud_mejorposi(1) and
Aptitud_mejorposi(3) and Aptitud_mejorposi(4) and Aptitud_mejorposi(5)) then
                                        Aptitud_mejorposg<= Aptitud_mejorposi(2);
                                        X1best <= mejorposiX1(2);
                                end if;

                                if Aptitud_mejorposi(3)> (Aptitud_mejorposi(1) and
Aptitud_mejorposi(2) and Aptitud_mejorposi(4) and Aptitud_mejorposi(5)) then
                                        Aptitud_mejorposg<= Aptitud_mejorposi(3);
                                        X1best <= mejorposiX1(3);
                                end if;
```

```
                        if Aptitud_mejorposi(4)> (Aptitud_mejorposi(1) and
Aptitud_mejorposi(2) and Aptitud_mejorposi(3) and Aptitud_mejorposi(5)) then
                                Aptitud_mejorposg<= Aptitud_mejorposi(4);
                                X1best <= mejorposiX1(4);
                        end if;

                        if Aptitud_mejorposi(5)> (Aptitud_mejorposi(1) and
Aptitud_mejorposi(2) and Aptitud_mejorposi(3) and Aptitud_mejorposi(4)) then
                                Aptitud_mejorposg<= Aptitud_mejorposi(5);
                                X1best <= mejorposiX1(5);
                        end if;

                        if Aptitud_mejorposi(1)> (Aptitud_mejorposi(2) and
Aptitud_mejorposi(3) and Aptitud_mejorposi(4) and Aptitud_mejorposi(5)) then
                                Aptitud_mejorposg<= Aptitud_mejorposi(1);
                                X1best <= mejorposiX1(1);
                        end if;
                Control<=Control+1;
                        if control = "11" then
                                Cont<='1';
                                Control<="00";

                        else
                                Cont<='0';
                        end if;
            when Vel_inicial=>
                        feedback_1<= not(X1(1)(3) xor X1(1)(2));
                        X1(1)<= (X1(1)+ (X1(1)(6 downto 0) & feedback_1));
-- Se genera la primer variable X1

                        feedback_2<= not(X1(2)(3) xor X1(2)(2));
                        X1(2)<= (X1(2)+ (X1(2)(6 downto 0) & feedback_1));
-- Se genera la primer variable X1

                        feedback_3<= not(X1(3)(3) xor X1(3)(2));
                        X1(3)<= (X1(3)+ (X1(3)(6 downto 0) & feedback_1));
-- Se genera la primer variable X1


                        feedback_4<= not(X1(4)(3) xor X1(4)(2));
                        X1(4)<= (X1(4)+ (X1(4)(6 downto 0) & feedback_1));
-- Se genera la primer variable X1

                        feedback_5<= not(X1(5)(3) xor X1(5)(2));
```

```
                                X1(5)<= (X1(5)+ (X1(5)(6 downto 0) & feedback_1));
-- Se genera la primer variable X1

            Control<=Control+1;
                                if control = "11" then
                                        Cont<='1';
                                        Control<="00";
                                else
                                        Cont<='0';
                                end if;
        when Xi_lim=>
                                ------------------------------
                                if X1(1)<Xmax then
                                        if X1(1)<Xmin then
                                                X1(1)<=Xmin;
                                        else
                                                X1(1)<=X1(1);
                                                end if;
                                else
                                        X1(1)<=Xmin;
                                end if;
                                ------------------------------
                                if X1(2)<Xmax then
                                        if X1(2)<Xmin then
                                                X1(2)<=Xmin;
                                        else
                                                X1(2)<=X1(2);
                                                end if;
                                else
                                                X1(2)<=Xmin+X"10";
                                end if;
                                ------------------------------
                                if X1(3)<Xmax then
                                                if X1(3)<Xmin then
                                                        X1(3)<=Xmin;
                                                else
                                                        X1(3)<=X1(3);
                                                end if;
                                else
                                                X1(3)<=Xmin+X"05";
                                end if;
                                ------------------------------
                                if X1(4)<Xmax then
                                                if X1(4)<Xmin then
```

```
                                        X1(4)<=Xmin;
                                else
                                        X1(4)<=X1(4);
                                end if;
                        else
                                X1(4)<=Xmin+X"09";
                        end if;
                        ------------------------------
                        if X1(5)<Xmax then
                                if X1(5)<Xmin then
                                        X1(5)<=Xmin;
                                else
                                        X1(5)<=X1(5);
                                end if;
                        else
                                X1(5)<=Xmin;
                        end if;
                        ------------------------------
                Control<=Control+1;
                        if control >= "10" then
                                ciclo_2<='1';
                                ciclo<='0';
                                Cont<='0';
                                Control<="00";
                                Ii<=1;Ij<=1; -- se inicializa los estados de
suma y funcion.
                        else
                                Cont<='1';
                        end if;
------
                when Xi_Max=>
                        if Aptitud_Xi(1)> Aptitud_mejorposi(1) then
                                Aptitud_mejorposi(1)<=Aptitud_Xi(1);
                                mejorposiX1(1)<=X1(1);
                        end if;
                        if Aptitud_Xi(2)> Aptitud_mejorposi(2) then
                                Aptitud_mejorposi(2)<=Aptitud_Xi(2);
                                mejorposiX1(2)<=X1(2);
                        end if;
                        if Aptitud_Xi(3)> Aptitud_mejorposi(3) then
                                Aptitud_mejorposi(3)<=Aptitud_Xi(3);
                                mejorposiX1(3)<=X1(3);
                        end if;
                        if Aptitud_Xi(4)> Aptitud_mejorposi(4) then
```

```
                    Aptitud_mejorposi(4)<=Aptitud_Xi(4);
                    mejorposiX1(4)<=X1(4);
            end if;
            if Aptitud_Xi(5)> Aptitud_mejorposi(5) then
                    Aptitud_mejorposi(5)<=Aptitud_Xi(5);
                    mejorposiX1(5)<=X1(5);
            end if;

            Control<=Control+1;
                    if control = "11" then
                            --ciclo<='1';
                            Cont<='1';
                            Control<="00";
                    else
                            Cont<='0';
                    end if;
    when Global=>
                    if Aptitud_mejorposi(1) > Aptitud_mejorposg then
                            Aptitud_mejorposg<=Aptitud_mejorposi(1);
                            X1best<=mejorposiX1(1);
                    end if;
                    if Aptitud_mejorposi(2) > Aptitud_mejorposg then
                            Aptitud_mejorposg<=Aptitud_mejorposi(2);
                            X1best<=mejorposiX1(2);
                    end if;
                    if Aptitud_mejorposi(3) > Aptitud_mejorposg then
                            Aptitud_mejorposg<=Aptitud_mejorposi(3);
                            X1best<=mejorposiX1(3);
                    end if;
                    if Aptitud_mejorposi(4) > Aptitud_mejorposg then
                            Aptitud_mejorposg<=Aptitud_mejorposi(4);
                            X1best<=mejorposiX1(4);
                    end if;
                    if Aptitud_mejorposi(5) > Aptitud_mejorposg then
                            Aptitud_mejorposg<=Aptitud_mejorposi(5);
                            X1best<=mejorposiX1(5);
                    end if;
    Control<=Control+1;
                    if control = "11" then
                            Cont<='1';
                            Control<="00";
                    else
                            Cont<='0';
                    end if;
```

```
                when Vel_completa=>

                                feedback_1 <= not(X1(1)(3) xor X1(1)(2));
                                Vel1a_1 := (w*X1(1))+ (c1*(Aptitud_mejorposi(1)-X1(1)))
+(c2*(Aptitud_mejorposg-X1(1)));
                                X1(1)<=X1(1)+("00000" & Vel1a_1(1 downto 0));
                                feedback_2 <= not(X1(2)(3) xor X1(2)(2));
                                Vel1a_2 := (w*X1(2))+ (c1*(Aptitud_mejorposi(2)-X1(2)))
+(c2*(Aptitud_mejorposg-X1(2)));
                                X1(2)<=X1(2)+("00000" & Vel1a_2(1 downto 0));--Vel1(1);
--Vel1(1); --Actualizo el valor dela posicion.}


                                feedback_3 <= not(X1(3)(3) xor X1(3)(2));
                                Vel1a_3 := (w*X1(3))+ (c1*(Aptitud_mejorposi(3)-X1(3)))
+(c2*(Aptitud_mejorposg-X1(3)));
                                X1(3)<=X1(3)+("00000" & Vel1a_3(9 downto 7));--Vel1(1);
                                feedback_4 <= not(X1(4)(3) xor X1(4)(2));
                                Vel1a_4 := (w*X1(4))+ (c1*(Aptitud_mejorposi(4)-X1(4)))
+(c2*(Aptitud_mejorposg-X1(4)));
                                X1(4)<=X1(4)+("00000" & Vel1a_4(9 downto 7));--Vel1(1);

                                feedback_5 <= not(X1(5)(3) xor X1(5)(2));
                                Vel1a_5 := (w*X1(5))+ (c1*(Aptitud_mejorposi(5)-X1(5)))
+(c2*(Aptitud_mejorposg-X1(5)));
                                X1(5)<=X1(5)+("00000" & Vel1a_5(1 downto 0));--Vel1(1)
                               Control<=Control+1;
                                        if control = "11" then
                                                Cont<='1';
                                                Control<="00";
                                                it<=it+1;
                                        else
                                                Cont<='0';
                                        end if;
--------------------------------------------------------------------------------
                when Iteracion=>
                        if it>=20 then
                                ciclo<='0';
                                Cont<='0';
                        else

                                ciclo<='1';
                                Cont<='1';
                        end if;
```

```
--------------------------------------------------------------------------------
                when rango =>

                Control<=Control+1;
                if control = "11" then
                        if  U=5 then
                                        Umbral_5<=X1best;
                                        Cont<='1';
                                        ciclo<='1';
                                        ciclo_2<='1';
                        elsif U=4 then
                                        Umbral_4<=X1best;
                                        Xmin<=X"0A"; -- 10
                                        Xmax<=X"C8"; -- 200
                                        ciclo<='0';
                                        Cont<='0';
                                        ciclo_2<='0';
                                        U<=5;
                        elsif U=3 then
                                        Umbral_3<=X1best;
                                        Xmin<=X"AC"; --172
                                        Xmax<=X"BE"; --190
                                        ciclo<='0';
                                        Cont<='0';
                                        ciclo_2<='0';
                                        U<=4;
                        elsif U=2 then
                                        Umbral_2<=X1best;
                                        Xmin<=X"90"; --144
                                        Xmax<=X"AB"; -- 171
                                        ciclo<='0';
                                        Cont<='0';
                                        ciclo_2<='0';
                                        U<=3;
                        elsif U=1 then
                                        Umbral_1<=X1best;
                                        Xmin<=X"57"; --87
                                        Xmax<=X"8F"; -- 143
                                        ciclo<='0';
                                        Cont<='0';
                                        ciclo_2<='0';
                                        U<=2;
                        elsif U=0 then
                                        Umbral_0<=X1best;
```

```
                                    Xmin<=X"3B"; --59
                                    Xmax<=X"56"; -- 86
                                    ciclo<='0';
                                    Cont<='0';
                                    ciclo_2<='0';
                                    U<=1;
                        end if;
                else
                        ciclo<='0';
                        Cont<='0';
                        ciclo_2<='1';

                                If Umbral_0=X"00" THEN
                                        Umbral_0<=X"1D";
                                END IF;
                                If Umbral_1=X"00" THEN
                                        Umbral_1<=X"3B";
                                END IF;
                                If Umbral_2=X"00" THEN
                                        Umbral_2<=X"57";
                                END IF;
                                If Umbral_3=X"00" THEN
                                        Umbral_3<=X"90";
                                END IF;
                                If Umbral_4=X"00" THEN
                                        Umbral_4<=X"AC";
                                END IF;
                                If Umbral_5=X"00" THEN
                                        Umbral_5<=X"0A";
                                END IF;


                end if;

