Optimización mediante el algoritmo de colonia de abejas artificial

Estudiante: Silvana Yanet García

Proyecto Final presentado como requisito para obtener el título de Ingeniero en Sistemas

Facultad de Ingeniería
Universidad Nacional de La Pampa

Tutor: Carolina Salto

General Pico, La Pampa, Argentina

Agradecimientos

Quiero expresar mi sincero reconocimiento a todos los que, de distintas formas y posiciones, han contribuido para la elaboración de este trabajo.

En primer lugar agradecer a mis padres: Jorge y Susana por sus consejos, su apoyo incondicional, por estar conmigo en los buenos y malos momentos, agradecerles sobre todo por la dicha de tenerlos, Gracias Totales. A mi hermano Renzo por su compañía de siempre y sobre todo por estar en estos últimos años de estudio, motivándome en cada momento, Gracias!!

A mi directora de Tesis sra Carolina Salto, por su dedicación, motivación y tiempo, siempre dispuesta a ayudarme en todo y en todo momento, me sentí muy cómoda. Además a sus hijos Tomas, Martin y Anita que estuvieron presentes en muchas ocasiones y contribuyeron a la finalización de este trabajo, me llevo una gran amistad de todos…Muchas Gracias.

A todos mis amigos por acompañarme siempre y estar en los momentos que los necesite, junto a mi flia fueron muy importantes en este camino…. Muy agradecida.

Todos mis compañeros y amigos que coseche en la facultad, gracias a todos por estar, la ayuda de uno a otro hicieron que todos de una manera u otra logre este objetivo.

Índice General

Índice de tablas

Índice de figuras

Capítulo 1: Introducción

La colonia artificial de abejas (Artificial Bee Colony siglas en Ingles ABC) es uno de los algoritmos más recientes en el dominio de la inteligencia colectiva. Fue creado por Dervis Karaboga en el 2005 [1], motivado por el comportamiento inteligente observado en las abejas domésticas. En este algoritmo de optimización numérica basado en poblaciones, las soluciones representadas como fuentes de alimento, son modificadas por las abejas artificiales que se mueven aleatoriamente sin influencia externa en un espacio de búsqueda multidimensional o dependiendo de su experiencia pasada y de sus compañeras de colmena. El objetivo de estas abejas es descubrir las fuentes de alimento con mayor néctar, simulando soluciones óptimas locales y óptimas globales, intentando equilibrar la exploración y la explotación del algoritmo.

El algoritmo ABC comienza con la generación de forma aleatoria de la población inicial de soluciones al problema de optimización, cada solución representa una fuente de alimento. Luego de este proceso, la población entra en un ciclo de repeticiones del proceso de búsqueda de las abejas empleadas, observadoras y exploradoras. Las abejas empleadas modifican las soluciones actuales para mejorarlas. Las abejas observadoras se encargan de seleccionar una fuente de alimento, de acuerdo a la información que comparten las abejas empleadas mediante la danza y se encargan se introducir modificaciones y chequean si la calidad de la fuente de alimento (cantidad de néctar), en caso de ser mejor olvidan la solución y memorizan la nueva solución. La cantidad de abejas empleadas y observadoras es igual a la cantidad de soluciones en la población. Las abejas exploradoras crean una nueva fuente de alimento de manera aleatoria, para reemplazar fuentes existentes que no han sido mejoradas por un determinado periodo de tiempo (denominado límite). Este ciclo se repite hasta alcanzar la condición de corte establecido, por ejemplo un número total de ciclos.

El algoritmo ABC tiene en principio los siguientes parámetros de control: la cantidad de fuentes de alimentos, que es igual a la cantidad de abejas empleadas y observadoras, el valor límite en el cual una fuente de alimento permanece sin mejoras, y la cantidad máxima de ciclos del proceso de búsqueda. Los valores de estos parámetros determinan en una gran medida si el algoritmo encontrará una solución óptima o casi óptima, y si va a encontrar una solución eficiente. La elección de los valores de los parámetros correctos es, sin embargo, una tarea difícil.

Por esta razón, este trabajo se centrará, además de profundizar en el conocimiento de la metaheurística ABC, en el adecuado ajuste de parámetros previamente identificados a fin de obtener soluciones en forma eficiente, en particular nos centraremos en la cantidad de fuentes de alimentos. El valor límite de permanencia de una fuente de alimentos está configurado tomando como base la dimensionalidad del problema. Por otra parte, en el

algoritmo se usa un proceso de selección de las fuentes de alimentos que es proporcional a la cantidad de néctar de la misma. En este trabajo proponemos considerar otro mecanismo de selección basado en la simple comparación de la calidad de las fuentes de alimento. Para realizar la experimentación se utilizarán distintas funciones de optimización, que presentan distintas complejidades y dimensionalidades, muy usadas en la literatura. Se realizará una extensa experimentación, la cual será validada utilizando técnicas estadísticas, para dar soporte a las conclusiones extraídas.

1.1 Objetivos

Con el desarrollo de este trabajo final se persiguen los siguientes objetivos:

- ✓ Presentar los principios básicos de Inteligencia Colectiva, en particular del algoritmo de optimización por colonia de abejas artificial ABC.
- ✓ Desarrollar e implementar una aplicación de software basada en el algoritmo ABC para resolver problemas de optimización numérica.
- ✓ Analizar la influencia de la cantidad de fuentes de alimentos en el rendimiento del algoritmo ABC, para detectar la cantidad más adecuada con el fin de obtener buena calidad de resultados para las funciones de optimización consideradas.
- ✓ Proponer el uso de otro mecanismo de selección de fuentes de alimentos que utilizan las abejas observadoras. Realizar una comparación con el mecanismo tradicional utilizado por ABC, teniendo en cuenta calidad de soluciones y tiempo de cómputo.
- ✓ Realizar la experimentación y análisis de resultados.
- ✓ Presentar conclusiones respecto del comportamiento observado, validadas desde un punto de vista estadístico.

1.2 Organización del trabajo

El documento está estructurado de la siguiente forma:

- ✓ *Capítulo 2:* Principios de Optimización. En este capítulo se muestran los tipos de problemas y técnicas de optimización, además de la definición y los componentes de un algoritmo bio-inspirado.
- ✓ *Capítulo 3:* Inteligencia Colectiva. En este capítulo se explican los conceptos básicos sobre inteligencia colectiva, introduciendo brevemente a la optimización mediante cúmulos de partículas y la optimización basada en la colonia de hormigas como sus paradigmas iniciales, concluyendo con una introducción al paradigma de optimización por colonia de abejas.
- ✓ *Capítulo 4:* Algoritmo de la colonia artificial de abejas (ABC). En este capítulo se presentan con mayor detalle los principios heurísticos, elementos y funcionamiento del algoritmo ABC.
- ✓ *Capítulo 5:* Descripción del algoritmo ABC desarrollado. En este capítulo se detallan los distintos componentes del algoritmo utilizado junto con la modificación

propuesta para introducir mejoras. Además se detallan las funciones de optimización consideradas.

- ✓ *Capítulo 6:* Resultados. En este capítulo se presentan los resultados obtenidos en la experimentación realizada para abordar las funciones de optimización por medio del algoritmo ABC y se discuten los mismos.
- ✓ *Capítulo 7:* Conclusiones y trabajos futuros. Para finalizar, en este capítulo se muestran las conclusiones que se pueden extraer de todo este proceso de acuerdo a los resultados obtenidos, y los trabajos futuros que se deriva de esta tesis.

Capítulo 2: Principios de optimización

Los problemas de optimización son comunes en muchas disciplinas y varios dominios. En problemas de optimización, se deben encontrar soluciones que son óptimas o casi óptimas con respecto a algunos objetivos. Por lo general, no somos capaces de resolver problemas en un solo paso, pero seguimos un proceso que nos guía a través de la resolución de los mismos. A menudo, el proceso de solución se separa en diferentes pasos que se ejecutan uno tras otro. Comúnmente, los pasos utilizados son: reconocer y definir el problema, construir y resolver modelos y, finalmente, evaluar e implementar dichas soluciones.

En pocas palabras decimos entonces que los problemas de optimización consisten en buscar soluciones posibles para algún problema. Estas soluciones deben cumplir ciertas reglas planteadas. Los problemas de optimización se pueden clasificar en dos tipos:

- ✓ *Problemas de optimización numérica:* Al sustituir el conjunto de valores para las variables del problema en la función objetivo, estos hacen que maximice o minimice el valor de la función.
- ✓ *Problemas de optimización combinatoria:* Para maximizar o minimizar el valor de la función objetivo, estos encuentran el orden de un conjunto de variables.

En este trabajo nos enfocaremos en el problema de optimización numérica y su forma de resolver utilizando técnicas de optimización. Por tal motivo, procederemos a especificar con más detalle los problemas de optimización y a introducir y caracterizar las técnicas de optimización.

2.1. Problemas de optimización

Un problema de optimización consiste en encontrar un $\vec{x}$ que optimice $f(\vec{x})$ (1)

Sujeto a:

$$G_i(\vec{x}) \leq 0,\ i = 1, \dots, m \qquad (2)$$

$$G_j(\vec{x}) = 0,\ j = 1, \dots, p \qquad (3)$$

donde:

$$x_i(lo) < x_i < x_i(up),\ i = 1, \dots, n \qquad (4)$$

son los límites inferior y superior de cada variable del problema

$\vec{x}$ es un vector *n* dimensional que representa una solución $x_i = [x_1, x_2, \dots, x_n]^T$ (5)

El objetivo del problema es hallar una solución que minimice una cierta medida de calidad o función objetivo (1), donde además debe satisfacer un conjunto de condiciones de

desigualdad (2) e igualdad (3) asociadas al problema y que las variables de decisión (5) están acotadas por límites inferiores y superiores (4), los cuales definen el espacio de búsqueda.

2.2 Técnicas de optimización

Las técnicas para resolver problemas de optimización numérica se clasifican de la siguiente forma [2]:

1. Métodos tradicionales. Aquí se distinguen:

- ✓ *Técnicas de variables simples:* Existen dos tipos de métodos en estas técnicas: métodos del gradiente que utilizan la información del gradiente para guiar la búsqueda; y los métodos directos que utilizan la función a optimizar para regir la búsqueda.

- ✓ *Técnicas multivariables:* Estas técnicas también se dividen en dos métodos: directos y del gradiente. Efectúan búsquedas en múltiples dimensiones, valiéndose, en ocasiones, de técnicas de variables simples.

- ✓ *Técnicas para problemas con restricciones:* Utilizan técnicas multivariables y/o de variables simples para realizar búsquedas en espacios restringidos.

- ✓ *Técnicas especializadas:* estas técnicas solo utilizan métodos para problemas específicos como programación entera.

2. Métodos no tradicionales: Los métodos no tradicionales son los que reúnen conceptos heurísticos para mejorar la búsqueda en problemas del mundo real, entonces decimos que no podemos aplicar los métodos tradicionales ya que algunos problemas pueden ser muy particulares o en algunos casos imposibles.

   Por otro lado, los métodos no tradicionales pueden ser aplicados en ciertos problemas, sin embargo los resultados y/o el tiempo requerido para obtener una solución no son los esperados por quien resuelve el problema. Se pueden observar dificultades al tratar de resolver problemas de optimización cuando [2]:

- ✓ El número de posibles soluciones es demasiado grande.
- ✓ Con el propósito de obtener alguna solución de algún problema con cierta complejidad, se deben utilizar modelos de la misma complejidad y por ende la solución es poco útil.
- ✓ La función de evaluación que describe la calidad de cada solución en el espacio de búsqueda varía con el tiempo y/o tiene ruido.
- ✓ Existen soluciones posibles que están altamente restringidas, esto dificulta inclusive la generación de al menos una solución posible que cumpla con ciertas restricciones del problema.

Como primera opción para resolver un problema de optimización se debe pensar en los métodos tradicionales, dado que pueden certificar la convergencia al óptimo global bajo ciertas condiciones, es decir, a la mejor de todas las posibles soluciones. Sin embargo, si la aplicación del método tradicional es compleja o, el costo computacional es alto y/o los resultados son buenos pero no los esperados, en este caso estamos hablando de óptimos locales. En estos casos se puede recurrir al uso de un método no tradicional, es decir, una heurística.

Encontrar una solución óptima global a un problema no es garantizado por los métodos heurísticos, sin embargo, darán una solución aceptable en un tiempo razonable. En la literatura especializada existen reportes publicados que destacan en ciertos casos el uso de métodos no tradicionales en la solución de problemas complejos [2].

Las heurísticas bio-inspiradas [1] o algoritmos bio-inspirados están dentro de los métodos heurísticos. Su funcionamiento está basado en fenómenos "inteligentes" encontrados en la naturaleza. Estos algoritmos se distinguen de acuerdo al tipo de fenómeno natural en el que se basan y se dividen en dos sub grupos:

- ✓ *Algoritmos evolutivos* [3]*:* son métodos de optimización donde la búsqueda de soluciones está basada en la teoría de la evolución de las especies (evolución biológica). Mantienen un conjunto de entidades que representan posibles soluciones, donde se destaca la supervivencia del más apto y la trasmisión de características de padres a hijos.
- ✓ *Inteligencia colectiva* [1]*:* Son métodos basados en la conducta de los seres vivos que interactúan de manera local con su ambiente. Estos seres simples permiten resolver problemas complejos a través de los comportamientos sociales.