--------------------------------------------------------------------------------
                when fin=>

                        On_tx<='1';
  end case;
end if;
end process;
--------------------------------------------------------------------------------
end Behavioral;
```

Módulo de transmision para el Algoritmo PSO – Transmision.vhd (ISE - Xilinx)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Trans is
   Port ( clk,rst_a,on_tx: in  STD_LOGIC;
                        Umbral_0,Umbral_1,Umbral_2,Umbral_3,Umbral_4,Umbral_5 : in
STD_LOGIC_VECTOR(7 DOWNTO 0);

                         tx : out  STD_LOGIC);
end Trans;

architecture Arq_Trans of Trans is

signal regtx:                STD_LOGIC_VECTOR( 7 downto 0);
signal cnttx:                STD_LOGIC_VECTOR(15 downto 0):="0000000000000000";
signal ttx:                          STD_LOGIC_VECTOR( 3 downto 0):="0000";
signal caracter:             STD_LOGIC_VECTOR( 7 downto 0);
constant baudtx:             STD_LOGIC_VECTOR(15 downto 0):="0000001101100100";
--115200 Bauds Funciona
begin

-- Reloj de transmisión
process (clk,rst_a,cnttx,caracter,ttx)begin
        if (rst_a='1' )then
                cnttx<= baudtx - 1;
                ttx <= "0000";
                caracter <= "00000000";
        elsif(caracter = "00000111") then --n 00001111 caracter + 1
                cnttx<= baudtx - 1;
                ttx <= "0000";
        elsif (clk'event and clk='1')then
                        cnttx<=cnttx+1;
                        if (cnttx=baudtx)then
                                ttx<=ttx+1;
                                cnttx<=(others=>'0');
                                if(ttx="1010")then
                                        caracter<=caracter+1;
```

```
                                    ttx <= "0000";
                                end if;
                            end if;
        end if;
end process;

-- Asignacion de Caracter

with caracter select
        regtx <=                    Umbral_0    when "00000000", --0
                                        Umbral_1    when "00000001", --1
                                        Umbral_2    when "00000010", --2
                                        Umbral_3    when "00000011", --3
                                        Umbral_4    when "00000100", --4
                                        Umbral_5 when others; --5
-- Protocolo de transmisión
with ttx select
                tx          <=      '1'             when "0000",
                                    '0'             when "0001", --bit de start
                                    regtx(0)        when "0010", --inicia transmision de
dato
                                    regtx(1)        when "0011",
                                    regtx(2) when "0100",
                                    regtx(3)        when "0101",
                                    regtx(4)        when "0110",
                                    regtx(5)        when "0111",
                                    regtx(6)        when "1000",
                                    regtx(7) when "1001",       --fin de transmision de
dato
                                    '1'             when "1010", --bit de stop
                                    '1'             when others;

end Arq_Trans;
```

Algoritmo ABC –abc.vhd (ISE - Xilinx)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;
use work.ABC_Package.all;

ENTITY abc IS
```

```
  PORT ( clk,rst : IN STD_LOGIC;
                          OUT_1:out std_logic_vector(15 downto 0);
                          OUT_2:out std_logic_vector(15 downto 0);
                          OUT_3:out std_logic_vector(15 downto 0);
                          OUT_4:out std_logic_vector(15 downto 0)
                          );

END abc;



ARCHITECTURE RTL OF abc IS

type s1_a is array (0 to 99) of std_logic_vector(15 downto 0);
constant S1: s1_a:= (
"0000000100100001","1111110111000011","0000001101001100","1111101110110101","0000010100110011","1111100111111101","0000011010110001","1111100011000001",
"0000011110100110","1111100000011000","0000011111111111","1111100000010000","0000011110110101","1111100010101010","0000011011001110","1111100111011010",
"0000010101011100","1111101110000111","0000001101111101","1111110110001111","0000000101010110","1111111111001001","1111111100010100","0000001000000111",
"1111110011100101","0000010000011100","1111101011110101","0000010111011101","1111100101101100","0000011100100110","1111100001101001","0000011111011110",
"1111100000000001","0000011111110101","1111100000111100","0000011101101010","1111100100010101","0000011001000111","1111101001111011","0000010010100101",
"1111110001010010","0000001010100011","1111111001110011","0000000001101100","0000000010110100","1111111000101100","0000001011100111","1111110000010010",
"0000010011011111","1111101001000111","0000011001110011","1111100011110010","0000011110000100","1111100000101100","0000011111111011","1111100000000101",
"0000011111010000","1111100010000010","0000011100000101","1111100110010111","0000010110101011","1111101100101111","0000001111011101","1111110100101001",
"0000000111000001","1111111101011101","1111111110000001","0000000110011101","1111110101001011","0000001110111101","1111101101001100","0000010110010001",
```

```
"1111100110101101","0000011011110011","1111100010001111","000001111100011
1","1111100000001000","0000011111111101","1111100000100100","0000011110010
000",
"1111100011100001","0000011010001000","1111101000101110","000001001111101
1","1111101111110010","0000001100001001","1111111000001001","0000000011011
001",
"0000000001001000","1111111010010111","0000001010000001","111111000111001
0","0000010010000111","1111101010010110","0000011000110000","1111100100100
111",
"0000011101011100","1111100001000101","0000011111110001","111110000000000
0");

type s2_a is array (0 to 99) of std_logic_vector(15 downto 0);
constant S2: s2_a:= (
"1111100000000000","1111111111101101","0000011111111111","000000000010010
0","1111100000000000","1111111111001001","0000011111111111","0000000001001
000",
"1111100000000001","1111111110100101","0000011111111101","000000000110110
0","1111100000000011","1111111110000001","0000011111111011","0000000010010
000",
"1111100000000101","1111111101011101","0000011111111000","000000001011010
1","1111100000001000","1111111100111000","0000011111110101","0000000011011
001",
"1111100000001100","1111111100010100","0000011111110001","000000001111110
1","1111100000010000","1111111011110000","0000011111101100","0000000100100
001",
"1111100000010101","1111111011001100","0000011111100111","000000010100010
0","1111100000011011","1111111010101001","0000011111100001","0000000101101
000",
"1111100000100001","1111111010000101","0000011111011011","000000011000110
0","1111100000101000","1111111001100001","0000011111010011","0000000110101
111",
"1111100000101111","1111111000111110","0000011111001100","000000011101001
1","1111100000111000","1111111000011011","0000011111000011","0000000111110
110",
"1111100001000000","1111110111111000","0000011110111010","000000100001100
1","1111100001001010","1111110111010101","0000011110110000","0000001000111
100",
"1111100001010100","1111110110110010","0000011110100110","000000100101111
1","1111100001011110","1111110110001111","0000011110011011","0000001010000
001",
"1111100001101001","1111110101101101","0000011110010000","000000101010001
1","1111100001110101","1111110101001010","0000011110000100","0000001011000
110",
```

```
"1111100010000010","1111110100101000","0000011101110111","0000001011100111","1111100010001111","1111110100000111","0000011101101010","0000001100001001",
"1111100010011100","1111110011100101","0000011101011100","0000001100101011","1111100010101011","1111110011000100","0000011101001101","0000001101001100",
"1111100010111001","1111110010100011","0000011100111110","0000001101101101");

type mix1_a is array (0 to 99) of std_logic_vector(15 downto 0); --mix1=s1+(s2*.5)
constant M_1: mix1_a:= (
"1111110100100001","1111110110111010","0000011101001011","1111101111000111","0000000100110100","1111100111100010","0000101010110000","1111100011100101",
"0000001110100111","1111011111101011","0000101111111110","1111100001000111","0000001110110111","1111100001101011","0000101011001100","1111101000100011",
"0000000101011111","1111101100110110","0000011101111001","1111110111101010","1111110101011011","1111111101100110","0000001001110100",
"1111100011101100","0000001110100111","1111111011101110","0000011001011100","1111010101110101","0000011010011111","1111110001100000","0000100001101110",
"1111010000001100","0000011101011011","1111110000110000","0000100000001100","1111010100100011","0000010110011100","1111111001101100","0000010101011001",
"1111100001100010","0000000111100110","0000001001100001","0000000100110010","1111110011001001","1111110101011101","0000011011010001","1111110011101010",
"0000000011110111","1111100101100111","0000101001011001","1111100111011011","0000001110100000","1111011100111001","0000101111011101","1111100100000000",
"0000001111110000","1111011101111110","0000101011100010","1111101010100011","0000000111010000","1111101000011001","0000011110110110","1111111001000111",
"1111110111101011","1111111000110110","0000001101010100","0000001011001101","1111100101111010","0000001010000101","1111111100011010","0000011011010010",
"1111010111100010","0000010110101010","1111110001010111","0000100100011001","1111010001000011","0000011010100011","1111101111100110","0000100011110011",
"1111010100100010","0000010100011101","1111110111101010","0000011001101111","1111100000111010","0000000110001101","0000000110111110","0000001001011101",
```

```
"1111110010010110","1111110100001010","0000011000101111","111111100000100
0","0000000011011100","1111100011111000","0000100111010111","1111101011001
110",
"0000001110111001","1111011010010111","0000101110010000","111110011011011
1");


type mix2_a is array (0 to 99) of std_logic_vector(15 downto 0); -- mix2=s2+(s1*.5);

constant M_2: mix2_a:= (
"1111100010010000","1111111011001111","0000100110100101","111111011111111
0","1111101010011010","1111110011001000","0000101101010111","1111110010101
001",
"1111101111010100","1111101110110001","0000101111111101","111111000111010
1","1111101111011110","1111101111010110","0000101101100010","1111110101111
110",
"1111101010110100","1111110100100000","0000100110110111","111111110111110
0","1111100010110100","1111111100011101","0000011101111111","0000000111011
101",
"1111011001111111","0000000100100011","0000010101101100","000000111110101
1","1111010011000111","0000001010000100","0000010000100001","0000010100010
000",
"1111010000010110","0000001011000111","0000010000000101","000001001111100
1","1111010010100110","0000000111001101","0000010100011111","0000001110111
011",
"1111011001001010","1111111111010111","0000011100010100","000000011100001
0","1111100010000010","1111110101111000","0000100101000111","1111111110111
000",
"1111101010011111","1111101101100010","0000101100000101","111111100100110
0","1111101111111010","1111101000110001","0000101111000001","1111110111111
001",
"1111110000101000","1111101000111001","0000101100111101","111111101110010
1","1111101100011111","1111101101101100","0000100110011111","0000000011010
000",
"1111100100110100","1111110101100000","0000011101100111","000000110010111
0","1111011100000100","1111111101101110","0000010101000001","0000010101001
010",
"1111010101000000","0000000011100111","0000001111010111","000001101000011
1","1111010001111010","0000000101001001","0000001110010110","0000011010001
110",
"1111010011110010","0000000001101101","0000010010001110","000001010110010
1","1111011010001000","1111111010001011","0000011001101110","0000001101110
110",
```

```
"1111100011000001","1111110000110001","0000100010011100","0000000101100100","1111101011101110","1111101000001111","0000101001100110","1111111111100000",
"1111110001100111","1111100011000110","0000101100110111","1111111101101101");