2.3 Componentes de un algoritmo bio-inspirado

Los algoritmos evolutivos y de inteligencia colectiva, de forma general, trabajan siguiendo el esquema que muestra la Figura 1. En primer lugar se genera un conjunto inicial de

soluciones al problema, conformando la población inicial. Luego se evalúa cada solución con la función objetivo a optimizar. En base al valor dado por la solución objetivo para esa solución, se seleccionan las mejores soluciones de la población. A partir de las soluciones seleccionadas y utilizando operadores de variación, se generan nuevas soluciones y se las evalúa. Por último, se escogen las soluciones que formarán parte de la nueva población y se prosigue con la siguiente iteración (generación).

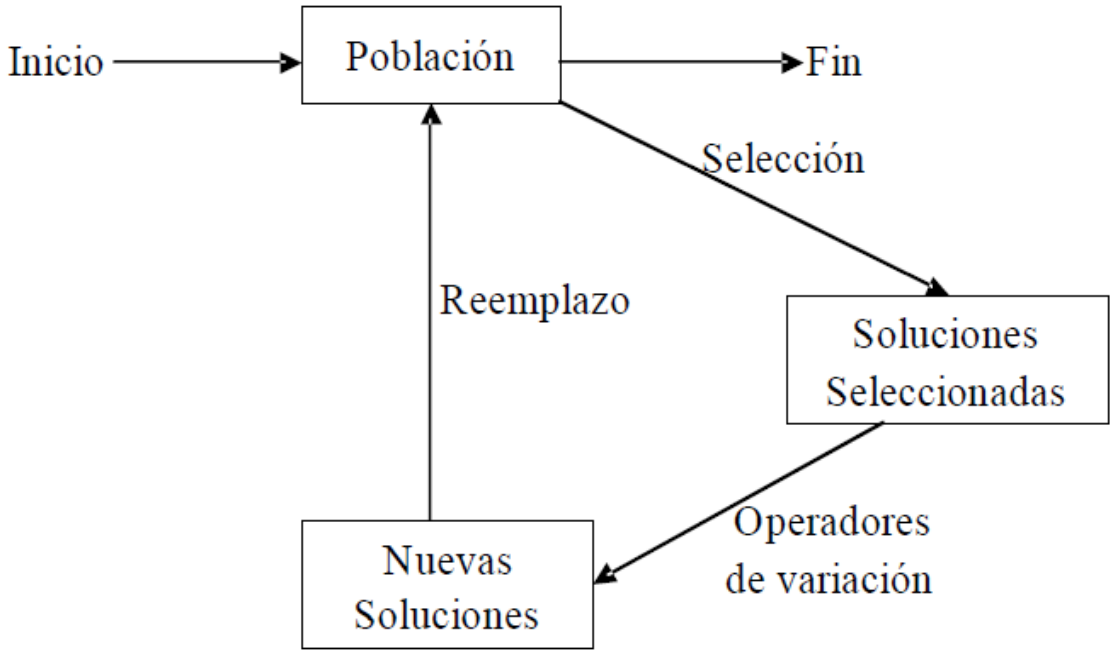

Figura 1. Esquema general de un algoritmo bio-inspirado

Entre los distintos algoritmos evolutivos y de inteligencia colectiva existen diferencias específicas, pero estos también tienen elementos en común, y son los siguientes:

- ✓ *Representación de soluciones:* Estas soluciones pueden verse de diferentes formas: como cadenas binarias y como números reales. Las variables de decisión toman valores reales, entonces en la optimización numérica la representación adecuada sería con números reales.

- ✓ *Mecanismo de selección:* La búsqueda se guía mediante este mecanismo o con el reemplazo. La intención es escoger un subconjunto de soluciones de la población que tengan características sobresalientes, de manera que puedan generar nuevas y mejores soluciones. El criterio para seleccionar depende del valor de la función objetivo de cada solución.

- ✓ *Operadores de variación:* Son los que permiten generar nuevas soluciones a partir de las soluciones existentes. Estos emulan comportamientos como la reproducción (apareamiento), variaciones aleatorias a nivel genético (mutaciones) o movimientos cooperativos (ir de un lado a otro).

- ✓ *Mecanismo de reemplazo:* Se encarga de seleccionar aquellas soluciones que formarán parte de la población para la siguiente generación. Este proceso puede basarse en la calidad de cada solución, en la "edad" de las soluciones, o tomando en cuenta aspectos estocásticos.

Capítulo 3: Swarm Intelligence (Inteligencia colectiva)

El objetivo en un problema de optimización es generalmente encontrar la mejor alternativa entre varias de ellas. Para estos problemas existen dos posibles estrategias de solución: mediante alguna técnica que reduce el problema o bien utilizando técnicas exhaustivas que en un tiempo razonable visiten todas las soluciones posibles del espacio de búsqueda. Existen problemas no simplificables con un espacio de búsqueda demasiado grande para explorarlo por completo con técnicas exhaustivas. Para ello se aplican las técnicas metaheurísticas. Decimos que las metaheurísticas son estrategias de alto nivel para explorar espacios de búsqueda. Por lo tanto, en este capítulo se realiza una breve introducción a las técnicas heurísticas bioinspiradas, en especial a los algoritmos de inteligencia colectiva aplicados a la resolución de problemas de optimización.

3.1 Inteligencia colectiva

Los algoritmos de Inteligencia colectiva (swarm intelligence), son técnicas heurísticas de inteligencia artificial basados en el estudio de comportamientos sociales y colectivos presentes en sistemas de la naturaleza, tales como los animales, generalmente de carácter descentralizado y auto organizativo. En esos sistemas el comportamiento social influye en los movimientos de las variables de decisión en el espacio de búsqueda y la orientación hacia soluciones óptimas. La expresión de Swarm Intelligence fue introducida por Gerardo Beni, Suzanne Hackwood y Jing Wang en 1989, en el contexto de sistemas robóticos celulares [1].

La Inteligencia Colectiva se trata de tareas propias del grupo, en las que intervienen las relaciones que se establecen entre los integrantes del grupo para realizar una tarea específica. Cabe mencionar que estas tareas realizadas por el grupo implican un proceso de búsqueda, que por lo general está relacionado con encontrar una fuente de alimento. Una característica de los algoritmos de inteligencia colectiva es que los individuos no son eliminados, sino que están en constante movimiento durante el proceso, ya que no cuenta con mecanismos de reemplazo de individuos como en los algoritmos evolutivos [4].

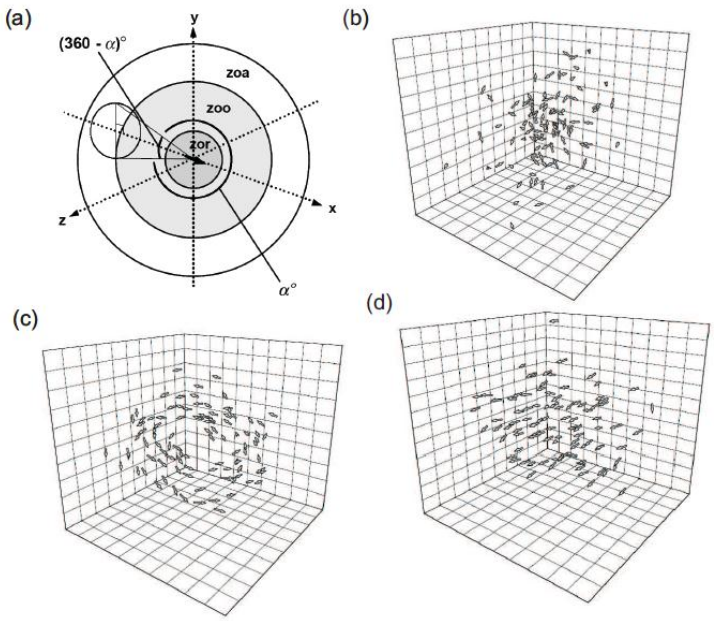

Figura 2. (a) Radio de influencia de un individuo, (b) colectivo Swarm, (c) colectivo Torus y (d) colectivo Dinamic/Highly Parallel Group

Los individuos deben mantener una distancia mínima entre ellos y el resto del cúmulo durante todo el proceso. Por otra parte, cada individuo debe evitar una distancia excesiva con el resto del grupo, de modo que tiende a ser atraído hacia los demás individuos, evitando así ser aislado y dando lugar a diferentes vecindarios dependiendo de la fuerza de atracción. Como se puede observar en la Figura 2 (a), el radio de influencia de cada individuo se compone de una zona inicial de repulsión (zor - para evitar colisiones), una zona de orientación (zoo - el individuo dirige su movimiento) y una zona exterior de atracción (zoa - el individuo evita ser aislado). Dependiendo de la configuración de cada zona del radio individual, se generaran diferentes comportamientos y formas en los vecindarios globales correspondiente a lo mostrado en las Figuras 2 (b), 2 (c) y 2 (d) [1].

Dentro de los algoritmos de inteligencia colectiva encontramos la optimización mediante cúmulos de partículas, la colonia de hormigas y la colonia artificial de abejas. En el trabajo se va a profundizar este último, pero a continuación se hará una introducción de los mismos.

3.2. Optimización mediante cúmulos de partículas

Dentro de las heurísticas de la inteligencia colectiva se encuentra la optimización por cúmulos de partículas (Particle Swarm Optimization siglas en inglés - PSO). Esta es una técnica evolutiva que fue propuesta por Kennedy y Eberhard (1995) [5]. Este método fue descubierto a través de un modelo social simplificado y está inspirado en el comportamiento social de organismos como los bancos de peces o bandadas de aves.

Se ha demostrado que PSO es una técnica muy eficiente para resolver problemas de optimización de tipo continuo [6]. Particularmente, PSO es un método que genera mejores resultados para problemas catalogados como difíciles. Sin embargo, se ha observado que su desempeño en problemas lineales no es muy bueno.

Para poder interpretar cómo actúa este tipo de algoritmo es necesario conocer su representación clásica, a partir de sus componentes.

3.2.1 Componentes de PSO

El algoritmo inicia con una población de soluciones aleatorias y busca el óptimo al actualizar generaciones. Una solución potencial es representada en PSO por un vector llamado partícula, individuo, elemento, agente o simplemente solución [7]. Cada una de estas partículas i tiene la misma dimensión n y se representan de la siguiente manera:

$$X_i = (x_{(i,1)}, x_{(i,2)}, ..., x_{(i,n)})$$

Las partículas, sobre el espacio de búsqueda n-dimensional, "vuelan" tratando de encontrar una solución óptima. Para hacer esto, cada individuo ajusta su posición de acuerdo a una combinación lineal de su inercia, su propia experiencia y del conocimiento del cúmulo.

Cada agente almacena en una memoria la mejor posición encontrada (visitada) hasta el instante actual t. La experiencia de la partícula se denota como:

$$P_i = (p_{(i,1)}, p_{(i,2)}, ..., p_{(i,n)})$$

El conocimiento del cúmulo es el conjunto de memorias de cada partícula. En PSO no existe la competencia entre individuos, por lo tanto, la interacción entre las partículas, para obtener un beneficio, es la norma. Para hacer esto, cada partícula pone a disposición de los demás su memoria de conocimiento.

La asignación de informantes (vecindario) es una forma de compartir la experiencia. Cada partícula recibe información de k agentes seleccionados de forma aleatoria en cada iteración del algoritmo. Después, la partícula determina entre sus informantes aquel que tenga la mejor posición previa. Posteriormente, lo selecciona para que sea parte del proceso de actualización de su posición. Normalmente, el valor de k es pequeño, sin embargo, puede ser tan grande como el tamaño del cúmulo. Este último criterio hace que las partículas tiendan hacia la mejor partícula encontrada hasta algún instante t. Las características del problema a solucionar determinan cuál de los dos métodos es más adecuado para su implementación.

La mejor partícula del vecindario se representa por:

$$G_i = (g_{(i,1)}, g_{(i,2)}, ..., g_{(i,n)})$$

Al igual que en otros algoritmo de tipo evolutivo, PSO necesita de una función de evaluación (también llamada función de posición). Esta permite determinar la calidad de las soluciones. Su importancia radica en que es la única forma de poder evaluar la posición de cada elemento.

Cada coordenada (elemento) de X de cada partícula tiene una velocidad o razón de cambio V, d= 1,2,..., n.

$$V_i = (v_{(i,1)}, v_{(i,2)}, ..., v_{(i,n)})$$

Para realizar un desplazamiento, la partícula determina la velocidad considerando su propia inercia W (busca evitar la convergencia prematura), su memoria de conocimiento y su confianza en el cúmulo, para después, sumarla a la posición actual.

El grado de confianza lo determinan los operadores aleatorios r_1 y r_2 (en el rango [0,..,-1]) junto con los coeficientes de confianza c_1 y c_2. Estos últimos, también llamados constantes de aceleración, son los términos que tiran a cada partícula hacia las posiciones P y G.

En otras palabras, las partículas hacen un movimiento hacia un punto intermedio tomando en cuenta la mejor posición previa, el mejor informante y un punto accesible desde la posición actual.

Las ecuaciones siguientes ajustan la velocidad y posición de cada partícula:

$$V_i^{t+1} = w^t V_i^t + c_1 r_1^t (P_i - X_i^t) + c_2 r_2^t (G_i - X_i^t)$$
$$X_i^{t+1} = X_i^t + V_i^{t+1}$$

dónde:

V_i^{t+1}, velocidad ajustada.

W^t , inercia del propio movimiento.

C_1 , coeficiente de confianza en la experiencia.

C_2 , coeficiente de confianza en la experiencia del grupo.

P_i , mejor posición previa de *i*.

X_i^t , posición actual de *i*.

G_i , mejor posición previa encontrada por el grupo.

r_1^t y r_2^t , operadores aleatorios entre 0 y 1.

X_i^{t+1} , posición de la partícula *i* después del ajuste.

Algoritmo 1: PSO clásico

t=0

Nube ← Inicializar Nube de Partículas

Repetir. Inicia el ciclo de búsqueda.

Para *i*=1 hasta tamaño (*Nube*) **hacer**

Evaluar la aptitud de cada partícula $\boldsymbol{x_i}$ de la *Nube*

Si *aptitud_*x_i es mejor que *aptitud_mejorpos*$_i$ **entonces**

mejorpos$_i$ ← x_i;

Fin Si

Si *aptitud_mejorpos*$_i$ es mejor que *aptitud_mejorpos* **entonces**

mejorpos ← *mejorpos*$_i$;

aptitud_mejorpos ← *aptitud_mejorpos*$_i$

Fin Si

Fin Para

Para *i*=1 hasta tamaño (*Nube*) **hacer**

Calcular la velocidad $\mathbf{v_i}$ de $\boldsymbol{x_i}$, en base a los valores

$\boldsymbol{x_i}$, **mejorpos**$_\mathbf{i}$ y **mejorpos**

Calcular la nueva posición de $\boldsymbol{x_i}$, de su valor actual y $\boldsymbol{v_i}$

Fin Para

t=*t*+1

Hasta (no se alcance la condición de parada)

Salida: Devuelve la mejor posición encontrada.

El pseudocódigo del PSO se muestra en el Algoritmo 1.

Debido a que en el proceso de actualización de la velocidad están inmiscuidos operadores aleatorios, el método de modificación de la posición de las partículas se puede considerar a fin de cuentas un paso estocástico, pero heurístico.

En términos generales, PSO se puede describir en tres pasos. El primero consiste en evaluar cada elemento para determinar la calidad de la posición actual. Esto permite que se puedan encontrar la mejor y las mejores partículas. Después, se deben realizar los ajustes necesarios de las mejores posiciones previas. Por último, se determinan los nuevos desplazamientos para cada partícula con la información ajustada. Por analogía, estos movimientos no son más que una forma de tratar de imitar a otros individuos.

Para detener el proceso se necesita que se cumpla un criterio de parada. Este puede ser determinado por un número fijo de ciclos, opcionalmente, combinado con un umbral de error aceptable. Al terminar la ejecución del algoritmo, la solución que reporta el método es la mejor posición previa encontrada por alguna partícula.

La versión original de PSO presentaba algunas desventajas. Si la mejor solución está estancada en algún óptimo local, todas las partículas tenderán rápidamente a concentrarse en ese punto. Otra importante desventaja era su poco control para tener un balance entre la exploración y la explotación. Para contrarrestar estos problemas, se utiliza el coeficiente de inercia W [7].

3.3. Colonia de hormigas

El algoritmo de optimización mediante colonias de hormigas (Ant Colony Optimization siglas en inglés ACO) fue desarrollado por Dorigo en 1992 [8]. ACO es un algoritmo probabilístico que imita la habilidad de las hormigas para encontrar el camino más corto desde su hormiguero hasta una fuente de alimento.

ACO esencialmente es un algoritmo constructivo; en cada iteración del algoritmo, cada hormiga construye una solución al problema recorriendo un grafo de construcción. Cada arco del grafo, representa los posibles pasos que la hormiga puede dar, tiene asociada dos tipos de informaciones que guían el movimiento de la hormiga [9]:

- ✓ *Información heurística:* mide la preferencia heurística de moverse desde de una posición a otra. Las hormigas no modifican esta información durante la ejecución del algoritmo.
- ✓ *Información de los rastros de feromona artificiales:* mide la "deseabilidad aprendida" del movimiento. Copia a la feromona real que colocan las hormigas naturales. Dependiendo de las soluciones encontradas por las hormigas, esta información se modifica durante la ejecución del algoritmo.

Decimos que este es un proceso muy interesante ya que las hormigas son insectos casi ciegos, donde pueden encontrar el camino más corto entre la fuente de alimento y su nido. Las hormigas se comunican mediante el tacto y el olfato, pero en lugar de oler el aire utilizan sus antenas para detectar olores muy sutiles producidos por hidrocarburos en sus alrededores. La superficie exterior del cuerpo de una hormiga contiene unos 25

hidrocarburos diferentes, que emiten aromas ligeramente diferentes, imperceptibles para los humanos.

Los cambios limitados en la concentración de los hidrocarburos (moléculas simples de hidrógeno y carbono) pueden provocar modificaciones de comportamiento interesantes entre las hormigas. Estos hidrocarburos que sirven para comunicarse se denominan feromonas. Al principio, las hormigas tienen una distribución aleatoria por todo el espacio de búsqueda, ya que las mismas desconocen el camino más corto entre el nido y el alimento.

Con el tiempo la cantidad de feromona se hará más presente en aquellos lugares en los que fueron mayormente explorados y así se evitará su rápida inhibición, siempre teniendo en cuenta que las hormigas se mueven a una velocidad constante. Es decir, que la distancia entre la fuente de alimento y el nido será transitada con mayor frecuencia por las hormigas, y de esta manera permitirá hacer más ágil el rastro del camino más corto.

Utilizando la feromona como medio de intercambio de información lo que se conoce como "stigmergy", las hormigas pueden establecer rutas cortas, de tal manera que aquellos rastros que contengan mayor cantidad de feromona serán recorridos con mayor seguridad por las hormigas.

En general, un algoritmo de ACO puede ser visto como la interrelación de tres procedimientos [10]:

- ✓ *Construcción de soluciones por hormigas:* administra una colonia de hormigas que visitan estados aleatoriamente de un problema considerado. Las hormigas pueden moverse aplicando una toma de decisión estocástica usando la información de los rastros de feromona y la información heurística. De esta forma, las hormigas construyen una solución óptima al problema.
- ✓ *Actualización de feromona:* proceso mediante el cual los rastros de feromona son modificados. El valor del rastro puede incrementarse debido a que las hormigas depositan feromona en cada uno de los componentes o conexiones que usan para moverse de un nodo a otro del problema.
- ✓ *Control de acciones:* procedimiento utilizado para llevar a cabo acciones centrales en las que las hormigas no tienen capacidad para desarrollarlas en forma individual.

En el Algoritmo 2 se muestra el pseudocódigo de ACO:

Algoritmo 2: ACO clásico

InicializarValoresFeromonas(T); {Inicializa los rastros de feromona}

antbs = generarSolución(T, *n*){Se inicializa antbs con una solución inicial aleatoria}

REPETIR. Inicia el ciclo de búsqueda

Para j→1 to μ **hacer**

ant_i = construirSolución(T, *n*); {La hormiga ant_i construye una solución}

actualizarFeromonaLocal(T , ant_j); {Actualización local de los rastros de feromona (ACS)}

Fin para

evaporarFeromonas(T) {Evaporación}

actualizarFeromonasGlobal(T, ant_i, *antbs*) {intensificación, actividad del demonio}

Para j→1 to μ **hacer**

Si f (ant_i) < f (*antbs*) **entonces**

antbs= ant_i {Actualizar la mejor solución, actividad del demonio}

Fin si

Fin para

HASTA (no (condición de terminación))

Salida: la mejor solución encontrada

3.4 Problemas que se resuelven usando ACO y PSO

En esta sección mencionaremos y describiremos en breve algunos de los problemas que pueden resolverse con los algoritmos mencionados anteriormente (ACO - PSO). Aquí van algunos ejemplos aunque el listado es interminable:

- ✓ El Problemas de Asignación Cuadrática (*Quadratic Assignment Problem*, por sus siglas en inglés QAP) fue introducido por Koopmans y Beckmann en 1957 como un modelo matemático para la ubicación de un conjunto de actividades económicas indivisibles [11]. El QAP puede ser descrito mejor como el problema de asignar un conjunto de elementos a un conjunto de ubicaciones, donde hay distancias entre las ubicaciones y flujos entre los elementos. El objetivo entonces es ubicar los elementos en las ubicaciones de forma tal que la suma del producto entre flujos y distancias sea mínima.

- ✓ Location Area Management (LA): Con la finalidad de encaminar las llamadas recibidas a los terminales móviles apropiados, una red inalámbrica debe recopilar periódicamente y mantener información sobre la localización de cada terminal móvil. Esto supone un consumo importante de los limitados recursos de dicha red. Además del ancho de banda utilizado para el registro (*location update*) y localización (*paging*) entre terminales móviles y estaciones base, también se consume energía de los dispositivos portátiles. Más aún, las frecuencias de las señales se pueden degradar perjudicando a la calidad de servicio (*Quality of Service QoS*) debido a la aparición de interferencias. Por otra parte, un error en la localización de un terminal obligaría a realizar una búsqueda adicional cuando se recibiese una nueva llamada lo cual incrementaría el consumo aún más. Por lo tanto, el objetivo de este problema es equilibrar las operaciones de registro y búsqueda para de esta forma minimizar el coste en el seguimiento de la localización de los terminales móviles [12].

- ✓ Una definición general del TSP (*Problema del Viajante de Comercio*) es la siguiente. Considere un conjunto de nodos N, representando ciudades, y un conjunto de arcos E conectando completamente los nodos de N. Sea $\boldsymbol{d_{ij}}$ la longitud del arco $(i, j) \in E$, esto es la distancia entre las ciudades i y j, con $i, j \in N$. El TSP es el problema de encontrar un circuito Hamiltoniano de longitud mínima sobre el grafo $G = (N, E)$, donde un circuito Hamiltoniano del grafo G es un tour cerrado que visita una y solo una vez todos los nodos $n = |N|$ de G, y su longitud está dada por la suma de las longitudes de los arcos del cual está compuesto el tour.

3.5 Colonia artificial de abejas

La colonia artificial de abejas (Artificial Bee Colony siglas en ingles ABC) es un algoritmo de optimización basado en poblaciones de abejas, caracterizado por su sencillez y facilidad de implementación. Podemos decir que es un algoritmo efectivo para resolver problemas de búsqueda compleja en diversos dominios [13]. Existen varias versiones de ABC, Baykasoglu [14] plantea una clasificación basada en tres principales modelos:

- ✓ *Comportamiento de la abeja reina:* El mejor individuo de una población puede generar varios descendientes. Se dice que este algoritmo es una adaptación de un algoritmo genético. Este modelo ha resuelto problemas combinatorios y algunas funciones numéricas no restringidas.

- ✓ *Comportamiento de unión:* Está basado en el comportamiento de apareamiento. Existen adaptaciones obtenidas de trabajos para resolver problemas numéricos no restringidos y algunos con restricciones simples. Sin embargo, no se especifican como se adaptó el modelo para el manejo de restricciones. Las funciones de prueba utilizadas son muy sencillas.

- ✓ *Comportamiento de forrajeo:* Para la obtención de fuentes de alimento las abejas domésticas se centran en el comportamiento colectivo. Actualmente existen 20 propuestas, en las cuales 16 resuelven problemas de optimización combinatoria y solo 4 están diseñadas para resolver problemas de optimización numérica, pero sin restricciones [15], además de que no existen reportes de que se hayan utilizado en problemas de ámbito industrial, ya que actualmente la investigación se centra mayormente en problemas de benchmark [16].

En el capítulo 4 nos enfocaremos en explicar detalladamente el modelo de forrajeo que propone Dervis Karaboga para resolver problemas de optimización numérica, debido a que es el objeto del trabajo de tesis.

Capítulo 4: Algoritmo de la colonia artificial de abejas

El algoritmo de la colonia artificial de abejas (ABC) es uno de los algoritmos más recientes en el dominio de la inteligencia colectiva. Fue propuesto por Dervis Karaboga en 2005 [17] está basado en el comportamiento de forrajeo de las abejas, y diseñado originalmente para problemas de optimización numérica, aunque puede ser utilizado para resolver problema de combinatoria.

ABC es un algoritmo de optimización inspirado en poblaciones, donde las soluciones del problema de optimización, llamadas fuentes de alimento, son modificadas por las abejas artificiales, que se desempeñan como operadores de variación. El objetivo de estas abejas es descubrir las fuentes de alimento con mayor néctar [15].

4.1 Comportamiento biológico

El proceso de búsqueda de néctar por parte de las abejas es un proceso de optimización, y el comportamiento de estas se modeló como una heurística de optimización basada en el modelo biológico que consta de los siguientes elementos [15]:

- ✓ *Fuente de alimento:* el valor de una fuente de alimento depende de muchos factores, como su proximidad a la colmena, riqueza o la concentración de la energía y la facilidad de extracción de esta energía. Es resumido en un valor numérico que indica su potencial.

- ✓ *Abejas recolectoras empleadas:* están asociadas a una fuente de alimento. Llevan con ellas información sobre esa fuente en particular, su distancia, ubicación y rentabilidad para compartirla a las abejas observadoras.

- ✓ *Abejas recolectoras desempleadas:* este tipo de abejas se encuentran buscando fuentes de alimento para explotar. Hay dos tipos:
  - ➢ *Exploradoras:* se encargan de buscar nuevas fuentes de alimento en el ambiente que rodea a la colmena. Es decir, lleva información sobre una fuente específica y la comparte con otras abejas esperando en la colmena. La información incluye la distancia, la dirección y el néctar de la fuente de alimento.
  - ➢ *Observadoras:* Con la información compartida por las empleadas o por otras exploradoras en el nido, estas buscan una fuente de alimento.

El intercambio de información entre las abejas es el más importante suceso en la formación del conocimiento colectivo, ya que mediante esta interacción las abejas decidirán el comportamiento que debe llevar la colmena.