--constant CS
signal control:std_logic_vector(3 downto 0):="0000";
signal clk_1,clk_2:integer range 0 to 10:=0 ;
signal feedback_1_bee1,feedback_2_bee1,feedback_3_bee1,feedback_4_bee1:std_logic:='0';
signal feedback_1_bee2,feedback_2_bee2,feedback_3_bee2,feedback_4_bee2:std_logic:='0';
signal feedback_1_bee3,feedback_2_bee3,feedback_3_bee3,feedback_4_bee3:std_logic:='0';
signal feedback_1_bee4,feedback_2_bee4,feedback_3_bee4,feedback_4_bee4:std_logic:='0';
signal feedback_1_bee5,feedback_2_bee5,feedback_3_bee5,feedback_4_bee5:std_logic:='0';

signal rand_1,rand_2,rand_3,rand_4 :std_logic_vector(9 downto 0);
signal i: integer range 8 to 70:=8 ;
signal i_a,i_a0,i_a1,i_b: integer range 0 to 99:=0 ;
signal X_a,X_b,X_c,X_d: std_logic_vector (25 downto 0):="00000000000000000000000000";
constant Xmin:std_logic_vector(15 downto 0):="1111100000000000";
constant Xmax:std_logic_vector(15 downto 0):="0000100000000000";
constant div: std_logic_vector(15 downto 0):= "0111100000000000" ;--"1100100";
type X1_a is array (0 to 19) of std_logic_vector(15 downto 0); --  Colonia de abejas # 5
        signal X1: X1_a:= (others => Xmin);
signal  M_1b1_bee1,M_1b2_bee1,M_2b1_bee1,M_2b2_bee1  :std_logic_vector(31 downto 0):=X"00000000";
signal  M_1b1_bee2,M_1b2_bee2,M_2b1_bee2,M_2b2_bee2  :std_logic_vector(31 downto 0):=X"00000000";
signal  M_1b1_bee3,M_1b2_bee3,M_2b1_bee3,M_2b2_bee3  :std_logic_vector(31 downto 0):=X"00000000";
signal  M_1b1_bee4,M_1b2_bee4,M_2b1_bee4,M_2b2_bee4  :std_logic_vector(31 downto 0):=X"00000000";
signal  M_1b1_bee5,M_1b2_bee5,M_2b1_bee5,M_2b2_bee5  :std_logic_vector(31 downto 0):=X"00000000";
-----------------------------------------------------------------------
type M1_a is array (0 to 99) of std_logic_vector(15 downto 0);
signal M_1a_bee1:  M1_a := (others => X"0000");
```

```
type M2_a is array (0 to 99) of std_logic_vector(15 downto 0);
signal M_2a_bee1:  M2_a := (others => X"0000");
signal M_1a_bee2:  M1_a := (others => X"0000");
signal M_2a_bee2:  M2_a := (others => X"0000");
signal M_1a_bee3:  M1_a := (others => X"0000");
signal M_2a_bee3:  M2_a := (others => X"0000");
signal M_1a_bee4:  M1_a := (others => X"0000");
signal M_2a_bee4:  M2_a := (others => X"0000");
signal M_1a_bee5:  M1_a := (others => X"0000");
signal M_2a_bee5:  M2_a := (others => X"0000");


---------- bee # 1 ----------
signal M1_S1_bee1,M2_S2_bee1 :std_logic_vector(15 downto 0):=X"0000";
signal M1_S1_a_bee1,M2_S2_a_bee1 :std_logic_vector(31 downto 0):=X"00000000";
signal M1_S1_b_bee1,M2_S2_b_bee1 :std_logic_vector(15 downto 0):=X"0000";
---------- bee # 2 ----------
signal M1_S1_bee2,M2_S2_bee2 :std_logic_vector(15 downto 0):=X"0000";
signal M1_S1_a_bee2,M2_S2_a_bee2 :std_logic_vector(31 downto 0):=X"00000000";
signal M1_S1_b_bee2,M2_S2_b_bee2 :std_logic_vector(15 downto 0):=X"0000";
---------- bee # 3 ----------
signal M1_S1_bee3,M2_S2_bee3 :std_logic_vector(15 downto 0):=X"0000";
signal M1_S1_a_bee3,M2_S2_a_bee3 :std_logic_vector(31 downto 0):=X"00000000";
signal M1_S1_b_bee3,M2_S2_b_bee3 :std_logic_vector(15 downto 0):=X"0000";
---------- bee # 4 ----------
signal M1_S1_bee4,M2_S2_bee4 :std_logic_vector(15 downto 0):=X"0000";
signal M1_S1_a_bee4,M2_S2_a_bee4 :std_logic_vector(31 downto 0):=X"00000000";
signal M1_S1_b_bee4,M2_S2_b_bee4 :std_logic_vector(15 downto 0):=X"0000";
---------- bee # 5 ----------
signal M1_S1_bee5,M2_S2_bee5 :std_logic_vector(15 downto 0):=X"0000";
signal M1_S1_a_bee5,M2_S2_a_bee5 :std_logic_vector(31 downto 0):=X"00000000";
signal M1_S1_b_bee5,M2_S2_b_bee5 :std_logic_vector(15 downto 0):=X"0000";
-----------------------------------------
signal M_S_bee1 :std_logic_vector(15 downto 0):=X"0000";
signal M_S_bee2 :std_logic_vector(15 downto 0):=X"0000";
signal M_S_bee3 :std_logic_vector(15 downto 0):=X"0000";
signal M_S_bee4 :std_logic_vector(15 downto 0):=X"0000";
signal M_S_bee5 :std_logic_vector(15 downto 0):=X"0000";
signal                          M_S_bee1a,M_S_bee2a,M_S_bee3a,M_S_bee4a,M_S_bee5a
:std_logic_vector(16 downto 0):="00000000000000000";


 Signal Xbest1,Xbest2,Xbest3,Xbest4 :std_logic_vector(15 downto 0):=X"0000";
 Signal  food_source_best :std_logic_vector(16 downto 0):="00000000000000000";
```

```
signal cont_ECM:std_logic_vector(1 downto 0):="00";
        Signal ECM_a :std_logic_vector(15 downto 0):=X"0000";--std_logic_vector:=
(others => X"0000");--
        Signal Fitness_sig :std_logic_vector(15 downto 0):=X"0000";--std_logic_vector:=
(others => X"0000");--
---------------------------------------------------------------------------------

signal X1a,X2a,X3a,X4a,X5a :std_logic_vector(15 downto 0):=X"0000";

signal poss_1 :std_logic_vector(1 downto 0):="00";
signal poss_2 :std_logic_vector(1 downto 0):="01";
signal poss_3 :std_logic_vector(1 downto 0):="10";
signal poss_4 :std_logic_vector(1 downto 0):="11";
signal poss_5 :std_logic_vector(1 downto 0):="10";


signal      v_1_bee1,v_2_bee1,v_3_bee1,v_4_bee1      :std_logic_vector(15      downto
0):=X"0000";
signal      v_1_bee2,v_2_bee2,v_3_bee2,v_4_bee2      :std_logic_vector(15      downto
0):=X"0000";
signal      v_1_bee3,v_2_bee3,v_3_bee3,v_4_bee3      :std_logic_vector(15      downto
0):=X"0000";
signal      v_1_bee4,v_2_bee4,v_3_bee4,v_4_bee4      :std_logic_vector(15      downto
0):=X"0000";
signal      v_1_bee5,v_2_bee5,v_3_bee5,v_4_bee5      :std_logic_vector(15      downto
0):=X"0000";
signal                        M_S_bee1b,M_S_bee2b,M_S_bee3b,M_S_bee4b,M_S_bee5b
:std_logic_vector(16 downto 0):="00000000000000000";
signal trip_bee1,trip_bee2,trip_bee3,trip_bee4,trip_bee5 : integer range 0 to 1000:=0 ;
signal iteracion : integer range 0 to 10000:=0;
---------------------------------------------------------------------------------


type                                    mis_estados                                    is
(init,init_randx,Mult_mixA,ECM,fitness,Bee_employed_1,Bee_employed_2,Bee_employ
ed_3_mult,Bee_employed_ECM,Bee_employed_5,
Bee_employed_6,bee_onlooker_1,bee_onlooker_2,bee_onlooker_3,bee_onlooker_4,b
ee_onlooker_5,bee_onlooker_6,bee_explorer,fin);--MemorizeBestSource);
signal Estado_Actual,Estado_Siguiente: mis_estados;
BEGIN

--     initial;
```

```
--     MemorizeBestSource;
--     for iter in 0 to maxCycle - 1 loop
--       SendEmployedBees;
--       CalculateProbabilities;
--       SendOnlookerBees;
--       MemorizeBestSource;
--       SendScoutBees;
--     end loop;
--     GlobalMins(run) := GlobalMin;
--     mean:=mean + GlobalMin;

Secuanciador:process (clk,rst)
begin
          if rst = '1' then
     Estado_Actual<=init;

  elsif (clk'event and clk = '1') then
                    Estado_Actual<=Estado_Siguiente;
   end if;

end process Secuanciador;
Maq_est:process (clk,rst,Estado_Actual,control)
begin
          case (Estado_Actual) is
                    when init => -- Initial;
                              Estado_Siguiente<=init_randx;
     when init_randx => -- Initial;
                              if control="0001" then
                                        Estado_Siguiente<=Mult_mixA;
                              elsif control="0000" then
                                        Estado_Siguiente<=init_randx;
                              end if;
    when Mult_mixA=>
                              if control="0001" then
                                        Estado_Siguiente <= Mult_mixA;--MemorizeBestSource;
                              elsif control="0010" then
                                        Estado_Siguiente <=ECM;
                              end if;
                    when ECM=>
                              if control="0010" then
                                        Estado_Siguiente <=ECM;
                              elsif control="0011" then
                                        Estado_Siguiente <=fitness;
                              end if;
```

```
when fitness=>
        if control="0011" then
                Estado_Siguiente <=fitness;
        elsif control="0100" then
                Estado_Siguiente <=Bee_employed_1;
        end if;

when Bee_employed_1=>
        if control="0101" then
                Estado_Siguiente <=Bee_employed_2;
        elsif control="0100" then
                Estado_Siguiente <=Bee_employed_1;
        end if;
when Bee_employed_2 =>
        if control="0101" then
                Estado_Siguiente <=Bee_employed_2;
        elsif control="0110" then
                Estado_Siguiente <=Bee_employed_3_mult;
        end if;
when   Bee_employed_3_mult=>
        if control="0111" then
                Estado_Siguiente <=Bee_employed_ECM;
        elsif control="0110" then
                Estado_Siguiente <=Bee_employed_3_mult;
        end if;
when   Bee_employed_ECM=>
        if control="1000" then
                Estado_Siguiente <=Bee_employed_5;
        elsif control="0111" then
                Estado_Siguiente <=Bee_employed_ECM;
        end if;

when   Bee_employed_5=>
        if control="1000" then
                Estado_Siguiente <=Bee_employed_5;
        elsif control="1001" then
                Estado_Siguiente <=Bee_employed_6;
        end if;
when   Bee_employed_6=>
        if control="1010" then
                Estado_Siguiente <=bee_onlooker_1;
        elsif control="1001" then
                Estado_Siguiente <=Bee_employed_6;
        end if;
```

```
when  bee_onlooker_1=>
      if control="1010" then
            Estado_Siguiente <=bee_onlooker_1;
      elsif control="1011" then
            Estado_Siguiente <=bee_onlooker_2;
      end if;
when  bee_onlooker_2=>
      if control="1011" then
            Estado_Siguiente <=bee_onlooker_2;
      elsif control="1100" then
            Estado_Siguiente <=bee_onlooker_3;
      end if;
when  bee_onlooker_3=>
      if control="1100" then
            Estado_Siguiente <=bee_onlooker_3;
      elsif control="1101" then
            Estado_Siguiente <=bee_onlooker_4;
      end if;

when  bee_onlooker_4=>
      if control="1101" then
            Estado_Siguiente <=bee_onlooker_4;
      elsif control="1110" then
            Estado_Siguiente <=bee_onlooker_5;
      end if;
when  bee_onlooker_5=>
      if control="1110" then
            Estado_Siguiente <=bee_onlooker_5;
      elsif control="1111" then
            Estado_Siguiente <=bee_onlooker_6;
      end if;
when  bee_onlooker_6=>
      if control="1111" then
            Estado_Siguiente <=bee_onlooker_6;
      elsif control="0111" then
            Estado_Siguiente <=bee_explorer;
      end if;
when  bee_explorer=>
      if control="0111" then
            Estado_Siguiente <=bee_explorer;
      elsif control="0011" then
            Estado_Siguiente <=fin;
      end if;
```

```
                when fin=>
                        if control="0101" then
                                Estado_Siguiente <=Bee_employed_1;
                        elsif control="0000" then
                                Estado_Siguiente <=fin;
                        end if;

        end case;

end process Maq_est;
initial_1:process (clk,rst,control,clk_1,i,estado_siguiente)
begin
--if Estado_Siguiente =init_randx then
if (clk'event and clk = '1')then-- then
        case Estado_Actual is
                when init =>
                                        i<=8;
                                        clk_1<=0;
                                        feedback_1_bee1  <='0';  feedback_2_bee1  <='0';
feedback_3_bee1 <='0'; feedback_4_bee1 <='0';
                                        feedback_1_bee2  <='0';  feedback_2_bee2  <='0';
feedback_3_bee2 <='0'; feedback_4_bee2 <='0';
                                        feedback_1_bee3  <='0';  feedback_2_bee3  <='0';
feedback_3_bee3 <='0'; feedback_4_bee3 <='0';
                                        feedback_1_bee4  <='0';  feedback_2_bee4  <='0';
feedback_3_bee4 <='0'; feedback_4_bee4 <='0';
                                        feedback_1_bee5  <='0';  feedback_2_bee5  <='0';
feedback_3_bee5 <='0'; feedback_4_bee5 <='0';
                                                X1(0)<="0000100000000000"; -- Bee #1
                                                X1(1)<=X"0000"; -- Bee #1
                                                X1(2)<=X"0000"; -- Bee #1
                                                X1(3)<="0000100000000000"; -- Bee #1

                                                X1(4)<="0000101011001010"; -- Bee #2
                                                X1(5)<="0000010100000000";-- Bee #2
                                                X1(6)<="0000000010111000";-- Bee #2
                                                X1(7)<="0000001001001100";-- Bee #2
    when init_randx =>
                        clk_1<=clk_1+1;
                        feedback_1_bee3 <= not(X1(4)(7) xor X1(7)(3));
                        feedback_2_bee3 <= not(X1(4)(3) xor X1(5)(4));
                        feedback_3_bee3 <= not(X1(5)(5) xor X1(6)(0));
```

```
                    feedback_4_bee3 <= not(X1(6)(2) xnor X1(6)(1));

                    X1(8)<= X1(15)(14 downto 0) & feedback_1_bee3;--(Xmax-Xmin)*
rand_1 ;
                    X1(9)<= X1(17)(14 downto 0) & feedback_2_bee3 ;--(Xmax-Xmin)*
rand_2 ;
                    X1(10)<= X1(18)(14 downto 0) & feedback_3_bee3;--(Xmax-Xmin)*
rand_3 ;
                    X1(11)<= X1(19)(14 downto 0) & feedback_4_bee3;--(Xmax-Xmin)*
rand_4 ;



                    feedback_1_bee4 <= not(X1(8)(8) xor X1(10)(3));
                    feedback_2_bee4 <= not(X1(7)(9) xor X1(11)(4));
                    feedback_3_bee4 <= (X1(9)(2) xor X1(7)(0));
                    feedback_4_bee4 <= not(X1(8)(2) xnor X1(11)(1));

                    X1(12)<= X1(8)(14 downto 0) & feedback_1_bee4;--(Xmax-Xmin)*
rand_1 ;
                    X1(13)<= X1(9)(14 downto 0) & feedback_2_bee4 ;--(Xmax-Xmin)*
rand_2 ;
                    X1(14)<= X1(10)(14 downto 0) & feedback_3_bee4;--(Xmax-Xmin)*
rand_3 ;
                    X1(15)<= X1(11)(14 downto 0) & feedback_4_bee4;--(Xmax-Xmin)*
rand_4 ;

                    feedback_1_bee5 <= not(X1(12)(8) xor X1(13)(3));
                    feedback_2_bee5 <= not(X1(14)(9) xor X1(14)(4));
                    feedback_3_bee5 <= (X1(9)(15) xor X1(9)(0));
                    feedback_4_bee5 <= not(X1(12)(2) xnor X1(11)(1));

                    X1(16)<= X1(16)(14 downto 0) & feedback_1_bee5;--(Xmax-Xmin)*
rand_1 ;
                    X1(17)<= X1(17)(14 downto 0) & feedback_2_bee5 ;--(Xmax-
Xmin)* rand_2 ;
                    X1(18)<= X1(18)(14 downto 0) & feedback_3_bee5;--(Xmax-Xmin)*
rand_3 ;
                    X1(19)<= X1(8)(14 downto 0) & feedback_4_bee5;--(Xmax-Xmin)*
rand_4 ;
                    X1a<= X1(16);
                    X2a<= X1(17);
                    X3a<= X1(18);
                    X4a<= X1(19);
```

```
                if clk_1>6 then
                        control<="0001" ;
                else
                        control<="0000" ;
    end if;
---------------------------  Multiplicacion de señal mista por  matrix A  -------------------------
when Mult_mixA=>
        i_a0<=i_b;
        i_a1<=i_a0+1;

-------------- bee #1 ------------------

        M_1b1_bee1<= M_1(i_a0)*X1(0)+M_1(i_a1)*X1(2);
        M_1b2_bee1<= M_1(i_a0)*X1(1)+M_1(i_a1)*X1(3);

        M_2b1_bee1<= M_2(i_a0)*X1(0)+M_2(i_a1)*X1(2);
        M_2b2_bee1<= M_2(i_a0)*X1(1)+M_2(i_a1)*X1(3);

        M_1a_bee1(i_a0)<=M_1b1_bee1(26 downto 11);
        M_1a_bee1(i_a1)<=M_1b2_bee1(26 downto 11);

        M_2a_bee1(i_a0)<=M_2b1_bee1(26 downto 11);
        M_2a_bee1(i_a1)<=M_2b2_bee1(26 downto 11);

-------------- bee #2 ------------------
        M_1b1_bee2<= M_1(i_a0)*X1(4)+M_1(i_a1)*X1(6);
        M_1b2_bee2<= M_1(i_a0)*X1(5)+M_1(i_a1)*X1(7);

        M_2b1_bee2<= M_2(i_a0)*X1(4)+M_2(i_a1)*X1(6);
        M_2b2_bee2<= M_2(i_a0)*X1(5)+M_2(i_a1)*X1(7);

        M_1a_bee2(i_a0)<=M_1b1_bee2(26 downto 11);
        M_1a_bee2(i_a1)<=M_1b2_bee2(26 downto 11);

        M_2a_bee2(i_a0)<=M_2b1_bee2(26 downto 11);
        M_2a_bee2(i_a1)<=M_2b2_bee2(26 downto 11);

-------------- bee #3 ------------------
        M_1b1_bee3<= M_1(i_a0)*X1(8)+M_1(i_a1)*X1(10);
        M_1b2_bee3<= M_1(i_a0)*X1(9)+M_1(i_a1)*X1(11);

        M_2b1_bee3<= M_2(i_a0)*X1(8)+M_2(i_a1)*X1(10);
        M_2b2_bee3<= M_2(i_a0)*X1(9)+M_2(i_a1)*X1(11);
```

```
        M_1a_bee3(i_a0)<=M_1b1_bee3(26 downto 11);
        M_1a_bee3(i_a1)<=M_1b2_bee3(26 downto 11);

        M_2a_bee3(i_a0)<=M_2b1_bee3(26 downto 11);
        M_2a_bee3(i_a1)<=M_2b2_bee3(26 downto 11);

        ---------- bee #4 ----------
        M_1b1_bee4<= M_1(i_a0)*X1(12)+M_1(i_a1)*X1(14);
        M_1b2_bee4<= M_1(i_a0)*X1(13)+M_1(i_a1)*X1(15);

        M_2b1_bee4<= M_2(i_a0)*X1(12)+M_2(i_a1)*X1(14);
        M_2b2_bee4<= M_2(i_a0)*X1(13)+M_2(i_a1)*X1(15);

        M_1a_bee4(i_a0)<=M_1b1_bee4(26 downto 11);
        M_1a_bee4(i_a1)<=M_1b2_bee4(26 downto 11);

        M_2a_bee4(i_a0)<=M_2b1_bee4(26 downto 11);
        M_2a_bee4(i_a1)<=M_2b2_bee4(26 downto 11);

        ---------- bee #5 ----------
        M_1b1_bee5<= M_1(i_a0)*X1(16)+M_1(i_a1)*X1(18);
        M_1b2_bee5<= M_1(i_a0)*X1(17)+M_1(i_a1)*X1(19);

        M_2b1_bee5<= M_2(i_a0)*X1(16)+M_2(i_a1)*X1(18);
        M_2b2_bee5<= M_2(i_a0)*X1(17)+M_2(i_a1)*X1(19);

        M_1a_bee5(i_a0)<=M_1b1_bee5(26 downto 11);
        M_1a_bee5(i_a1)<=M_1b2_bee5(26 downto 11);

        M_2a_bee5(i_a0)<=M_2b1_bee5(26 downto 11);
        M_2a_bee5(i_a1)<=M_2b2_bee5(26 downto 11);
--
        if clk_2>6 then

                        if i_b>=98 then
                                control<="0010" ;
                                i_a<=0;

                        else
                                i_b<=i_b+2;
                                clk_2<=0;
                        end if;

         else
```

```
                        clk_2<=clk_2+1;

   end if;

-------------------------------------------------------------------------------------------------
when ECM=>

                                i_a0<=i_a;
                ----------  bee #1 ----------------------
                                M1_S1_bee1<=M_1a_bee1(i_a0)-S1(i_a0);
                                M2_S2_bee1<=M_2a_bee1(i_a0)-S2(i_a0);
                                M1_S1_b_bee1<=M1_S1_b_bee1+M1_S1_bee1;--    borrar
despues de usar
                                M2_S2_b_bee1<=M2_S2_b_bee1+M2_S2_bee1;  --  borrar
despues de usar

                                M_S_bee1<=(M1_S1_b_bee1)+ (M2_S2_b_bee1); -- esta es
tan grande como la cantidad de alimento

                ----------  bee #2 ----------------------
                                M1_S1_bee2<=M_1a_bee2(i_a0)-S1(i_a0);
                                M2_S2_bee2<=M_2a_bee2(i_a0)-S2(i_a0);
                                M1_S1_b_bee2<=M1_S1_b_bee2+M1_S1_bee2;--    borrar
despues de usar
                                M2_S2_b_bee2<=M2_S2_b_bee2+M2_S2_bee2;  --  borrar
despues de usar

                                M_S_bee2<=(M1_S1_b_bee2)+(M2_S2_b_bee2); -- esta es
tan grande como la cantidad de alimento
                ----------  bee #3 ----------------------
                                M1_S1_bee3<=M_1a_bee3(i_a0)-S1(i_a0);
                                M2_S2_bee3<=M_2a_bee3(i_a0)-S2(i_a0);
                                M1_S1_b_bee3<=M1_S1_b_bee3+M1_S1_bee3;--    borrar
despues de usar
                                M2_S2_b_bee3<=M2_S2_b_bee3+M2_S2_bee3;  --  borrar
despues de usar
                                M_S_bee3<=(M1_S1_b_bee3)+(M2_S2_b_bee3); -- esta es
tan grande como la cantidad de alimento
                ----------  bee #4 ----------------------
                                M1_S1_bee4<=M_1a_bee4(i_a0)-S1(i_a0);
                                M2_S2_bee4<=M_2a_bee4(i_a0)-S2(i_a0);
                                M1_S1_b_bee4<=M1_S1_b_bee4+M1_S1_bee4;--    borrar
despues de usar
```