Las abejas empleadas comunican la información de la fuente de alimento que están explotando a las abejas observadoras por medio de una danza, donde el ángulo respecto al sol indica la dirección de la fuente y un zigzagueo indica la distancia. Las danzas con mayor duración describen las fuentes de alimento más rentables y más probables a ser elegidas por las abejas observadoras [17]. Una vez que las fuentes de alimento se han agotado son abandonadas (ya sea por las abejas empleadas u observadoras) y reemplazadas por nuevas fuentes encontradas por las abejas exploradoras.

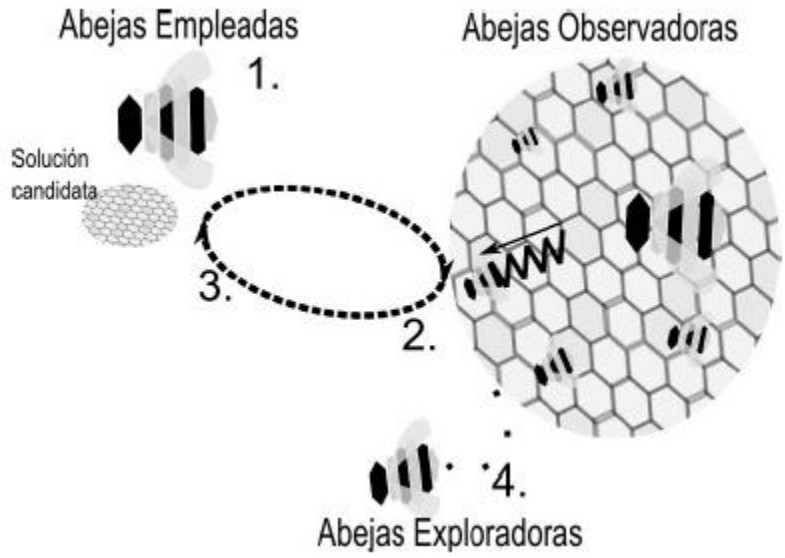

Figura 3. Comportamiento del forrajeo de las abejas

En la Figura 3 vemos las abejas empleadas asignadas a una fuente de alimento (1). Luego en (2) se observa la comunicación de información de las fuentes de alimento por medio de una danza, y las abejas observadoras visitan las fuentes de alimento más prometedoras (3). Y por último, las abejas exploradoras buscan nuevas fuentes (4).

4.2 Comportamiento artificial

A continuación se describen de manera general los elementos del algoritmo ABC:

- ✓ *Generación de las fuentes de alimento:* Se lleva a cabo de manera aleatoria y con base en los límites inferior y superior de cada variable del problema. Una fuente de alimento es una solución al problema de optimización.

- ✓ *Abejas empleadas:* Su número es proporcional al número de fuentes de alimento, es decir por cada fuente hay una abeja empleada. Su función es evaluar y modificar las soluciones actuales para mejorarlas (busca nuevas fuentes cercanas a la actual). Si la nueva posición no es mejor entonces se mantiene la posición actual.

- ✓ *Abejas observadoras:* al igual que las abejas empleadas, su número es proporcional al número de fuentes de alimento. Estas abejas seleccionarán una fuente de alimento, de acuerdo a la información que comparten las abejas empleadas mediante la danza.

- ✓ *Abejas exploradoras:* Estas abejas crean una nueva fuente de alimento de manera aleatoria, para reemplazar fuentes existentes que no han sido mejoradas.

- ✓ *Límite:* determina el número máximo de ciclos que una fuente de alimento puede persistir sin mejorar antes de ser reemplazada. El límite se incrementa a partir de que una fuente que no es modificada por las abejas, ya sean empleadas u observadoras, hasta obtener su valor máximo permitido, de acuerdo a esto, las abejas exploradoras se encargan de inicializar el límite a 0 por cada nueva posición generada. El límite se inicializa a 0 cada vez que se modifica (mejora) una fuente por una abeja empleada u observadora.

4.3 Algoritmo básico

En la sección 4.2 vemos claramente cómo se comportan las abejas según su rol. A continuación en el Algoritmo 3 explicaremos los principales pasos del mismo [17]:

Algoritmo 3: Principales Pasos del Algoritmo ABC

1- Inicializar la población.
2- REPETIR.
   - ✓ Colocar las abejas empleadas en las fuentes de alimento;
   - ✓ Colocar las abejas observadoras en las fuentes de alimento dependiendo de la cantidad de néctar;
   - ✓ Enviar a las exploradoras a la zona de búsqueda para descubrir nuevas fuentes de alimento.
   - ✓ Memorizar la mejor fuente de alimento encontrada.
3- HASTA (se cumplen los requisitos de terminación).

En el algoritmo ABC cada ciclo de la búsqueda consiste en tres pasos:

- ✓ El envío de la abejas empleadas a las fuentes de alimento y evaluar sus cantidades de néctar;

- ✓ Selección de las fuentes de alimento por parte de las observadoras después de compartir la información de las abejas empleadas y determinar la cantidad de néctar de las fuentes de alimento;
- ✓ La determinación de las abejas exploradoras y luego enviarlas a posibles fuentes de alimento en forma aleatoria.

En la fase de inicialización, las abejas empleadas seleccionan al azar un conjunto de posiciones de las fuentes de alimento y se determinan sus cantidades de néctar. Entonces, estas abejas entran en la colmena y comparten la información de néctar de las fuentes de alimento con las abejas observadoras, que esperan en la zona de baile dentro de la colmena. Después de compartir la información, cada abeja empleada retorna a la zona de la fuente de alimento visitada previamente, ya que existe una fuente de alimento en su memoria; y a continuación elige una nueva fuente de alimento por medio de la información visual en el vecindario y evalúa su cantidad de néctar.

Una abeja observadora elige una fuente de alimento dependiendo de la información distribuida por las abejas empleadas en la zona de baile dependiendo de la cantidad de néctar de la fuente. Si la cantidad de néctar de una fuente de alimento aumenta, entonces también aumenta la probabilidad con la que la fuente de alimento es elegida por una observadora. Después que la abeja observadora llega a la zona seleccionada, elige una nueva fuente de alimento en el vecindario dependiendo de la información visual, como en el caso de la abeja empleada. Esta selección de una nueva fuente de alimento se realiza comparando las posiciones de las mismas en forma visual. Cuando una fuente de alimento es abandonada por las abejas, una abeja exploradora determina una nueva fuente de alimento en forma aleatoria sustituyendo a la abandonada. En cada ciclo una exploradora sale a buscar una nueva fuente de alimento como máximo. El número de abejas empleadas y observadoras es igual. Estos pasos son repetidos por un número determinado de ciclos o hasta que se cumpla un criterio de terminación específico.

En el algoritmo ABC, la posición de una fuente de alimento representa una posible solución del problema de optimización y la cantidad néctar de una fuente de alimento corresponde a la calidad (fitness) de la solución asociada. El número de las abejas empleadas o de las abejas observadoras es igual al número de soluciones en la población.

En la primer etapa, el ABC genera una población inicial P (C = 0), distribuida al azar, de SN soluciones (posiciones de las fuentes de alimento), donde SN indica el tamaño de la población. Cada solución (fuente de alimento) x_i (i = 1, 2, ..., SN) es un vector D-dimensional, donde D es el número de parámetros de optimización. Después de la inicialización, la población en las posiciones (soluciones) se somete a ciclos repetidos, C = 1, 2,. . . , C_{max} , del proceso de búsqueda de las abejas empleadas, observadoras y las exploradoras.

Una abeja empleada u observadora produce probabilísticamente una modificación de la posición (solución) en su memoria para encontrar una nueva fuente de alimento y pone a prueba la cantidad de néctar (valor de fitness) de la nueva fuente (nueva solución).

La producción de nuevas fuentes de alimento se basa en un proceso de comparación de las fuentes de alimento en una región dependiendo de la información recopilada visualmente por las abejas. Como así también, podemos decir que la producción de una nueva posición de la fuente de alimento se basa en un proceso de comparación de las posiciones de las fuentes de alimento, pero las abejas artificiales no utilizan ninguna información en comparación sino que ellas seleccionan al azar una posición de fuente de alimento y producen una modificación en la que existe en su memoria. Si la cantidad de néctar de la nueva fuente de alimento es mayor que la de la anterior, la abeja memoriza la nueva posición y se olvida de la antigua, caso contrario se mantiene la posición anterior.

Después de que todas las abejas empleadas completan el proceso de búsqueda, comparten la información del néctar de las fuentes de alimento y su información de posición con las abejas observadoras en la zona de baile, donde la abeja observadora evalúa esa información de todas las abejas empleadas y elige una fuente de alimento con una probabilidad relacionada con su valor de néctar.

Las fuentes de alimento representan a cada solución como un vector n-dimensional, y las abejas a los operadores de variación. Cuando ellas visitan las fuentes de alimento calcularán una nueva solución u_{ij} de acuerdo a la ecuación 4.3.1, dónde x_{ij} representa la fuente de alimento dónde se encuentra la abeja en ese momento, x_{kj} es una fuente de alimento seleccionada aleatoriamente y diferente a x_{ij}, j es el ciclo actual y ϕ es un número real aleatorio entre [-1; 1]. Por último, u_{ij} denota la ubicación de la nueva solución con respecto a la posición actual x_{ij}.

$$u_{ij} = x_{ij} + \emptyset_{ij} * (x_{ij} - x_{kj}) \qquad (4.3.1)$$

Las abejas observadoras seleccionan las fuentes de alimento de acuerdo a una probabilidad p_i asociada a la fuente de alimento, la cual se calcula (según Ecuación 4.3.2).

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \qquad (4.3.2)$$

dónde fit_i es el valor de aptitud de la solución. El valor es proporcional a la cantidad de néctar que tiene la solución *i*.

En el Algoritmo 4 se muestra el pseudocódigo detallado del funcionamiento del ABC clásico.

Algoritmo 4: Algoritmo ABC clásico

1- Inicializar la población de soluciones x_{ij};0; i = 1, … , SN
2- Evaluar la población
3- **Para** c = 1 hasta Cmax **hacer**
4- Generar nuevas soluciones u_{ij} para las abejas empleadas usando la Ecuación 4.3.1 y evaluarlas
5- Conservar la mejor solución entre x_{ij}y u_{ij}
6- Seleccionar las soluciones que serán visitadas por las abejas observadoras usando la Ecuación 4.3.2
7- Generar nuevas soluciones u_{ij} para las abejas observadoras usando la Ecuación 4.3.1 y evaluarlas.
8- Conservar la mejor solución entre x_{ij}y u_{ij}
9- Verificar si existen fuentes abandonadas (si ya se alcanzó el Límite) y reemplazarlas por soluciones generadas aleatoriamente por las abejas exploradoras.
10- Conservar la mejor solución encontrada hasta el momento
11- **Fin Para**
12- Retornar la mejor solución.

Según lo expresado en los párrafos anteriores, los parámetros del algoritmo son los siguientes:

- ✓ *SN:* es el número de fuentes de alimento
- ✓ *Cmax:* el número total de ciclos que ejecutará ABC
- ✓ *Límite (limit):* número de ciclos que será conservada una solución sin mejora antes de ser reemplazada por una nueva solución generada por una abeja exploradora.

4.4 Variante de ABC propuesta

El algoritmo ABC básico utiliza una selección proporcional al fitness para que las abejas observadoras elijan las posiciones de las fuentes de alimento. Nuestra propuesta se basa en introducir una variante al algoritmo original donde modificamos este mecanismo de selección que realizan las abejas observadoras. La modificación consiste en utilizar un mecanismo de selección por torneo. Este método consiste en realizar la selección en base a comparaciones directas entre las fuentes de alimento.

Existen dos versiones de selección mediante torneo:

- ✓ *Determinística*

- ✓ *Probabilística*

En la primera versión se selecciona al azar un número de *t* de fuentes de alimento, por lo que generalmente se escoge *t*=2 (torneo binario). Luego entre las fuentes de alimento seleccionadas se elige la de mayor fitness como para utilizar como fuente de alimento x_{ij} en la ecuación 4.3.1.

En la versión probabilística, la diferencia radica en el paso de selección de la fuente de alimento ganadora del torneo. En lugar de escoger siempre la mejor, se genera un número aleatorio en el intervalo [0,...,1], si es mayor que un parámetro *p* (fijado para todo el proceso evolutivo) se escoge la fuente de alimento de mayor néctar y en caso contrario, la de menor cantidad de néctar. Generalmente *p* toma valores en el rango $0.5 < p < 1$.

Variando la cantidad de soluciones que participan en cada torneo se puede modificar la presión selectiva. Cuando participan muchas soluciones en cada torneo, la presión selectiva es elevada y las peores soluciones apenas tienen oportunidades de aportar su información en generación de nuevas posiciones. Un caso particular es el elitismo global. Se trata de un torneo en el que participan todos los individuos de la población con lo cual la selección se vuelve totalmente determinística. Cuando el tamaño del torneo es reducido, la presión selectiva disminuye y las peores soluciones tienen más oportunidades de ser seleccionadas. En este trabajo se optó por utilizar la versión determinística por ser la más usada en la literatura.

Capítulo 5: Descripción del algoritmo ABC desarrollado

Como se detalló en los capítulos anteriores, ABC es un algoritmo que se utiliza principalmente para resolver problemas de optimización numérica no restringida o de optimización combinatoria, usando poblaciones de abejas, y donde su objetivo principal es descubrir las mejores fuentes de alimento, o dicho con otras palabras, encontrar soluciones óptimas.