```
                                M2_S2_b_bee4<=M2_S2_b_bee4+M2_S2_bee4;  --  borrar
despues de usar
                                M_S_bee4<=( M1_S1_b_bee4)+( M2_S2_b_bee4); -- esta
es tan grande como la cantidad de alimento
                        ----------  bee #5 ----------
                                M1_S1_bee5<=M_1a_bee5(i_a0)-S1(i_a0);
                                M2_S2_bee5<=M_2a_bee5(i_a0)-S2(i_a0);
                                M1_S1_b_bee5<=M1_S1_b_bee5+M1_S1_bee5;--    borrar
despues de usar
                                M2_S2_b_bee5<=M2_S2_b_bee5+M2_S2_bee5;  --  borrar
despues de usar

                                M_S_bee5<=(M1_S1_b_bee5)+(M2_S2_b_bee5); -- esta es
tan grande como la cantidad de alimento
                        if clk_2>9 then         -- reloj de processo

                                        if i_a>=98 then -- reloj de control de avance en las
señales
                                                control<="0011" ;
                                                i_b<=0;
                                M_S_bee1a<='0' & M_S_bee1; -- esta es tan grande como
la cantidad de alimento
                                M_S_bee2a<='0' & M_S_bee2; -- esta es tan grande como
la cantidad de alimento
                                M_S_bee3a<='0' & M_S_bee3; -- esta es tan grande como
la cantidad de alimento
                                M_S_bee4a<='0' & M_S_bee4; -- esta es tan grande como
la cantidad de alimento
                                M_S_bee5a<='0' & M_S_bee5; -- esta es tan grande como
la cantidad de alimento

                                                clk_1<=0;
                                        else
                                                i_a<=i_a+1;
                                                clk_2<=0;
                                        end if;

                        else
                                        clk_2<=clk_2+1;

                        end if;

when fitness=>
----- bee # 1 ----------
```

```
        if M_S_bee1a < M_S_bee2a then
                if M_S_bee1a < M_S_bee3a then
                        if M_S_bee1a < M_S_bee4a then
                                if M_S_bee1a < M_S_bee5a then
                                        food_source_best<=M_S_bee1a;
                                        Xbest1<=X1(0); Xbest2<=X1(1);
                                        Xbest3<=X1(2); Xbest4<=X1(3);
                                end if;
                        end if;
                end if;
        end if;

----- bee # 2 ----------------
        if M_S_bee2a < M_S_bee1a then
                if M_S_bee2a < M_S_bee3a then
                        if M_S_bee2a < M_S_bee4a then
                                if M_S_bee2a < M_S_bee5a then
                                        food_source_best<=M_S_bee2a;
                                        Xbest1<=X1(4); Xbest2<=X1(5);
                                        Xbest3<=X1(6); Xbest4<=X1(7);
                                end if;
                        end if;
                end if;
        end if;

----- bee # 3 ----------------
        if M_S_bee3a < M_S_bee1a then
                if M_S_bee3a < M_S_bee2a then
                        if M_S_bee3a < M_S_bee4a then
                                if M_S_bee3a < M_S_bee5a then
                                        food_source_best<=M_S_bee3a;
                                        Xbest1<=X1(8); Xbest2<=X1(9);
                                        Xbest3<=X1(10); Xbest4<=X1(11);
                                end if;
                        end if;
                end if;
        end if;
----- bee # 4 ----------------
        if M_S_bee4a < M_S_bee1a then
                if M_S_bee4a < M_S_bee2a then
                        if M_S_bee4a < M_S_bee3a then
                                if M_S_bee4a < M_S_bee5a then
                                        food_source_best<=M_S_bee4a;
                                        Xbest1<=X1(12); Xbest2<=X1(13);
```

```
                                        Xbest3<=X1(14); Xbest4<=X1(15);
                                end if;
                        end if;
                end if;
        end if;
----- bee # 5 ----------------
        if M_S_bee5a < M_S_bee1a then
                if M_S_bee5a < M_S_bee2a then
                        if M_S_bee5a < M_S_bee3a then
                                if M_S_bee5a < M_S_bee4a then
                                        food_source_best<=M_S_bee5a;
                                        Xbest1<=X1(16); Xbest2<=X1(17);
                                        Xbest3<=X1(18); Xbest4<=X1(19);
                                end if;
                        end if;
                end if;
        end if;
        if clk_1>9 then
                control<="0100" ;
                clk_2<=0;
        else
                clk_1<=clk_1+1;
        end if;

when Bee_employed_1=>
                        feedback_1_bee1 <= not(X1a(7) xor X1a(3));
                        feedback_2_bee1 <= not(X2a(3) xor X2a(4));
                        feedback_3_bee1 <= not(X3a(5) xor X3a(0));
                        feedback_4_bee1 <= not(X4a(2) xnor X4a(1));
                        feedback_4_bee2 <= not(X4a(2) xnor X4a(1));

                        X1a<= X1a(14 downto 0) & feedback_1_bee3;--(Xmax-Xmin)*
rand_1 ;
                        X2a<= X2a(14 downto 0) & feedback_2_bee3 ;--(Xmax-Xmin)*
rand_2 ;
                        X3a<= X3a(14 downto 0) & feedback_3_bee3;--(Xmax-Xmin)*
rand_3 ;
                        X4a<= X4a(14 downto 0) & feedback_4_bee3;--(Xmax-Xmin)*
rand_4 ;
                        X5a<= X5a(14 downto 0) & feedback_4_bee2;--(Xmax-Xmin)*
rand_4 ;

                                feedback_1_bee4 <= not(poss_1(1) xor poss_3(0));
                                feedback_2_bee4 <= not(poss_2(1) xor poss_4(1));
```

```
                                feedback_3_bee4 <= (poss_2(0) xor poss_3(0));
                                feedback_4_bee4 <= not(poss_4(0) xnor poss_1(1));


                                poss_1<=poss_1(0) & feedback_1_bee4;
                                poss_2<=poss_2(0) & feedback_2_bee4;
                                poss_3<=poss_3(0) & feedback_3_bee4;
                                poss_4<=poss_4(0) & feedback_4_bee4;
                                poss_5<=poss_5(0) & feedback_4_bee4;

        if clk_2>9 then
                control<="0101" ;
                clk_1<=0;
        else
                clk_2<=clk_2+1;
        end if;
when Bee_employed_2=>

---------- bee # 1 ----------
        if poss_1="00" then
                v_1_bee1<=X1(0)+X1a;
                v_2_bee1<=X1(1);
                v_3_bee1<=X1(2);
                v_4_bee1<=X1(3);
        elsif poss_1="01" then
                v_1_bee1<=X1(0);
                v_2_bee1<=X1(1)+X1a;
                v_3_bee1<=X1(2);
                v_4_bee1<=X1(3);
        elsif poss_1="10" then
                v_1_bee1<=X1(0);
                v_2_bee1<=X1(1);
                v_3_bee1<=X1(2)+X1a;
                v_4_bee1<=X1(3);
        else
                v_1_bee1<=X1(0);
                v_2_bee1<=X1(1);
                v_3_bee1<=X1(2);
                v_4_bee1<=X1(3)+X1a;
        end if;


---------- bee # 2 ----------
        if poss_2="00" then
```

```
                v_1_bee2<=X1(4)+X2a;
                v_2_bee2<=X1(5);
                v_3_bee2<=X1(6);
                v_4_bee2<=X1(7);
        elsif poss_2="01" then
                v_1_bee2<=X1(4);
                v_2_bee2<=X1(5)+X2a;
                v_3_bee2<=X1(6);
                v_4_bee2<=X1(7);
        elsif poss_2="10" then
                v_1_bee2<=X1(4);
                v_2_bee2<=X1(5);
                v_3_bee2<=X1(6)+X2a;
                v_4_bee2<=X1(7);
        else
                v_1_bee1<=X1(4);
                v_2_bee1<=X1(5);
                v_3_bee1<=X1(6);
                v_4_bee1<=X1(7)+X2a;
        end if;
---------- bee # 3 ----------
        if poss_3="00" then
                v_1_bee3<=X1(8)+X3a;
                v_2_bee3<=X1(9);
                v_3_bee3<=X1(10);
                v_4_bee3<=X1(11);
        elsif poss_3="01" then
                v_1_bee3<=X1(8);
                v_2_bee3<=X1(9)+X3a;
                v_3_bee3<=X1(10);
                v_4_bee3<=X1(11);
        elsif poss_3="10" then
                v_1_bee3<=X1(8);
                v_2_bee3<=X1(9);
                v_3_bee3<=X1(10)+X3a;
                v_4_bee3<=X1(11);
        else
                v_1_bee3<=X1(8);
                v_2_bee3<=X1(9);
                v_3_bee3<=X1(10);
                v_4_bee3<=X1(11)+X3a;
        end if;
---------- bee # 4 ----------
        if poss_4="00" then
```

```
                v_1_bee4<=X1(12)+X4a;
                v_2_bee4<=X1(13);
                v_3_bee4<=X1(14);
                v_4_bee4<=X1(15);
        elsif poss_4="01" then
                v_1_bee4<=X1(12);
                v_2_bee4<=X1(13)+X4a;
                v_3_bee4<=X1(14);
                v_4_bee4<=X1(15);
        elsif poss_4="10" then
                v_1_bee4<=X1(12);
                v_2_bee4<=X1(13);
                v_3_bee4<=X1(14)+X4a;
                v_4_bee4<=X1(15);
        else
                v_1_bee4<=X1(12);
                v_2_bee4<=X1(13);
                v_3_bee4<=X1(14);
                v_4_bee4<=X1(15)+X4a;
        end if;

---------- bee # 5 ----------
        if poss_5="00" then
                v_1_bee5<=X1(16)+X5a;
                v_2_bee5<=X1(17);
                v_3_bee5<=X1(18);
                v_4_bee5<=X1(19);
        elsif poss_5="01" then
                v_1_bee5<=X1(16);
                v_2_bee5<=X1(17)+X5a;
                v_3_bee5<=X1(18);
                v_4_bee5<=X1(19);
        elsif poss_5="10" then
                v_1_bee5<=X1(16);
                v_2_bee5<=X1(17);
                v_3_bee5<=X1(18)+X5a;
                v_4_bee5<=X1(19);
        else
                v_1_bee5<=X1(16);
                v_2_bee5<=X1(17);
                v_3_bee5<=X1(18);
                v_4_bee5<=X1(19)+X5a;
        end if;
--------------------------------------
```

```
        if clk_1>9 then
                control<="0110" ;
                clk_2<=0;
        else
                clk_1<=clk_1+1;
        end if;
when Bee_employed_3_mult=>
i_a0<=i_b;
        i_a1<=i_a0+1;

---------- bee #1 ----------

        M_1b1_bee1<= M_1(i_a0)*v_1_bee1+M_1(i_a1)*v_3_bee1;
        M_1b2_bee1<= M_1(i_a0)*v_2_bee1+M_1(i_a1)*v_4_bee1;

        M_2b1_bee1<= M_2(i_a0)*v_1_bee1+M_2(i_a1)*v_3_bee1;
        M_2b2_bee1<= M_2(i_a0)*v_2_bee1+M_2(i_a1)*v_4_bee1;

        M_1a_bee1(i_a0)<=M_1b1_bee1(26 downto 11);
        M_1a_bee1(i_a1)<=M_1b2_bee1(26 downto 11);

        M_2a_bee1(i_a0)<=M_2b1_bee1(26 downto 11);
        M_2a_bee1(i_a1)<=M_2b2_bee1(26 downto 11);

---------- bee #2 ----------

        M_1b1_bee2<= M_1(i_a0)*v_1_bee2+M_1(i_a1)*v_3_bee2;
        M_1b2_bee2<= M_1(i_a0)*v_2_bee2+M_1(i_a1)*v_4_bee2;

        M_2b1_bee2<= M_2(i_a0)*v_1_bee2+M_2(i_a1)*v_3_bee2;
        M_2b2_bee2<= M_2(i_a0)*v_2_bee2+M_2(i_a1)*v_4_bee2;

        M_1a_bee2(i_a0)<=M_1b1_bee2(26 downto 11);
        M_1a_bee2(i_a1)<=M_1b2_bee2(26 downto 11);

        M_2a_bee2(i_a0)<=M_2b1_bee2(26 downto 11);
        M_2a_bee2(i_a1)<=M_2b2_bee2(26 downto 11);

---------- bee #3 ----------

        M_1b1_bee3<= M_1(i_a0)*v_1_bee3+M_1(i_a1)*v_3_bee3;
        M_1b2_bee3<= M_1(i_a0)*v_2_bee3+M_1(i_a1)*v_4_bee3;

        M_2b1_bee3<= M_2(i_a0)*v_1_bee3+M_2(i_a1)*v_3_bee3;
```

```
        M_2b2_bee3<= M_2(i_a0)*v_2_bee3+M_2(i_a1)*v_4_bee3;

        M_1a_bee3(i_a0)<=M_1b1_bee3(26 downto 11);
        M_1a_bee3(i_a1)<=M_1b2_bee3(26 downto 11);

        M_2a_bee3(i_a0)<=M_2b1_bee3(26 downto 11);
        M_2a_bee3(i_a1)<=M_2b2_bee3(26 downto 11);
---------- bee #4 ----------

        M_1b1_bee4<= M_1(i_a0)*v_1_bee4+M_1(i_a1)*v_3_bee4;
        M_1b2_bee4<= M_1(i_a0)*v_2_bee4+M_1(i_a1)*v_4_bee4;

        M_2b1_bee4<= M_2(i_a0)*v_1_bee4+M_2(i_a1)*v_3_bee4;
        M_2b2_bee4<= M_2(i_a0)*v_2_bee4+M_2(i_a1)*v_4_bee4;

        M_1a_bee4(i_a0)<=M_1b1_bee4(26 downto 11);
        M_1a_bee4(i_a1)<=M_1b2_bee4(26 downto 11);

        M_2a_bee4(i_a0)<=M_2b1_bee4(26 downto 11);
        M_2a_bee4(i_a1)<=M_2b2_bee4(26 downto 11);
---------- bee #5 ----------

        M_1b1_bee5<= M_1(i_a0)*v_1_bee5+M_1(i_a1)*v_3_bee5;
        M_1b2_bee5<= M_1(i_a0)*v_2_bee5+M_1(i_a1)*v_4_bee5;

        M_2b1_bee5<= M_2(i_a0)*v_1_bee5+M_2(i_a1)*v_3_bee5;
        M_2b2_bee5<= M_2(i_a0)*v_2_bee5+M_2(i_a1)*v_4_bee5;

        M_1a_bee5(i_a0)<=M_1b1_bee5(26 downto 11);
        M_1a_bee5(i_a1)<=M_1b2_bee5(26 downto 11);

        M_2a_bee5(i_a0)<=M_2b1_bee5(26 downto 11);
        M_2a_bee5(i_a1)<=M_2b2_bee5(26 downto 11);

                M1_S1_b_bee1<=X"0000";
                M2_S2_b_bee1<=X"0000";
                M1_S1_b_bee2<=X"0000";
                M2_S2_b_bee2<=X"0000";
                M1_S1_b_bee3<=X"0000";
                M2_S2_b_bee3<=X"0000";
                M1_S1_b_bee4<=X"0000";
                M2_S2_b_bee4<=X"0000";
                M1_S1_b_bee5<=X"0000";
                M2_S2_b_bee5<=X"0000";
```

```
if clk_2>6 then
                        if i_b>=98 then
                                control<="0111" ;
                                i_a<=0;

                        else
                                i_b<=i_b+2;
                                clk_2<=0;
                        end if;
          else
                        clk_2<=clk_2+1;

   end if;
when Bee_employed_ECM=>

                                i_a0<=i_a;
                ----------  bee #1 ----------------------
                                M1_S1_bee1<=M_1a_bee1(i_a0)-S1(i_a0);
                                M2_S2_bee1<=M_2a_bee1(i_a0)-S2(i_a0);
                                M1_S1_b_bee1<=M1_S1_b_bee1+M1_S1_bee1;--    borrar
despues de usar
                                M2_S2_b_bee1<=M2_S2_b_bee1+M2_S2_bee1;  --  borrar
despues de usar


                ----------  bee #2 ----------------------
                                M1_S1_bee2<=M_1a_bee2(i_a0)-S1(i_a0);
                                M2_S2_bee2<=M_2a_bee2(i_a0)-S2(i_a0);

                                M1_S1_b_bee2<=M1_S1_b_bee2+M1_S1_bee2;--    borrar
despues de usar
                                M2_S2_b_bee2<=M2_S2_b_bee2+M2_S2_bee2;  --  borrar
despues de usar

                ----------  bee #3 ----------------------
                                M1_S1_bee3<=M_1a_bee3(i_a0)-S1(i_a0);
                                M2_S2_bee3<=M_2a_bee3(i_a0)-S2(i_a0);
                                M1_S1_b_bee3<=M1_S1_b_bee3+M1_S1_bee3;--    borrar
despues de usar
                                M2_S2_b_bee3<=M2_S2_b_bee3+M2_S2_bee3;  --  borrar
despues de usar
                                ----------  bee #4 ----------------------
                                M1_S1_bee4<=M_1a_bee4(i_a0)-S1(i_a0);
```

```
                                M2_S2_bee4<=M_2a_bee4(i_a0)-S2(i_a0);
                                M1_S1_b_bee4<=M1_S1_b_bee4+M1_S1_bee4;--    borrar
despues de usar
                                M2_S2_b_bee4<=M2_S2_b_bee4+M2_S2_bee4;  --  borrar
despues de usar

                                ---------------  bee #5 -----------------------
                                M1_S1_bee5<=M_1a_bee5(i_a0)-S1(i_a0);
                                M2_S2_bee5<=M_2a_bee5(i_a0)-S2(i_a0);
                                M1_S1_b_bee5<=M1_S1_b_bee5+M1_S1_bee5;--    borrar
despues de usar
                                M2_S2_b_bee5<=M2_S2_b_bee5+M2_S2_bee5;  --  borrar
despues de usar
                                M_S_bee1<=M1_S1_b_bee1+M2_S2_b_bee1; -- esta es tan
grande como la cantidad de alimento
                                M_S_bee2<=M1_S1_b_bee2+M2_S2_b_bee2; -- esta es tan
grande como la cantidad de alimento
                                M_S_bee3<=M1_S1_b_bee3+M2_S2_b_bee3; -- esta es tan
grande como la cantidad de alimento
                                M_S_bee4<=M1_S1_b_bee4+M2_S2_b_bee4; -- esta es tan
grande como la cantidad de alimento
                                M_S_bee5<=M1_S1_b_bee5+M2_S2_b_bee5; -- esta es tan
grande como la cantidad de alimento
                        if clk_2>9 then         -- reloj de processo
                                if i_a>=98 then -- reloj de control de avance en las
señales
                                        control<="1000" ;
                                        i_b<=0;

                                M_S_bee1b<='0' & M_S_bee1; -- esta es tan grande como
la cantidad de alimento
                                M_S_bee2b<='0' & M_S_bee2; -- esta es tan grande como
la cantidad de alimento
                                M_S_bee3b<='0' & M_S_bee3; -- esta es tan grande como
la cantidad de alimento
                                M_S_bee4b<='0' & M_S_bee4; -- esta es tan grande como
la cantidad de alimento
                                M_S_bee5b<='0' & M_S_bee5; -- esta es tan grande como
la cantidad de alimento
                                        clk_1<=0;
                                else
                                        i_a<=i_a+1;
                                        clk_2<=0;
                                end if;
```

```
                        else
                                clk_2<=clk_2+1;

                        end if;


when Bee_employed_5=>

---------- bee # 1 ----------
        if M_S_bee1b < M_S_bee1a then
                        M_S_bee1a<=M_S_bee1b;
                        X1(0)<=v_1_bee1;
                        X1(1)<=v_2_bee1;
                        X1(2)<=v_3_bee1;
                        X1(3)<=v_4_bee1;
                                trip_bee1<=0;
                else
                                trip_bee1<=trip_bee1+1;
        end if;

---------- bee # 2 ----------
        if M_S_bee2b < M_S_bee2a then
                        M_S_bee2a<=M_S_bee2b;
                        X1(4)<=v_1_bee2;
                        X1(5)<=v_2_bee2;
                        X1(6)<=v_3_bee2;
                        X1(7)<=v_4_bee2;
                                trip_bee2<=0;
                else
                                trip_bee2<=trip_bee2+1;

        end if;
---------- bee # 3 ----------
        if M_S_bee3b < M_S_bee3a then
                        M_S_bee3a<=M_S_bee3b;
                        X1(8)<=v_1_bee3;
                        X1(9)<=v_2_bee3;
                        X1(10)<=v_3_bee3;
                        X1(11)<=v_4_bee3;
                                trip_bee3<=0;
                else
                                trip_bee3<=trip_bee3+1;
        end if;
```

```
---------- bee # 4 ----------
        if M_S_bee4b < M_S_bee4a then
                        M_S_bee4a<=M_S_bee4b;
                        X1(12)<=v_1_bee4;
                        X1(13)<=v_2_bee4;
                        X1(14)<=v_3_bee4;
                        X1(15)<=v_4_bee4;
                                trip_bee4<=0;
                 else
                                trip_bee4<=trip_bee4+1;
        end if;
---------- bee # 5 ----------
        if M_S_bee5b < M_S_bee5a then
                        M_S_bee5a<=M_S_bee5b;
                        X1(16)<=v_1_bee5;
                        X1(17)<=v_2_bee5;
                        X1(18)<=v_3_bee5;
                        X1(19)<=v_4_bee5;
                                trip_bee5<=0;
                 else
                                trip_bee5<=trip_bee5+1;
        end if;



        if clk_1>1 then
                control<="1001" ;
                clk_2<=0;
        else
                clk_1<=clk_1+1;
        end if;

when Bee_employed_6=> -- busqueda del minimo global employed

        if  M_S_bee1a < food_source_best then
                food_source_best<=M_S_bee1a;
                Xbest1<=X1(0); Xbest2<=X1(1);
                Xbest3<=X1(2); Xbest4<=X1(3);
        elsif  M_S_bee2a < food_source_best then
                food_source_best<=M_S_bee2a;
                Xbest1<=X1(4); Xbest2<=X1(5);
                Xbest3<=X1(6); Xbest4<=X1(7);
        elsif  M_S_bee3a < food_source_best then
                food_source_best<=M_S_bee3a;
```

```
                Xbest1<=X1(8); Xbest2<=X1(9);
                Xbest3<=X1(10); Xbest4<=X1(11);
        elsif  M_S_bee4a < food_source_best then
                food_source_best<=M_S_bee4a;
                Xbest1<=X1(12); Xbest2<=X1(13);
                Xbest3<=X1(14); Xbest4<=X1(15);
        elsif  M_S_bee5a < food_source_best then
                food_source_best<=M_S_bee5a;
                Xbest1<=X1(16); Xbest2<=X1(17);
                Xbest3<=X1(18); Xbest4<=X1(19);
        end if;
        if clk_2>9 then
                control<="1010" ;
                clk_1<=0;
        else
                clk_2<=clk_2+1;
        end if;

when bee_onlooker_1=>

                        feedback_1_bee1 <= not(X1a(7) xor X1a(3));
                        feedback_2_bee1 <= not(X2a(3) xor X2a(4));
                        feedback_3_bee1 <= not(X3a(5) xor X3a(0));
                        feedback_4_bee1 <= not(X4a(2) xnor X4a(1));
                        feedback_4_bee2 <= not(X4a(2) xnor X4a(1));

                        X1a<= X1a(14 downto 0) & feedback_1_bee3;--(Xmax-Xmin)*
rand_1 ;
                        X2a<= X2a(14 downto 0) & feedback_2_bee3 ;--(Xmax-Xmin)*
rand_2 ;
                        X3a<= X3a(14 downto 0) & feedback_3_bee3;--(Xmax-Xmin)*
rand_3 ;
                        X4a<= X4a(14 downto 0) & feedback_4_bee3;--(Xmax-Xmin)*
rand_4 ;
                        X5a<= X5a(14 downto 0) & feedback_4_bee2;--(Xmax-Xmin)*
rand_4 ;