En este capítulo se presenta en detalle el algoritmo de la colonia artificial de abejas propuesto originalmente por Dervis Karaboga para resolver una serie de funciones de optimización. En primer lugar, se hará una presentación del pseudocódigo desarrollado e implementado de las distintas funcionalidades del algoritmo ABC con la descripción de los cuatros métodos más importantes. Para cerrar el capítulo describiremos y caracterizaremos las funciones de optimización utilizadas para probar el comportamiento del algoritmo.

5.1 Pseudocódigo desarrollado

Algoritmo 5: Algoritmo ABC desarrollado

1- Lectura de parámetros:

número_fuentes
límite_permitido
max_ciclos
2 – Etapa de Inicialización
  a- Se inicializa la colonia: X_i ($1 \leq i \leq$ número_fuentes)
  b- Elegir la mejor fuente de *X* y guardarla en *mejor_X*
3 - j =1 (ciclos)
4 – REPETIR. Inicia el ciclo de búsqueda
5 – Procedimiento crear_empleada (sección 5.1.1)
6 - Procedimiento observadora (sección 5.1.2)
7 – Procedimiento exploradora (sección 5.1.3)
8- Una vez procesados los pasos anteriores se evalúa la colonia y se memoriza cuál de todas las fuentes de alimento actuales es la mejor en *mejor_X*.
9 - j = j + 1;
10 - HASTA (max_ciclos).
11 – FIN

El pseudocódigo del ABC desarrollado en este trabajo se muestra en el Algoritmo 5. Comienza con la lectura de los parámetros necesarios para el funcionamiento del algoritmo (paso 1):

- ✓ *número_ fuentes:* cantidad de fuentes de alimento consideradas,
- ✓ *límite_permitido:* número máximo de ciclos que una fuente de alimento puede permanecer sin mejorar antes de ser reemplazada.
- ✓ *max_ciclos:* cantidad de ciclos del bucle central del algoritmo.

En el paso 2 se inicializa en forma aleatoria la colonia artificial X y se determina la mejor solución inicial mejor_X. En las siguientes secciones se describirán los métodos desarrollados en la implementación del pseudocódigo.

5.1.1 Procedimiento crear_empleada

Analizamos a continuación en forma detallada el funcionamiento del procedimiento crear_empleada, cuyo pseudocódigo se muestra en el Algoritmo 6. En este procedimiento se crean nuevas soluciones para las abejas empleadas para cada posición o fuente usando la siguiente evaluación: Uij = Xij + ϕij(Xij-Xkj), donde:

- ✓ Xij es la fuente de alimento actual
- ✓ Xkj es una posición tomada aleatoriamente conforme al número de fuentes existentes.
- ✓ Φij valor aleatorio entre 0 y 1

A continuación se analiza si el valor de Uij está contenido entre el límite inferior y superior de cada variable del problema. En caso de no estarlo se asigna el valor inferior o superior según corresponda.

Algoritmo 6: crear_empleada

1 – REPETIR.
2 - U*ij* = X*ij* + ϕ*ij*(X*ij*-X*kj*), donde *i* = 1, 2, ..., *SN (*posiciones de la fuente de alimento*), j* es la solución actual y *k* solución elegida al azar.
3- Se aplican los criterios para el manejo de los límites de U*i*
4 - Se evalúa U*i*
5 – HASTA (numero_fuentes)
6 – Se actualiza X*i*
7 - FIN

5.1.2 Procedimiento observadora

Continuando con el análisis de los procedimientos de las abejas, vamos a detallar el funcionamiento del procedimiento observadora. Aquí se producen nuevas soluciones por las abejas observadoras bajo uno de los siguientes métodos de selección:

- ✓ *Torneo*

- ✓ *Ruleta*

Una vez llevado a cabo el torneo o la ruleta se repiten los procesos de las empleadas. Luego se verifican el límite de cada variable del problema. Posteriormente se evalúa X*i* y se actualiza la colonia.

Algoritmo 7: observadora

1 – REPETIR.
2 - Se elige una fuente X_k siguiendo un método de selección //torneo o proporcional con ruleta
3 - U*ij* = X*ij* + ϕ*ij*(X*ij*-X*kj*), donde *i* = 1, 2, ..., *SN (*posiciones de la fuente de alimento*), j* es la solución actual y *k* solución elegida al azar.
4 - Se aplican los criterios para el manejo de los límites de U*i*
5 - Se evalúa U*i*
6 – HASTA (número_fuentes)
7 - Se actualiza X*i*
8 – FIN

5.1.3 Procedimiento exploradora

Esta sección está dedicada al análisis del procedimiento de las abejas exploradoras, donde se determina el abandono de una fuente de alimento por parte de las abejas empleadas en el caso de que se supere el límite permitido, por lo tanto se reemplaza la posición actual por una nueva posición que es generada aleatoriamente.

Algoritmo 8: exploradora

1-REPETIR

2 – Si el límite permitido de U*i* es superior limite_permitido

3 – Se inicializa U*i* en forma aleatoria

4 - Se evalúa U*i*

5 – HASTA (número_fuentes)
6 – FIN

5.2 Funciones de optimización

El comportamiento del algoritmo de optimización ABC propuesto se examina usando funciones de optimización. A continuación se enumeran algunas de las funciones y conjuntos de datos comúnmente utilizados para las pruebas de algoritmos de optimización en la literatura. Por cada función se realiza una descripción de las características de las mismas (modalidad, dimensionalidad, etc.), se presenta el gráfico en dos dimensiones,

además se especifica el dominio de entrada y mínimo global. Se agrupan de acuerdo a las similitudes en sus propiedades físicas y formas significativas [18].

5.2.1 Funciones con muchos mínimos locales (*Many Local Minima*)

- ✓ **Función Ackley**

Descripción:

La función Ackley es ampliamente utilizada para las pruebas de los algoritmos de optimización. En su forma de dos dimensiones, como se muestra en la Figura 4, se caracteriza por una región externa casi plana, y un gran orificio en el centro. La función representa un riesgo para los algoritmos de optimización, en particular los algoritmos hillclimbing, de ser atrapados en uno de sus muchos mínimos locales. Los valores de las variables recomendadas son: $a = 20$, $b = 0{,}2$ y $c = 2\pi$. Esta función es de dimensión d.

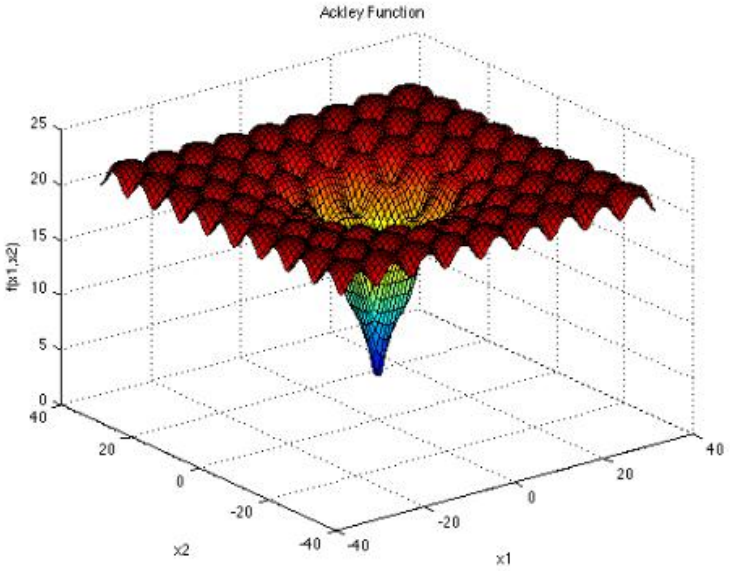

Figura 4. Gráfico de la Función Ackley

Función:

$$f(\mathbf{x}) = -a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$

Dominio de entrada:

La función se evalúa por lo general en el hipercubo $x_i \in$ [-32.768, 32.768], para todo $i = 1, \ldots, d$, aunque también puede estar restringida a un dominio más pequeño.

Mínimo global:

$$f(x^*) = 0, \text{at } x^* = (0, \ldots, 0)$$

Función Schaffer2

Descripción:
La función Schaffer2 se muestra en un dominio de entrada más pequeño en el segundo diagrama de la Figura 5 para mostrar detalles. Esta función es de dimensión 2.

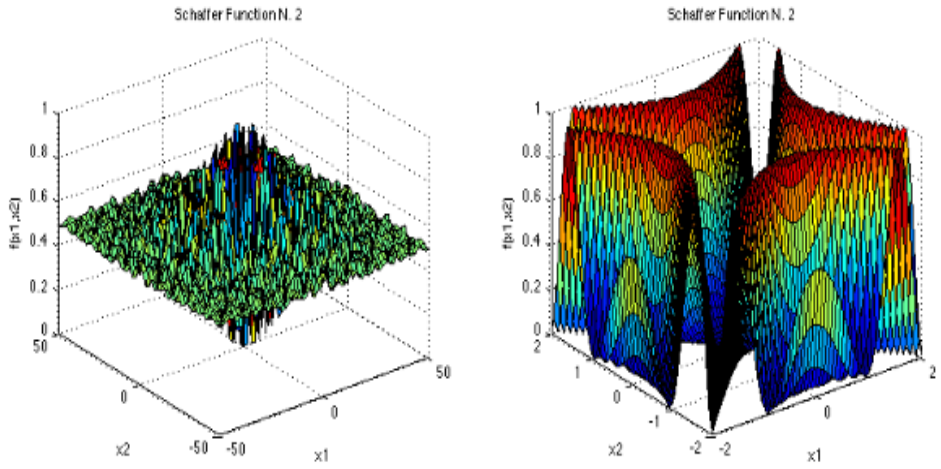

Figura 5. Gráfico de la Función Schaffer2

Función:

$$f(\mathbf{x}) = 0.5 + \frac{\sin^2(x_1^2 - x_2^2) - 0.5}{[1 + 0.001(x_1^2 + x_2^2)]^2}$$

Dominio de entrada:
La función se evalúa por lo general en el hipercubo $x_i \in$ [-100, 100], para todo i = 1, 2.

Mínimo global:

$$f(x^*) = 0, \text{at } x^* = (0,0)$$

- ✓ **Función Schwefel1.2**

Descripción:
La función Schwefel1.2 es de dimensión *d*.

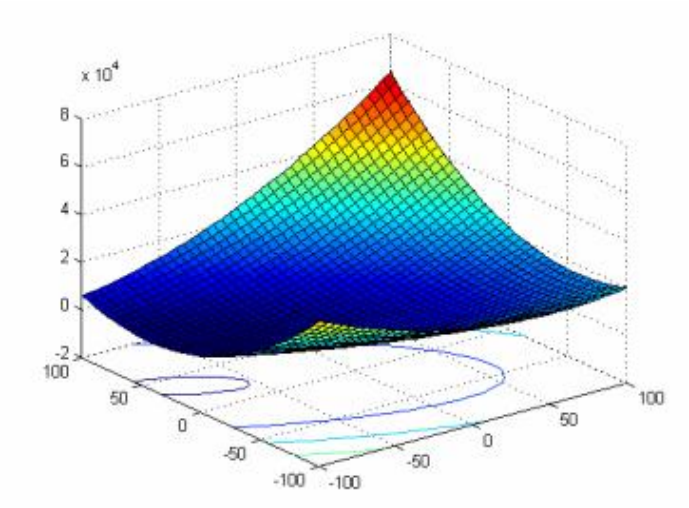

Figura 6. Gráfico de la Función Schwefel1.2

Función:

$$F_2(\mathbf{x}) = \sum_{i=1}^{D} (\sum_{j=1}^{i} z_j)^2$$

Dominio de entrada:
La función se evalúa por lo general en el hipercubo $x_i \in$ [-100, 100], para todo $i = 1, \ldots, d$.

Mínimo global:

$$f(x^*) = 0, \text{at } x^* = (0, \ldots, 0)$$

5.2.2 Funciones en forma de taza (*Bowl-Shaped*)

- ✓ **Función Suma de diferentes potencias**

Descripción:
La función suma de funciones diferentes potencias es unimodal. Esta función es de dimensión *d*. En la Figura 7 se muestra en su forma de dos dimensiones.

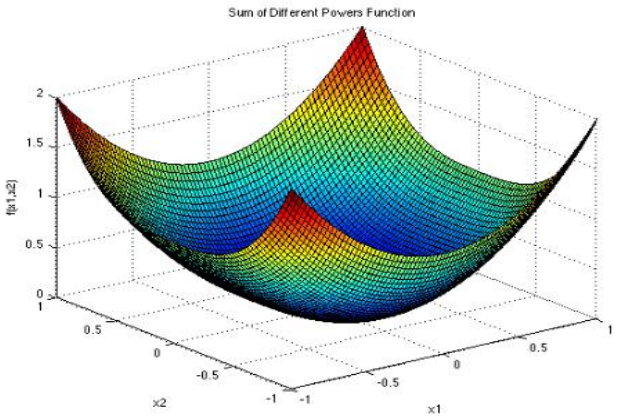

Figura 7. Gráfico de la Función Suma de diferentes potencias

Función:

$$f(\mathbf{x}) = \sum_{i=1}^{d} |x_i|^{i+1}$$

Dominio de entrada:
La función se evalúa por lo general en el hipercubo $x_i \in$ [-1, 1], para todo i = 1, ..., d.

Mínimo global:

$$f(x^*) = 0, \text{at } x^* = (0, \ldots, 0)$$

✓ **Función Esfera**

Descripción:
La función de la esfera tiene d mínimos locales a excepción de la global. Es continua, convexa y unimodal. Esta función es de dimensión d. En la Figura 8 se muestra su forma bidimensional.