                                feedback_1_bee4 <= not(poss_1(1) xor poss_3(0));
                                feedback_2_bee4 <= not(poss_2(1) xor poss_4(1));
                                feedback_3_bee4 <= (poss_2(0) xor poss_3(0));
                                feedback_4_bee4 <= not(poss_4(0) xnor poss_1(1));
                                feedback_1_bee2 <= (poss_2(1) xnor poss_3(0));

                                poss_1<=poss_1(0) & feedback_1_bee4;
```

```
                                poss_2<=poss_2(0) & feedback_2_bee4;
                                poss_3<=poss_3(0) & feedback_3_bee4;
                                poss_4<=poss_4(0) & feedback_4_bee4;
                                poss_5<=poss_5(0) & feedback_1_bee2;

        if clk_1>3 then
                control<="1011";
                clk_2<=0;
        else
                clk_1<=clk_1+1;
        end if;
when bee_onlooker_2 =>
---------- bee # 1 ----------
if poss_5="00" then
        if poss_1="00" then
                v_1_bee1<=X1(0)+X1a;
                v_2_bee1<=X1(1);
                v_3_bee1<=X1(2);
                v_4_bee1<=X1(3);
        elsif poss_1="01" then
                v_1_bee1<=X1(0);
                v_2_bee1<=X1(1)+X1a;
                v_3_bee1<=X1(2);
                v_4_bee1<=X1(3);
        elsif poss_1="10" then
                v_1_bee1<=X1(0);
                v_2_bee1<=X1(1);
                v_3_bee1<=X1(2)+X1a;
                v_4_bee1<=X1(3);
        else
                v_1_bee1<=X1(0);
                v_2_bee1<=X1(1);
                v_3_bee1<=X1(2);
                v_4_bee1<=X1(3)+X1a;
        end if;
---------- bee # 2 ----------
elsif poss_5="01" then
        if poss_2="00" then
                v_1_bee2<=X1(4)+X2a;
                v_2_bee2<=X1(5);
                v_3_bee2<=X1(6);
                v_4_bee2<=X1(7);
        elsif poss_2="01" then
                v_1_bee2<=X1(4);
```

```
                v_2_bee2<=X1(5)+X2a;
                v_3_bee2<=X1(6);
                v_4_bee2<=X1(7);
        elsif poss_2="10" then
                v_1_bee2<=X1(4);
                v_2_bee2<=X1(5);
                v_3_bee2<=X1(6)+X2a;
                v_4_bee2<=X1(7);
        else
                v_1_bee1<=X1(4);
                v_2_bee1<=X1(5);
                v_3_bee1<=X1(6);
                v_4_bee1<=X1(7)+X2a;
        end if;
---------- bee # 3 ----------
elsif poss_5="10" then
        if poss_3="00" then
                v_1_bee3<=X1(8)+X3a;
                v_2_bee3<=X1(9);
                v_3_bee3<=X1(10);
                v_4_bee3<=X1(11);
        elsif poss_3="01" then
                v_1_bee3<=X1(8);
                v_2_bee3<=X1(9)+X3a;
                v_3_bee3<=X1(10);
                v_4_bee3<=X1(11);
        elsif poss_3="10" then
                v_1_bee3<=X1(8);
                v_2_bee3<=X1(9);
                v_3_bee3<=X1(10)+X3a;
                v_4_bee3<=X1(11);
        else
                v_1_bee3<=X1(8);
                v_2_bee3<=X1(9);
                v_3_bee3<=X1(10);
                v_4_bee3<=X1(11)+X3a;
        end if;
---------- bee # 4 ----------
elsif poss_5="11" then
        if poss_4="00" then
                v_1_bee4<=X1(12)+X4a;
                v_2_bee4<=X1(13);
                v_3_bee4<=X1(14);
                v_4_bee4<=X1(15);
```

```
        elsif poss_4="01" then
                v_1_bee4<=X1(12);
                v_2_bee4<=X1(13)+X4a;
                v_3_bee4<=X1(14);
                v_4_bee4<=X1(15);
        elsif poss_4="10" then
                v_1_bee4<=X1(12);
                v_2_bee4<=X1(13);
                v_3_bee4<=X1(14)+X4a;
                v_4_bee4<=X1(15);
        else
                v_1_bee4<=X1(12);
                v_2_bee4<=X1(13);
                v_3_bee4<=X1(14);
                v_4_bee4<=X1(15)+X4a;
        end if;

---------- bee # 5 ----------

        if poss_5="00" then
                v_1_bee5<=X1(16)+X5a;
                v_2_bee5<=X1(17);
                v_3_bee5<=X1(18);
                v_4_bee5<=X1(19);
        elsif poss_5="01" then
                v_1_bee5<=X1(16);
                v_2_bee5<=X1(17)+X5a;
                v_3_bee5<=X1(18);
                v_4_bee5<=X1(19);
        elsif poss_5="10" then
                v_1_bee5<=X1(16);
                v_2_bee5<=X1(17);
                v_3_bee5<=X1(18)+X5a;
                v_4_bee5<=X1(19);
        else
                v_1_bee5<=X1(16);
                v_2_bee5<=X1(17);
                v_3_bee5<=X1(18);
                v_4_bee5<=X1(19)+X5a;
        end if;
end if; --control random de cual se vuelve a explorar
        if clk_1>9 then
                control<="1100";
                clk_2<=0;
```

```
        else
                clk_1<=clk_1+1;
        end if;
when   bee_onlooker_3=>
i_a0<=i_b;
        i_a1<=i_a0+1;
-------------- bee #1 ------------------

        M_1b1_bee1<= M_1(i_a0)*v_1_bee1+M_1(i_a1)*v_3_bee1;
        M_1b2_bee1<= M_1(i_a0)*v_2_bee1+M_1(i_a1)*v_4_bee1;

        M_2b1_bee1<= M_2(i_a0)*v_1_bee1+M_2(i_a1)*v_3_bee1;
        M_2b2_bee1<= M_2(i_a0)*v_2_bee1+M_2(i_a1)*v_4_bee1;

        M_1a_bee1(i_a0)<=M_1b1_bee1(26 downto 11);
        M_1a_bee1(i_a1)<=M_1b2_bee1(26 downto 11);

        M_2a_bee1(i_a0)<=M_2b1_bee1(26 downto 11);
        M_2a_bee1(i_a1)<=M_2b2_bee1(26 downto 11);

-------------- bee #2 ------------------

        M_1b1_bee2<= M_1(i_a0)*v_1_bee2+M_1(i_a1)*v_3_bee2;
        M_1b2_bee2<= M_1(i_a0)*v_2_bee2+M_1(i_a1)*v_4_bee2;

        M_2b1_bee2<= M_2(i_a0)*v_1_bee2+M_2(i_a1)*v_3_bee2;
        M_2b2_bee2<= M_2(i_a0)*v_2_bee2+M_2(i_a1)*v_4_bee2;

        M_1a_bee2(i_a0)<=M_1b1_bee2(26 downto 11);
        M_1a_bee2(i_a1)<=M_1b2_bee2(26 downto 11);

        M_2a_bee2(i_a0)<=M_2b1_bee2(26 downto 11);
        M_2a_bee2(i_a1)<=M_2b2_bee2(26 downto 11);

-------------- bee #3 ------------------

        M_1b1_bee3<= M_1(i_a0)*v_1_bee3+M_1(i_a1)*v_3_bee3;
        M_1b2_bee3<= M_1(i_a0)*v_2_bee3+M_1(i_a1)*v_4_bee3;

        M_2b1_bee3<= M_2(i_a0)*v_1_bee3+M_2(i_a1)*v_3_bee3;
        M_2b2_bee3<= M_2(i_a0)*v_2_bee3+M_2(i_a1)*v_4_bee3;

        M_1a_bee3(i_a0)<=M_1b1_bee3(26 downto 11);
        M_1a_bee3(i_a1)<=M_1b2_bee3(26 downto 11);
```

```
        M_2a_bee3(i_a0)<=M_2b1_bee3(26 downto 11);
        M_2a_bee3(i_a1)<=M_2b2_bee3(26 downto 11);
---------- bee #4 ----------

        M_1b1_bee4<= M_1(i_a0)*v_1_bee4+M_1(i_a1)*v_3_bee4;
        M_1b2_bee4<= M_1(i_a0)*v_2_bee4+M_1(i_a1)*v_4_bee4;

        M_2b1_bee4<= M_2(i_a0)*v_1_bee4+M_2(i_a1)*v_3_bee4;
        M_2b2_bee4<= M_2(i_a0)*v_2_bee4+M_2(i_a1)*v_4_bee4;

        M_1a_bee4(i_a0)<=M_1b1_bee4(26 downto 11);
        M_1a_bee4(i_a1)<=M_1b2_bee4(26 downto 11);

        M_2a_bee4(i_a0)<=M_2b1_bee4(26 downto 11);
        M_2a_bee4(i_a1)<=M_2b2_bee4(26 downto 11);
---------- bee #5 ----------

        M_1b1_bee5<= M_1(i_a0)*v_1_bee5+M_1(i_a1)*v_3_bee5;
        M_1b2_bee5<= M_1(i_a0)*v_2_bee5+M_1(i_a1)*v_4_bee5;

        M_2b1_bee5<= M_2(i_a0)*v_1_bee5+M_2(i_a1)*v_3_bee5;
        M_2b2_bee5<= M_2(i_a0)*v_2_bee5+M_2(i_a1)*v_4_bee5;

        M_1a_bee5(i_a0)<=M_1b1_bee5(26 downto 11);
        M_1a_bee5(i_a1)<=M_1b2_bee5(26 downto 11);

        M_2a_bee5(i_a0)<=M_2b1_bee5(26 downto 11);
        M_2a_bee5(i_a1)<=M_2b2_bee5(26 downto 11);

                M1_S1_b_bee1<=X"0000";
                M2_S2_b_bee1<=X"0000";
                M1_S1_b_bee2<=X"0000";
                M2_S2_b_bee2<=X"0000";
                M1_S1_b_bee3<=X"0000";
                M2_S2_b_bee3<=X"0000";
                M1_S1_b_bee4<=X"0000";
                M2_S2_b_bee4<=X"0000";
                M1_S1_b_bee5<=X"0000";
                M2_S2_b_bee5<=X"0000";
if clk_2>6 then
                        if i_b>=98 then
                                control<="1101" ;
                                i_a<=0;
```

```
                        else
                                i_b<=i_b+2;
                                clk_2<=0;
                        end if;

          else
                        clk_2<=clk_2+1;

   end if;
when bee_onlooker_4 =>
i_a0<=i_a;
                ----------  bee #1 ----------
                                M1_S1_bee1<=M_1a_bee1(i_a0)-S1(i_a0);
                                M2_S2_bee1<=M_2a_bee1(i_a0)-S2(i_a0);
                                M1_S1_b_bee1<=M1_S1_b_bee1+M1_S1_bee1;--    borrar
despues de usar
                                M2_S2_b_bee1<=M2_S2_b_bee1+M2_S2_bee1;  --  borrar
despues de usar


                ----------  bee #2 ----------
                                M1_S1_bee2<=M_1a_bee2(i_a0)-S1(i_a0);
                                M2_S2_bee2<=M_2a_bee2(i_a0)-S2(i_a0);
                                M1_S1_b_bee2<=M1_S1_b_bee2+M1_S1_bee2;--    borrar
despues de usar
                                M2_S2_b_bee2<=M2_S2_b_bee2+M2_S2_bee2;  --  borrar
despues de usar