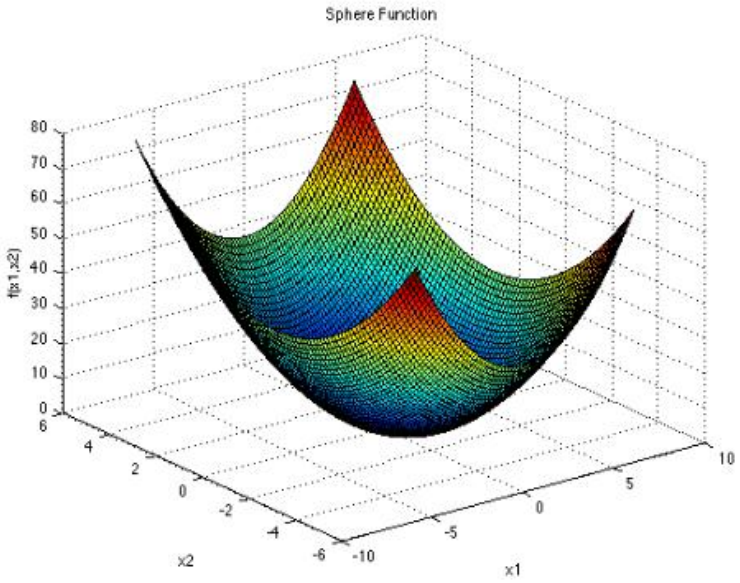

Figura 8. Gráfico de la Función Esfera

Función:

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2$$

Dominio de entrada:
La función se evalúa por lo general en el hipercubo $x_i \in$ [-5.12, 5.12], para todo i = 1, ..., d.

Mínimo global:

$$f(x^*) = 0, \text{at } x^* = (0, \ldots, 0)$$

5.2.3 Funciones con forma de plato (*Plate-Shaped*)

- ✓ **Función Booth**

Descripción:
La función Booth es de dimensión 2.

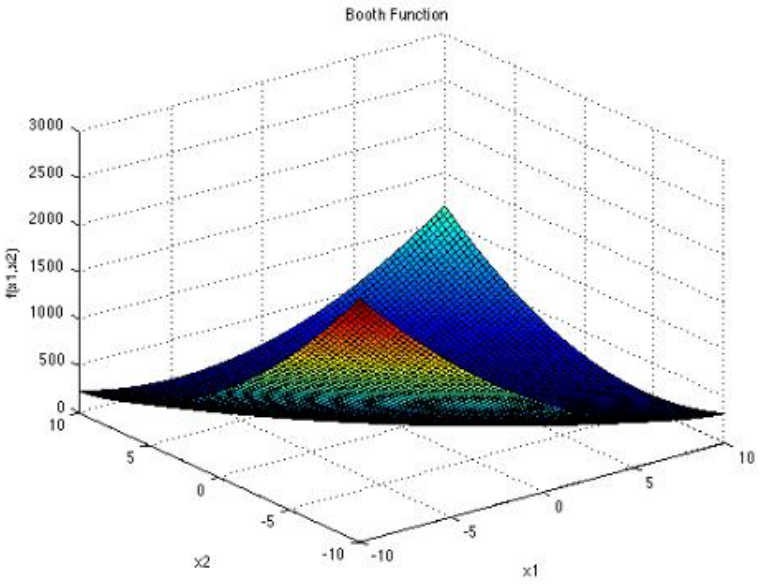

Figura 9. Gráfico de la Función Booth

Función:

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

Dominio de entrada:
La función se evalúa por lo general en el hipercubo $x_i \in$ [-10, 10], para todo i = 1, 2.

Mínimo global:

$$f(x^*) = 0, \text{at } x^* = (1,3)$$

- ✓ **Función Matyas**

Descripción:
La función Matyas no tiene los mínimos locales, excepto el global. Esta función es de dimensión 2.

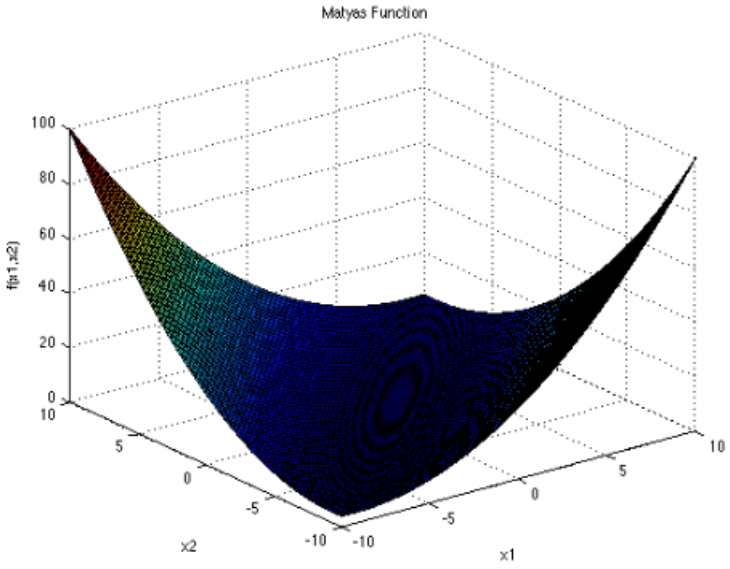

Figura 10. Gráfico de la Función Matyas

Función:

$$f(\mathbf{x}) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$$

Dominio de entrada:

La función se evalúa por lo general en el hipercubo $x_i \in$ [-10, 10], para todo i = 1, 2.

Mínimo global:

$$f(x^*) = 0, \text{at } x^* = (0,0)$$

5.2.4 Funciones con valles (*Valley-Shaped*)

- ✓ **Función Rosenbrock**

Descripción:

La función Rosenbrock, también conocida como la función Valley, es un popular problema de prueba para algoritmos de optimización basados en gradiente. Se muestra en la Figura 11 en su forma de dos dimensiones. La función es unimodal, y el mínimo global se encuentra en un valle estrecho y parabólico. Sin embargo, a pesar de que este valle es fácil de encontrar, la convergencia al mínimo es difícil. Esta función es de dimensión *d*.

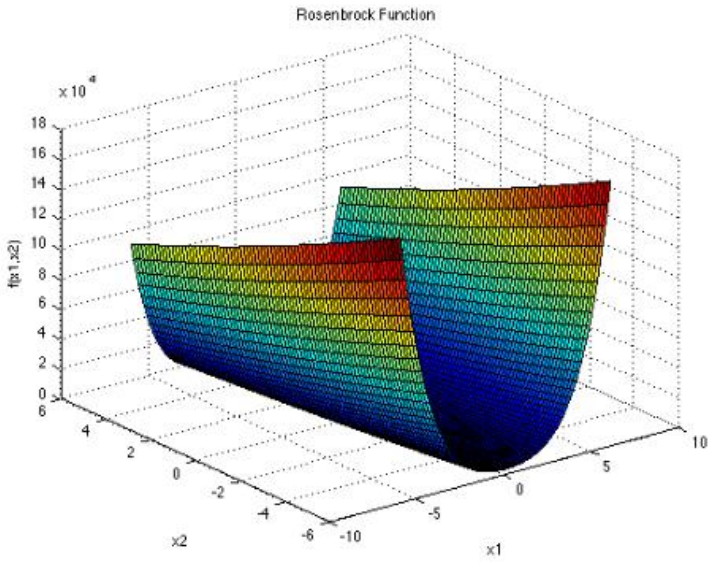

Figura 11. Gráfico de la Función Rosenbrock

Función:

$$f(\mathbf{x}) = \sum_{i=1}^{d-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

Dominio de entrada:
La función se evalúa por lo general en el hipercubo $x_i \in$ [-30, 30], para todo $i = 1, ..., d$.

Mínimo global:

$$f(x^*) = 0, \text{at } x^* = (1, \ldots, 1)$$

5.2.5 Otras

- ✓ **Función Beale**

  Descripción:
  La función de Beale es multimodal, con picos agudos en las esquinas del dominio de entrada. Esta función es de dimensión 2.

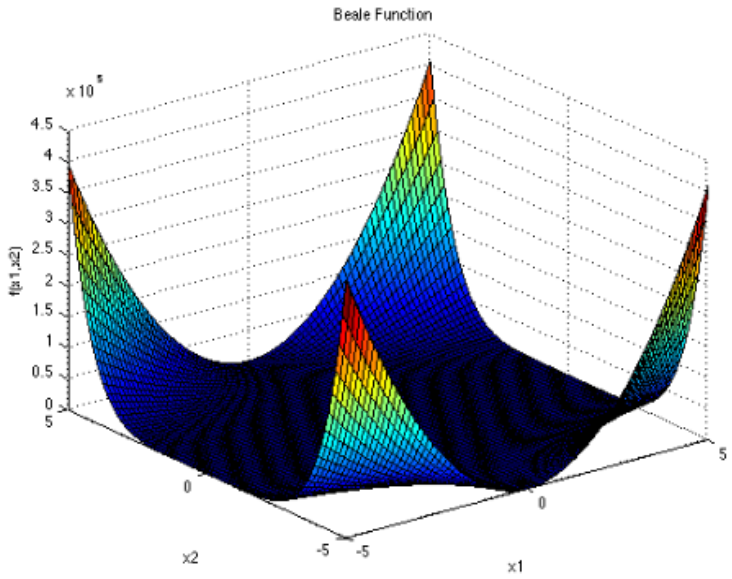

Figura 12. Gráfico de la Función Beale

Función:

$$f(\mathbf{x}) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2$$

Dominio de entrada:

La función se evalúa por lo general en el hipercubo $x_i \in$ [-4.5, 4.5], para todo i = 1, 2.

Mínimo global:

$$f(x^*) = 0, \text{at } x^* = (3,0.5)$$

- ✓ **Función Branin**

Descripción:

La función Branin, o Branin-Hoo tiene tres mínimos global. Los valores recomendados de a, b, c, r, s y t son:

a = 1
b = 5,1 / (4π2)
c = 5 / π
r = 6
s = 10
t = 1 / (8π)

Esta función es de dimensión 2.

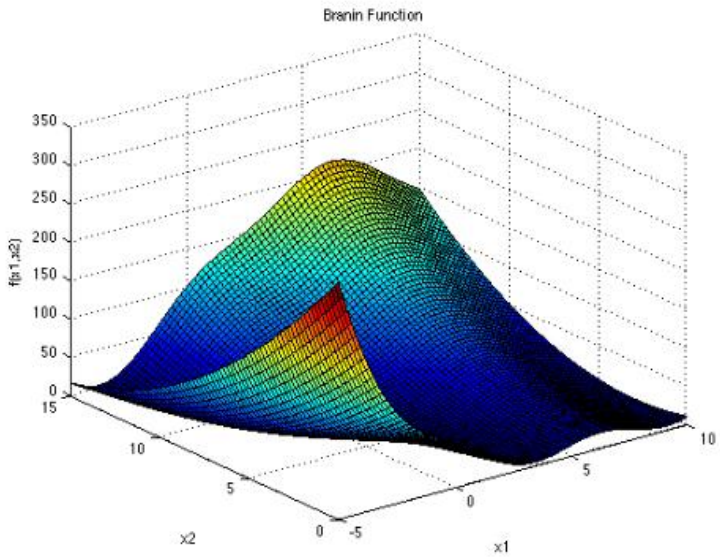

Figura 13. Gráfico de la Función Branin

Función:

$$f(\mathbf{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)\cos(x_1) + s$$

Dominio de entrada:

La función se evalúa por lo general en el hipercubo $x_1 \in$ [-5, 10], $x_2 \in$ [0, 15].

Mínimo global:

$f(x^*) = 0.3977887, \text{at } x^* = (-\pi, 12.275), (\pi, 2.275)\ and\ (9.42478, 2.475)$

Capítulo 6: Resultados

En este capítulo se presenta la experimentación realizada para abordar las funciones de optimización por medio del algoritmo ABC. Este capítulo está formado por tres partes bien diferenciadas. En la primer parte se realiza una descripción de las variables utilizadas para la experimentación. En la segunda parte se presenta la experimentación con distintas cantidades de fuentes y se hace una comparación de resultados, a fin de determinar el tamaño más adecuado de la población de abejas. En la última sección, se analiza el comportamiento del algoritmo ABC original con el ABC propuesto modificando el método de selección.

6.1 Parametrización

Inicialmente detallaremos las variables utilizadas en el presente trabajo. Como el tamaño de la colonia (SN) es importante para lograr un buen desempeño del algoritmo, se han considerado un subconjunto de posibles valores para SN = {20,50,100}. El número máximo de ciclos se fijó en 100.000 para todas las funciones. El ABC básico utilizado en este estudio emplea un parámetro de control, que se denomina límite. Una fuente de alimento no será explotada más y se supone que será abandonada cuando se exceda el límite de la fuente. Esto significa que la solución con "excede límite permitido" no se puede mejorar más. Definimos una relación para el valor límite en función de la dimensión del problema (D) y del tamaño de la colonia: $\text{limite} = \text{SN} * \text{D}$ [19]:

Como conjunto de prueba se utilizaron una serie de funciones de optimización, descriptas en el capítulo anterior. Este conjunto es lo suficientemente grande como para incluir muchos tipos diferentes de problemas como unimodal (U) y multimodal (M), separables (S) y no separable (N). En la Tabla 1 se presenta un resumen de las funciones utilizadas en la experimentación, detallando el rango inicial (RI), la dimensión (D), las características (C) y el óptimo (O).

Tabla 1. Funciones de optimización utilizadas en la experimentación 1. D: Dimensión, C: Características, U: Unimodal, M: Multimodal, S: Separable, N: No Separable y O: Optimo.

Funciones	RI	D	C	O
Esfera	[-100,100]	30	US	0,000000
Ackley	[-32.768, 32.768]	30	MN	0,000000
Rosenbrock	[-30, 30]	30	UN	0,000000
Schwefel1.2	[-100, 100]	30	UN	0,000000
Beale	[-4.5, 4.5]	2	UN	0,000000
Booth	[-10, 10]	2	MS	0,000000
Branin	[-5, 10]x[0,15]	2	MS	0,3977887
Matyas	[-10,10]	2	UN	0,000000

Schaffer2	[-100, 100]	30	MN	0,000000
Sumpow	[-1, 1]	30	MN	0,000000

El algoritmo fue programado en lenguaje C utilizando como entorno de desarrollo Dev C++, el cual fue instalado en una Toshiba Satellite L745 Las ejecuciones se realizaron en una PC con las siguientes características: AMD Phenom8450 con 3 cores a 2 GHz c/u, 2GB de RAM y sistema operativo Slackware (versión del kernel 2.6.27.7-smp).

6.2 – Análisis del tamaño de la colonia

En esta sección se compara el desempeño del algoritmo ABC en función de diferentes cantidades de fuentes. El objetivo es conocer el mejor valor de SN para resolver los problemas de optimización en cuestión. Para ello analizaremos el algoritmo bajo tres métricas: calidad de la mejor solución hallada para cada función, esfuerzo numérico realizado en el proceso de optimización y tiempo requerido tanto para encontrar la mejor solución como el tiempo total del algoritmo. Para este análisis se consideran valores de SN de 20, 50 y 100 fuentes, por ello los algoritmos los diferenciaremos como ABC_Ruleta_20, ABC_Ruleta_50 y ABC_Ruleta_100, respectivamente.

Las pruebas que se realizaron constan de 30 ejecuciones independientes de cada algoritmo y para cada función de optimización. Teniendo en cuenta que se consideran 10 funciones y 3 algoritmos, se hace un total de 900 (10×3×30) ejecuciones.

Para comenzar con el análisis nos abocaremos a considerar el promedio de las mejores soluciones obtenidas por las 3 variantes del algoritmo ABC en estudio. La Tabla 2 muestra los resultados.

Tabla 2. Promedio de la mejor solución

Función	ABC_Ruleta_20	ABC_Ruleta_50	ABC_Ruleta_100
Esfera	0,000000	0,000000	0,000000
Ackley	0,000000	0,000000	0,000000
Rosenbrock	0,317015	0,003715	0,002085
Schwefel1.2	0,000002	0,000030	0,000079
Beale	0,000037	0,000000	0,000000
Booth	0,000000	0,000000	0,000000
Branin	0,397887	0,397887	0,397887
Matyas	0,000000	0,000000	0,000000
Schaffer2	0,000000	0,000000	0,000000
Sumpow	0,000000	0,000000	0,000000

Describiendo en detalle, se observa que ABC_Ruleta_20 no encuentra el óptimo en tres funciones, mientras que ABC_Ruleta_50 y ABC_Ruleta_100 presentan dificultades en 2

funciones de las 10 analizadas. Una particularidad de estas funciones es que pertenecen a una misma categoría como lo es ser unimodales y no separables. En conclusión a lo dicho anteriormente, se indica que en 7 de las 10 funciones consideradas se pudieron encontrar el óptimo utilizando distintas cantidades de fuentes. Por esta razón, a partir de ahora los análisis de resultados que siguen sólo consideran las funciones para las cuales todos los algoritmos fueron capaces de encontrar el valor óptimo.

La Tabla 3 muestra el promedio de ciclos en las que se encuentra el valor óptimo para cada una de las funciones, es decir la velocidad de convergencia de cada uno de los algoritmos. ABC_Ruleta_100 es el algoritmo que encuentra la mejor solución de manera más rápida (menor cantidad de ciclos promedio). Es importante notar que para las funciones Booth, Branin, Schaffer2 y Sumpow la diferencia entre ciclos promedio que presentan los algoritmos ABC_Ruleta_50 y ABC_Ruleta_100 es mínima. Una situación muy similar se observa con los valores promedio de la última final de la tabla para cada uno de los algoritmos. Esto indicaría que ambas variantes no presentan diferencias de en la cantidad de evaluaciones para hallar buenas soluciones.

Tabla 3. Promedio de ciclos con distintas cantidades de fuentes

Función	**ABC_Ruleta_20**	**ABC_Ruleta_50**	**ABC_Ruleta_100**
Esfera	**41186**	42759	43220
Ackley	6859	4953	**4625**
Booth	332	**266**	267
Branin	170	141	**137**
Matyas	53844	4530	**4137**
Schaffer2	61	56	**52**
Sumpow	**86825**	99913	99929
Promedio	27039	21803	**21767**

A continuación, el análisis se enfoca en los tiempos en donde se encontró la mejor solución con las distintas cantidades de fuentes. La Figura 14 ilustra los tiempos promedios de la mejor solución en las diferentes funciones de optimización y con los diferentes algoritmos (ABC_Ruleta_20, ABC_Ruleta_50 yABC_Ruleta_100).

En las funciones Booth, Branin, Matyas y Schaffer2 los tiempos donde se obtiene el óptimo se aproximan a cero en los tres algoritmos. En cambio en la funcion Ackley hay una mínima diferencia con las funciones mencionadas anteriormente ya que los tiempos rondan entre 1 y 3 segundos.

Ademas, se puede observar que en la función Esfera los tiempos para encuentrar la mejor solución de los tres algoritmos rondan entre 4 y 11 segundos, es decir, con ABC_Ruleta_20 el tiempo fue de 4,30 segundos, en ABC_Ruleta_50 de 11,27 segundos y por último en ABC_Ruleta_100 en 24,80 segundos.

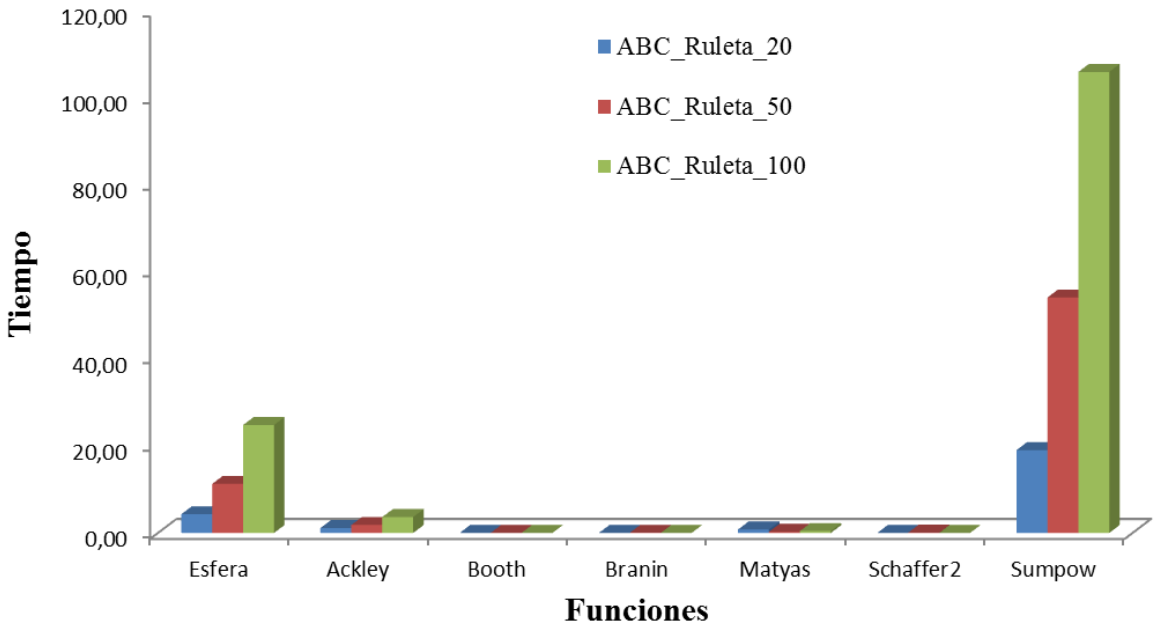

Figura 14 . Tiempo de la mejor solución con diferentes fuentes.

Por último, vemos una clara diferencia en la función Sumpow con los tiempos en donde se encuentra la mejor solución en comparación a las demás funciones. Observamos que para el algoritmo ABC_Ruleta_20 el tiempo es de 19,03 segundos, en ABC_Ruleta_50 es de 54,10 segundos (ronda en 1 minuto) y por último en ABC_Ruleta_100 el tiempo está por arriba de 105 segundos que equivale a casi 2 minutos.

Concluyendo, en función de los resultados obtenidos, se observa que ABC_Ruleta_50 y ABC_Ruleta_100 presentan calidad de resultados similares con una misma velocidad de convergencia, pero los tiempos en los que cada algoritmo encuentra esas buenas soluciones son muy distintos. Por esta razón es que de aquí en adelante optaremos por utilizar un tamaño de fuente SN=50.

6.3 – Análisis del mecanismo de selección

En esta sección se presentan los resultados obtenidos por el algoritmo ABC bajo distintos métodos de selección de las abejas observadoras, según se explicó en el capítulo 5:

- ✓ *ABC_Torneo:* algoritmo ABC con selección por torneo para seleccionar la abeja observadora x_k.
- ✓ *ABC_Ruleta:* algoritmo ABC con selección por ruleta para seleccionar la abeja observadora x_k.

Las pruebas que se realizaron constan de 30 ejecuciones independientes de cada algoritmo y para cada función de optimización. Teniendo en cuenta que se consideran 10 funciones y 2 algoritmos, se hace un total de 600 (10x2x30) ejecuciones.

Como primera observación, el algoritmo ABC_ Ruleta logró obtener el valor óptimo en 8 de las 10 funciones consideradas. Por otro lado, vale destacar, que el algoritmo ABC_Torneo logró obtener el valor óptimo para todas las funciones de optimización consideradas en este trabajo.

Tabla 4. Promedio de las mejores soluciones encontradas por las variantes ABC

Función	ABC_Torneo	ABC_Ruleta
Esfera	0,000000	0,000000
Ackley	0,000000	0,000000
Rosenbrock	0,000000	0,003715
Schwefel1.2	0,000000	0,000030
Beale	0,000000	0,000000
Booth	0,000000	0,000000
Branin	0,397887	0,397887
Matyas	0,000000	0,000000
Schaffer2	0,000000	0,000000
Sumpow	0,000000	0,000000

En la Tabla 4 podemos observar claramente que en la función Rosenbrock y en Schwefel1.2 con el algoritmo ABC_ Ruleta no se llega al valor óptimo, por lo que estas dos funciones no se incluirán en el análisis de resultados realizado en esta sección. Una situación distinta se presenta en el algoritmo ABC_Torneo, el cual obtiene el valor óptimo en la totalidad de las funciones.

Teniendo en cuenta lo detallado anteriormente decimos que la mejora propuesta en nuestro trabajo, ABC_Torneo, es adecuada ya que el algoritmo obtiene el valor óptimo en la totalidad de las funciones elegidas.

A continuación, nos abocaremos a mostrar el esfuerzo computacional de los algoritmos, es decir la cantidad de ciclos necesarias para llegar al valor óptimo, y los tiempos involucrados en dicha optimización, para aquellas funciones que han encontrado el valor óptimo.

En la Tabla 5 podemos observar las ciclos promedio en las que se encontraron las soluciones óptimas para cada una de las funciones de optimización considerando los dos algoritmos en el estudio.

Tabla 5. Iteración promedio donde se encontró la solución óptima.

Función	ABC_Torneo	ABC_Ruleta
Esfera	**33849**	42759
Ackley	**4369**	4953
Beale	**264**	691
Booth	**151**	266
Branin	**106**	141
Matyas	**2245**	4530
Schaffer2	**33**	56
Sumpow	**44154**	99913

Observando la tabla se presenta una clara diferencia entre la cantidad de ciclos necesarios por ABC_Torneo y ABC_ Ruleta. En las 8 funciones consideradas y que se llegó al óptimo, ABC_Torneo encuentra en forma más rápida los valores óptimos (menor cantidad de ciclos). Para las funciones Beale, Booth, Matyas, Schaffer2 y Sumpow la reducción que obtiene ABC_Torneo ronda aproximadamente el 50%, mientras que para las siguientes funciones: Esfera, Ackley y Branin la reducción es aproximadamente del 20%.

Un punto interesante surge al analizar el tiempo de ejecución de las variantes del algoritmo ABC en estudio al resolver las diferentes funciones de optimización. En la Figura 15 se muestra el tiempo total promedio de ejecución. En dicha figura se observa claramente que ABC_Torneo presenta en la mayoría de las funciones una ejecución más rápida que ABC_ Ruleta. Esto se debe al cálculo extra en la que se incurre al calcular los valores de probabilidad de las fuentes para aplicar la selección.