                ----------  bee #3 ----------
                                M1_S1_bee3<=M_1a_bee3(i_a0)-S1(i_a0);
                                M2_S2_bee3<=M_2a_bee3(i_a0)-S2(i_a0);
                                M1_S1_b_bee3<=M1_S1_b_bee3+M1_S1_bee3;--    borrar
despues de usar
                                M2_S2_b_bee3<=M2_S2_b_bee3+M2_S2_bee3;  --  borrar
despues de usar

                                ----------  bee #4 ----------
                                M1_S1_bee4<=M_1a_bee4(i_a0)-S1(i_a0);
                                M2_S2_bee4<=M_2a_bee4(i_a0)-S2(i_a0);
                                M1_S1_b_bee4<=M1_S1_b_bee4+M1_S1_bee4;--    borrar
despues de usar
                                M2_S2_b_bee4<=M2_S2_b_bee4+M2_S2_bee4;  --  borrar
despues de usar
```

```
                        ---------- bee #5 ----------
                        M1_S1_bee5<=M_1a_bee5(i_a0)-S1(i_a0);
                        M2_S2_bee5<=M_2a_bee5(i_a0)-S2(i_a0);
                        M1_S1_b_bee5<=M1_S1_b_bee5+M1_S1_bee5;--  borrar
despues de usar
                        M2_S2_b_bee5<=M2_S2_b_bee5+M2_S2_bee5;  --  borrar
despues de usar

                        M_S_bee1<=M1_S1_b_bee1+M2_S2_b_bee1; -- esta es tan
grande como la cantidad de alimento
                        M_S_bee2<=M1_S1_b_bee2+M2_S2_b_bee2; -- esta es tan
grande como la cantidad de alimento
                        M_S_bee3<=M1_S1_b_bee3+M2_S2_b_bee3; -- esta es tan
grande como la cantidad de alimento
                        M_S_bee4<=M1_S1_b_bee4+M2_S2_b_bee4; -- esta es tan
grande como la cantidad de alimento
                        M_S_bee5<=M1_S1_b_bee5+M2_S2_b_bee5; -- esta es tan
grande como la cantidad de alimento
                if clk_2>9 then         -- reloj de processo

                        if i_a>=98 then -- reloj de control de avance en las
señales
                                control<="1110" ;
                                i_b<=0;
                        M_S_bee1b<='0' & M_S_bee1; -- esta es tan grande como
la cantidad de alimento
                        M_S_bee2b<='0' & M_S_bee2; -- esta es tan grande como
la cantidad de alimento
                        M_S_bee3b<='0' & M_S_bee3; -- esta es tan grande como
la cantidad de alimento
                        M_S_bee4b<='0' & M_S_bee4; -- esta es tan grande como
la cantidad de alimento
                        M_S_bee5b<='0' & M_S_bee5; -- esta es tan grande como
la cantidad de alimento
                                clk_1<=0;
                        else
                                i_a<=i_a+1;
                                clk_2<=0;
                        end if;
                else
                        clk_2<=clk_2+1;
                end if;
when bee_onlooker_5=>
```

```
---------- bee # 1 ----------
        if M_S_bee1b < M_S_bee1a then
                        M_S_bee1a<=M_S_bee1b;
                        X1(0)<=v_1_bee1;
                        X1(1)<=v_2_bee1;
                        X1(2)<=v_3_bee1;
                        X1(3)<=v_4_bee1;
                                trip_bee1<=0;
                else
                                trip_bee1<=trip_bee1+1;
        end if;

---------- bee # 2 ----------
        if M_S_bee2b < M_S_bee2a then
                        M_S_bee2a<=M_S_bee2b;
                        X1(4)<=v_1_bee2;
                        X1(5)<=v_2_bee2;
                        X1(6)<=v_3_bee2;
                        X1(7)<=v_4_bee2;
                                trip_bee2<=0;
                else
                                trip_bee2<=trip_bee2+1;
        end if;
---------- bee # 3 ----------
        if M_S_bee3b < M_S_bee3a then
                        M_S_bee3a<=M_S_bee3b;
                        X1(8)<=v_1_bee3;
                        X1(9)<=v_2_bee3;
                        X1(10)<=v_3_bee3;
                        X1(11)<=v_4_bee3;
                                trip_bee3<=0;
                else
                                trip_bee3<=trip_bee3+1;
        end if;
---------- bee # 4 ----------
        if M_S_bee4b < M_S_bee4a then
                        M_S_bee4a<=M_S_bee4b;
                        X1(12)<=v_1_bee4;
                        X1(13)<=v_2_bee4;
                        X1(14)<=v_3_bee4;
                        X1(15)<=v_4_bee4;
                                trip_bee4<=0;
                else
                                trip_bee4<=trip_bee4+1;
```

```
        end if;
---------- bee # 5 ----------
        if M_S_bee5b < M_S_bee5a then
                        M_S_bee5a<=M_S_bee5b;
                        X1(16)<=v_1_bee5;
                        X1(17)<=v_2_bee5;
                        X1(18)<=v_3_bee5;
                        X1(19)<=v_4_bee5;
                                trip_bee5<=0;
                 else
                                trip_bee5<=trip_bee5+1;
        end if;



        if clk_1>1 then
                control<="1111" ;
                clk_2<=0;
        else
                clk_1<=clk_1+1;
        end if;

when bee_onlooker_6=> -- busqueda del minimo global employed

        if  M_S_bee1a < food_source_best then
                food_source_best<=M_S_bee1a;
                Xbest1<=X1(0); Xbest2<=X1(1);
                Xbest3<=X1(2); Xbest4<=X1(3);
        elsif  M_S_bee2a < food_source_best then
                food_source_best<=M_S_bee2a;
                Xbest1<=X1(4); Xbest2<=X1(5);
                Xbest3<=X1(6); Xbest4<=X1(7);
        elsif  M_S_bee3a < food_source_best then
                food_source_best<=M_S_bee3a;
                Xbest1<=X1(8); Xbest2<=X1(9);
                Xbest3<=X1(10); Xbest4<=X1(11);
        elsif  M_S_bee4a < food_source_best then
                food_source_best<=M_S_bee4a;
                Xbest1<=X1(12); Xbest2<=X1(13);
                Xbest3<=X1(14); Xbest4<=X1(15);
        elsif  M_S_bee5a < food_source_best then
                food_source_best<=M_S_bee5a;
                Xbest1<=X1(16); Xbest2<=X1(17);
                Xbest3<=X1(18); Xbest4<=X1(19);
```

```
        end if;
        if clk_2>9 then
                control<="0111" ;
                clk_1<=0;
        else
                clk_2<=clk_2+1;
        end if;

when bee_explorer =>
                        feedback_1_bee1 <= not(X1(1)(7) xor X1(2)(3));
                        feedback_2_bee1 <= not(X1(2)(3) xor X1(3)(4));
                        feedback_3_bee1 <= not(X1(2)(5) xor X1(0)(0));
                        feedback_4_bee1 <= not(X1(1)(2) xnor X1(6)(1));

                        feedback_1_bee2 <= not(X1(4)(7) xor X1(6)(3));
                        feedback_2_bee2 <= not(X1(5)(3) xor X1(5)(4));
                        feedback_3_bee2 <= not(X1(5)(5) xor X1(7)(0));
                        feedback_4_bee2 <= not(X1(4)(2) xnor X1(4)(1));

                        feedback_1_bee3 <= not(X1(4)(7) xor X1(7)(3));
                        feedback_2_bee3 <= not(X1(4)(3) xor X1(5)(4));
                        feedback_3_bee3 <= not(X1(5)(5) xor X1(6)(0));
                        feedback_4_bee3 <= not(X1(6)(2) xnor X1(6)(1));

                        feedback_1_bee4 <= not(X1(8)(8) xor X1(10)(3));
                        feedback_2_bee4 <= not(X1(7)(9) xor X1(11)(4));
                        feedback_3_bee4 <= (X1(9)(2) xor X1(7)(0));
                        feedback_4_bee4 <= not(X1(8)(2) xnor X1(11)(1));

                        feedback_1_bee5 <= not(X1(12)(8) xor X1(13)(3));
                        feedback_2_bee5 <= not(X1(14)(9) xor X1(14)(4));
                        feedback_3_bee5 <= (X1(9)(15) xor X1(9)(0));
                        feedback_4_bee5 <= not(X1(12)(2) xnor X1(11)(1));
                        ------------  bee # 1 -----------
                        if trip_bee1>150 then
                                X1(0)<= X1(0)(14 downto 0) & feedback_1_bee1;--(Xmax-
Xmin)* rand_1 ;
                                X1(1)<= X1(1)(14 downto 0) & feedback_2_bee1 ;--(Xmax-
Xmin)* rand_2 ;
                                X1(2)<= X1(2)(14 downto 0) & feedback_3_bee1;--(Xmax-
Xmin)* rand_3 ;
                                X1(3)<= X1(3)(14 downto 0) & feedback_4_bee1;--(Xmax-
Xmin)* rand_4 ;
                        end if;
```

```
            ----------  bee # 2 -----------
                    if trip_bee2>150 then
                            X1(4)<= X1(4)(14 downto 0) & feedback_1_bee2;--(Xmax-
Xmin)* rand_1 ;
                            X1(5)<= X1(5)(14 downto 0) & feedback_2_bee2 ;--(Xmax-
Xmin)* rand_2 ;
                            X1(6)<= X1(6)(14 downto 0) & feedback_3_bee2;--(Xmax-
Xmin)* rand_3 ;
                            X1(7)<= X1(7)(14 downto 0) & feedback_4_bee2;--(Xmax-
Xmin)* rand_4 ;
                    end if;


            ----------  bee # 3 -----------
                    if trip_bee3>150 then
                            X1(8)<= X1(15)(14 downto 0) & feedback_1_bee3;--(Xmax-
Xmin)* rand_1 ;
                            X1(9)<= X1(17)(14 downto 0) & feedback_2_bee3 ;--(Xmax-
Xmin)* rand_2 ;
                            X1(10)<= X1(18)(14 downto 0) & feedback_3_bee3;--(Xmax-
Xmin)* rand_3 ;
                            X1(11)<= X1(19)(14 downto 0) & feedback_4_bee3;--(Xmax-
Xmin)* rand_4 ;
                    end if;

            ----------  bee # 4 -----------
                    if trip_bee4>150 then
                            X1(12)<= X1(8)(14 downto 0) & feedback_1_bee4;--(Xmax-
Xmin)* rand_1 ;
                            X1(13)<= X1(9)(14 downto 0) & feedback_2_bee4 ;--(Xmax-
Xmin)* rand_2 ;
                            X1(14)<= X1(10)(14 downto 0) & feedback_3_bee4;--(Xmax-
Xmin)* rand_3 ;
                            X1(15)<= X1(11)(14 downto 0) & feedback_4_bee4;--(Xmax-
Xmin)* rand_4 ;
                    end if;

            ----------  bee # 4 -----------
                    if trip_bee5>150 then
                            X1(16)<= X1(16)(14 downto 0) & feedback_1_bee5;--(Xmax-
Xmin)* rand_1 ;
                            X1(17)<= X1(17)(14 downto 0) & feedback_2_bee5 ;--
(Xmax-Xmin)* rand_2 ;
```

```
                                X1(18)<= X1(18)(14 downto 0) & feedback_3_bee5;--(Xmax-
Xmin)* rand_3 ;
                                X1(19)<= X1(8)(14 downto 0) & feedback_4_bee5;--(Xmax-
Xmin)* rand_4 ;
                        end if;


        if clk_1>9 then
                control<="0011" ;
                clk_2<=0;
        else
                clk_1<=clk_1+1;
        end if;

when fin=>
                        iteracion<=iteracion+1;
IF iteracion<500 then
control<="0101" ;
else
control<="0000" ;
end if;
OUT_1<=Xbest1;
OUT_2<=Xbest2;
OUT_3<=Xbest3;
OUT_4<=Xbest4;
end case;
end if;

end process initial_1;


END RTL;
```