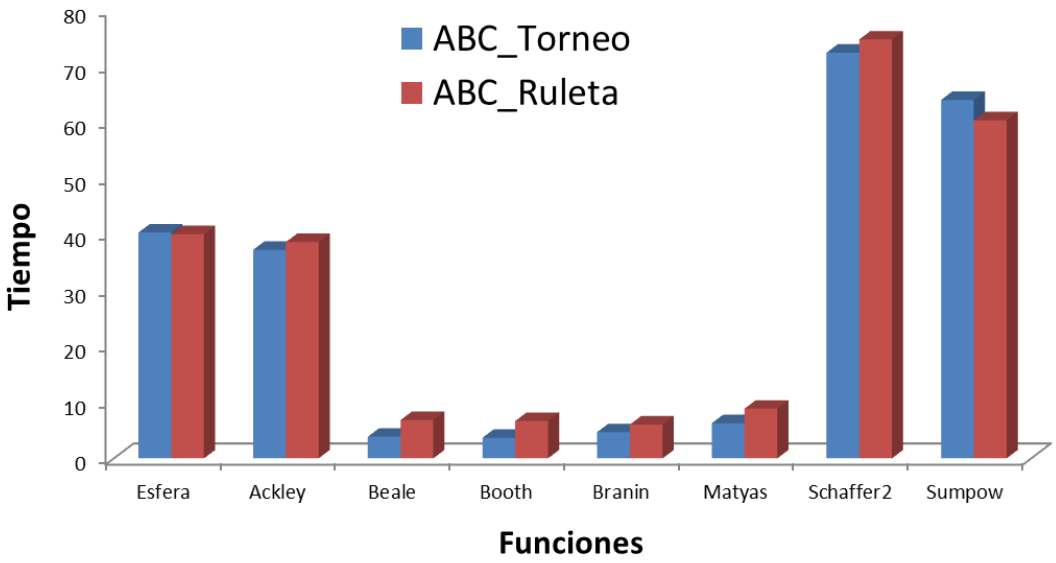

Figura 15. Tiempos promedios de ejecución

Describiendo en detalle, en la función Esfera y en la Sumpow hay una mínima diferencia en que el ABC_ Ruleta presenta tiempos de ejecución menores que ABC_Torneo. En Esfera esa diferencia es de 30 decisegundo, en cambio en Sumpow es de 3 segundos aproximadamente. Pero vale resaltar que en las funciones Ackley, Rosenbrock, Schwefel1.2, Beale, Booth, Matyas, Branin y Schaffer2 ABC_Torneo es más rápido que ABC_ Ruleta.

Otro punto importante para el análisis de los resultados obtenidos es analizar el tiempo en que se encontró la mejor solución. La Figura 16 ilustra los tiempos promedios de la mejor solución en las diferentes funciones de optimización para los algoritmos en estudio.

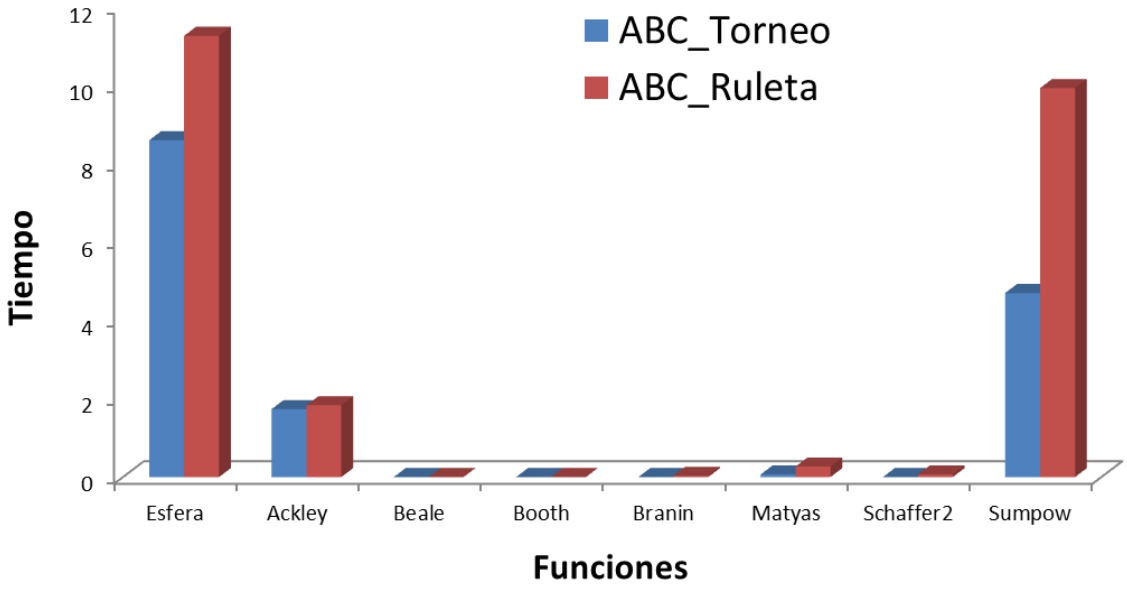

Figura 16. Tiempo promedio de la mejor solución

Un punto importante para destacar aquí es que en todas las funciones de optimización se encuentra más rápido la mejor solución con el método de selección ABC_Torneo.

En la función Esfera vemos que los tiempos para encontrar la mejor solución están por arriba de los 8 segundos. Por otro lado en la función Sumpow el tiempo de la mejor solución ronda entre 4 y 9 segundos. Por último, en la función Ackley observamos que los tiempos oscilan en 1 segundo.

Para el resto de las funciones este último análisis se utiliza la Tabla 6, dado que en la Figura 16 no se puede apreciar la diferencia en los tiempos ya que los mismos rondan los 0 segundos.

Tabla 6 Tiempo promedio de la mejor solución

Función	ABC_Torneo	ABC_Ruleta
Esfera	8,60	11,27
Ackley	1,73	1,83
Beale	0,00	0,00
Booth	0,00	6,67
Branin	0,00	0,03
Matyas	0,07	0,27
Schaffer2	0,00	0,07
Sumpow	24,23	54,10

Observamos que tanto para ABC_Torneo y ABC_ Ruleta en las funciones Beale y Booth el proceso de optimización es muy simple ya que se encuentra la mejor solución en forma muy temprana, es decir en 0 segundos. Por otra parte, se puede observar en detalle que en las funciones Matyas, Branin y Schaffer2, los tiempos de la mejor solución son muy chiquitos para los dos algoritmos considerados.

Por consiguiente, los bajos tiempos de ejecución y los buenos resultados presentados en su mayoría por ABC_Torneo, convierten a nuestra propuesta en una buena opción para resolver funciones de optimización de distintas características.

Para dar cierre a este trabajo, decimos que de los dos métodos de selección que se utilizaron por las abejas observadoras, dieron resultados positivos, especialmente el método ABC_Torneo. Este último ha demostrado ser más adecuado que ABC_ Ruleta, obteniendo buena calidad de resultados con tiempos computacionales menores.

Capítulo 7: Conclusiones y trabajo futuro

A continuación se presentan las conclusiones a las que se llegaron a partir del experimentación y resultados obtenidos, además del trabajo futuro que se deriva de esta investigación.

7.1 – Conclusiones

En este trabajo de tesis se presentó una introducción al paradigma de inteligencia colectiva, y en particular a los algoritmos de colonia de abejas artificial (ABC). Se presentó una descripción detallada de cada uno de sus componentes, es decir las actividades específicas que desarrolla cada una de las abejas artificiales dentro de la colonia, para resolver funciones de optimización.

Uno de los parámetros importantes del algoritmo es el número de fuentes de alimento (SN), el cual afecta directamente el desempeño del algoritmo. En caso de que esta cantidad sea insuficiente, el algoritmo tiene pocas posibilidades de recorrer el espacio en busca de buenas fuentes de alimento. Por otro lado, si el número de fuentes es demasiado grande, el algoritmo será excesivamente lento. Se realizaron pruebas para observar el desempeño del algoritmo ABC con diferentes cantidades de fuentes y de esta manera conocer el mejor valor de SN para resolver los problemas de optimización planteados. Los valores de SN que se consideraron fueron de 20, 50 y 100, abarcando un rango de pocas a muchas fuentes de alimento. Utilizar un tamaño pequeño de SN (algoritmo ABC_Ruleta_20) hace que el algoritmo encuentre el óptimo para una menor cantidad de funciones de optimización que en el caso de considerar una mayor cantidad de fuentes de alimentos (algoritmos ABC_Ruleta_50 y ABC_Ruleta_100). Al analizar los tiempos de ejecución, el incrementar la cantidad de fuentes produce un aumento en los tiempos de ejecución, situación esperable debido a que el algoritmo debe realizar más operaciones y evaluaciones en cada ciclo. Por lo tanto, el mejor compromiso entre calidad de soluciones y tiempo de ejecución se observa con un valor de SN=50, cantidad que se utiliza en el restante estudio.

Además, se propuso una variación al algoritmo ABC tradicional relacionado el método de selección de las fuentes de alimento por parte de las abejas observadoras. En el algoritmo ABC tradicional el método de selección es proporcional a la cantidad de néctar de las fuentes de alimento (algoritmo denominado ABC_Ruleta). La variante propuesta hace uso de la selección por torneo binario, resultando el algoritmo ABC_Torneo. Los resultados obtenidos por ABC_Torneo fueron prometedores, dado que en la totalidad de las funciones de optimización propuestas se llegó al valor óptimo correspondiente, en contraposición al ABC_Ruleta que en 3 funciones no logra obtener ese valor. Además, ABC_Torneo encontró en forma más rápida los valores óptimos (menor cantidad de ciclos) que ABC_Ruleta. Finalmente, debido a la simplicidad del método de selección por torneo, los tiempos totales de ejecución de ABC_Torneo son inferiores al algoritmo tradicional. De lo

anterior se desprende la superioridad del algoritmo propuesto ABC_Torneo respecto del algoritmo tradicional.

7.2 – Trabajos futuros

Si algo destaca a la ciencia es que esta no se detiene, es por esto que derivada de esta investigación se observan como trabajos futuros los siguientes:

- ✓ Considerar otros métodos de selección en el proceso de elección de las abejas observadoras (por rango, muestreo estocástico universal, entre otros)
- ✓ Estudiar los efectos de utilizar distintas proporciones de abejas empleadas con respecto al número de abejas observadores, buscando reducir la convergencia del algoritmo.
- ✓ Agregar más cantidades de funciones de optimización para que se puedan resolver con los dos métodos de selección propuestos en este trabajo.

Bibliografía

[1] J. García Nieto y E. Alba Torres, “Algoritmos Basados en Inteligencia Colectiva para la Resolución de Problemas de Bioinformática y Telecomunicaciones”, Universidad de Málaga, Septiembre 2007.

[2] E. Mezura-Montes, B. Hernández-Ocaña y O. Cetina-Dominguez, "Nuevas Heurísticas Inspiradas en la Naturaleza para Optimización Numérica (Mecatrónica)", Editorial IPN, 2004.

[3] C. A. Mezura-Montes y E. Coello, "Constrained Optimization via MultiobjectiveEvolutionary Algorithms". In J. Knowles, D. Corne, and K. Deb, Multiobjective, 2008.

[4] E. Bonabeau, M. Dorigo y G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems., Oxford University Press, 1999.

[5] Eberhart, J. Kennedy y Russell, “ Particle Swarm Optimization”, 1995.

[6] Z. X. Yang y W. Zhang, Solving numerical optimization problems by simulating particulates in potential field with cooperative agents. International Conference on Artificial Intelligence, 2002.

[7] Gómez y J. A. Franco, “Un algoritmo basado en la optimización por enjambre de partículas para el problema de asignación axial 3-dimensional”, La Paz, Baja California Sur, México, Septiembre del 2011.

[8] M. Dorigo, “Ant Colony Optimization: A New Meta-Heuristic”, 1992.

[9] J. García Nieto y E. Alba Torres, “Algoritmos Basados en Inteligencia Colectiva para la Resolución de Problemas de Bioinformática y Telecomunicaciones”, Universidad de Málaga, Septiembre 2007.

[10] E. Téllez Enríquez, “Uso de una Colonia de Hormigas para resolver Problemas de Programación de Horarios”, Tesis de Maestría en Ciencias de la Computación, LANIA Xalapa, Enero 2007.

[11] . M. J. Beckmann y T. C. Koopmans, "Assignment problems and the location of economic activities", Econometrica, 25(1):53–76, 1957.

[12] P. Gondim., "Genetic Algorithms and the Location Area Partitioning Problem in Cellular Networks", In Proceedings of IEEE the 46th Vehicular Technology Conference, pages 18351841, 1996.

[13] D. Karaboga, B. Akay y C. Ozturk, “Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks”, Springer.Verlag, Berlin Heidelberg: 318-329, 2007.

[14] A. Baykasoglu, L. Ozbakir y P. Tapkan, "Artificial bee colony algorithm and its

application to generalized assignment problem", Vienna, Austria: In F. T. Chan and M. K. Tiwari, editors, Swarm Intelligence: Focus on Ant and Particle Swarm Optimization, pages 113–144. Itech Education and Pub., 2007.

[15] D. Karaboga y B. Basturk, "On the performance of artificial bee colony (ABC) algorithm", Applied Soft Computing 8: 687–697, 2008.

[16] D. Karaboga, "an idea based on honey bee swarm for numerical optimization", technical report-tr06, Octubre, 2005.

[17] B. Basturk y D. Karaboga, "A powerful and efficient algorithm for numerical function optimization: articial bee colony (abc) algorithm", Journal of Global Optimization, 39(3):459471, 2007.

[18] "Optinization Test Functions and Datasets" URL: http://www.sfu.ca/~ssurjano/optimization.html, Fecha de ultima visita: 10 de Agosto 2016.

[19] D. Karaboga y B. Akay, «A comparative study of Artificial Bee Colony algorithm,» *Applied Mathematics and Computation* , pp. 108-132, 2009.