Octubre
del 2017

Memorias de la **Escuela de** Cómputo Evolutivo

CIMAT, Guanajuato, Gto.

Editores

Carlos Segura González
Salvador Botello Rionda
Arturo Hernández Aguirre
Sergio Ivvan Valdez Peña
Mario Alberto Ibarra Manzano
Juan Gabriel Aviña Cervantes
Ivan Cruz Aceves
Fernando Cervantes Sánchez

ISBN: 978-84-947311-9-8

Prólogo

La escuela de cómputo evolutivo del CIMAT surge con el afán de fomentar el interés existente en México por el campo del cómputo evolutivo, y de manera más general, por el campo de optimización con métodos estocásticos. En el año 2016 se dio inicio a la Primera Escuela de Cómputo Evolutivo que se ha celebrado en CIMAT. Por ser su primera edición, se decidió realizar un evento pequeño, de dos días, que tuvo una muy buena aceptación cubriéndose todas las plazas reservadas. A raíz de esto, se decidió dar continuidad a este evento y que este creciera, por lo que en esta segunda edición se incrementó la duración de la escuela, se incluyó una sesión de pósteres para que hubiera más interacción entre los participantes, y se tomó la decisión de publicar un libro en el que de manera didáctica se expusieran los principales temas tratados en la escuela, así como la información de los pósteres presentados. El objetivo de este libro, no es tanto profundizar en los diferentes temas, sino introducir diferentes tópicos, así como presentar de forma resumida algunos de los trabajos que están desarrollando muchos tesistas que están trabajando en este campo a lo largo de todo México. Con esto esperamos atraer a más estudiantes y fomentar las colaboraciones, pues siempre es bueno que se difundan los diferentes trabajos que se están realizando y los temas en los que se está investigando. El libro está organizado en dos partes. La primera parte está constituida por un conjunto de capítulos escritos por los ponentes de la escuela. Se trata de capítulos introductorios, orientados a quienes se inician en este apasionante campo. La segunda parte está constituida por capítulos cuyos autores principales son los estudiantes que presentaron sus pósteres en la Escuela, durante la cual recibieron retroalimentación para poder mejorar y hacer más accesibles sus trabajos. Se realizó un esfuerzo por revisar todos estos capítulos y se seleccionaron sólo aquellos cuyos revisores dieron la recomendación oportuna para su publicación. Dada la gran aceptación de la Escuela, y el nivel de los ponentes que se ha conseguido atraer a la misma, esperamos seguir organizando está escuela (la edición 2018 ya está en camino), y seguir fomentando que se realice investigación y trabajos de calidad.

Dr. Carlos Segura
Coordinador del Comité Organizador de la Escuela de Cómputo Evolutivo 2017

Índice general

Parte 1 – Capítulos de los Expositores

Parte 2 – Pósteres de la Escuela de Cómputo Evolutivo

Cuckoo Search y su implementación práctica

JORGE MARIO CRUZ–DUARTE, DICIS, Universidad de Guanajuato (UG), México
IGNACIO MARTÍN–DÍAZ, DICIS, Universidad de Guanajuato (UG), México
IVÁN CRUZ–ACEVES, CONACYT, Centro de Investigación en Matemáticas (CIMAT), A.C., México

Resumen En la actualidad existe una amplia cantidad de algoritmos inspirados en la naturaleza para resolver todo tipo de problemas, algunos de ellos se enfocan en los bien conocidos problemas de optimización. Entre éstos se pueden destacar los métodos de enjambre de partículas, algoritmos genéticos, recocido simulado y evolución diferencial. La lista es extensa y serían necesarias más páginas para llegar a mencionarlos todos. Sin embargo, en este artículo se revisa la, relativamente reciente, técnica metaheurística conocida como *Cuckoo Search* (CS). Para ello se hace una breve introducción al método, luego se detalla su procedimiento y, posteriormente, se describe su implementación incorporando algunas pruebas preliminares.

1. INTRODUCCIÓN

Cuckoo Search (CS) fue propuesto por Xin–She Yang y Suash Deb en el año 2009 [9], quienes además son conocidos por otros métodos como *Firefly Algorithm* [10], *Bat Algorithm* [11], y *Flower Pollination Algorithm* [13]. CS es un algoritmo metaheurístico inspirado en el comportamiento de más de 100 especies de cucúlidos (familia: *Cuculidae* y subfamilia: *Cuculinae*). Los cucos se pueden encontrar casi en cualquier lugar del mundo con diferentes nombres locales, por ejemplo, en México se les llaman correcaminos, cuclillos, garrapateros, entre otros [2]. Particularmente, estas aves se caracterizan por emplear como estrategia de supervivencia el parasitismo de puesta (o de nido), que consiste en colocar a su descendencia en nidos ajenos para que sean empollados y criados por otras aves [3]. Para ello, las hembras de cuco han desarrollado como mecanismo biológico la puesta de 16 a 22 huevos de diferentes colores, con el objetivo de engañar a la madre huésped. Además de este esquema de supervivencia, CS incluye una estrategia de búsqueda basada en una caminata aleatoria descrita por la distribución de Lévy, también conocida como vuelo de Lévy (o *Lévy's flight*) [8]. Este comportamiento ha sido observado en diversos seres vivos como microorganismos, abejas, moscas, albatros, tiburones y humanos [6].

2. DESCRIPCIÓN DEL MÉTODO

El proceso de búsqueda de CS tiene como objetivo encontrar el mejor nido, relacionado con la mejor solución de un problema de optimización. Para ello, se identifican aleatoriamente un número definido de nidos dentro de la región factible, en donde cada uno representa un candidato a la solución del problema. Seguidamente, los cucos esconden sus huevos en los nidos, siendo cada huevo un elemento para medir la calidad de la solución. Si un huevo posee una excelente calidad (logra engañar al ave huésped), éste eclosionará y sobrevivirá para ser parte de la siguiente generación de cucos. Si lo anterior no sucede, el huevo perecerá o el nido será abandonado. Así, los nidos de baja calidad serán reemplazados por otros en cada nueva generación. Lo anterior se puede resumir en tres reglas simples:

1. Cada cuco elige aleatoriamente un nido para depositar uno o varios huevos.
2. Los huevos mejor camuflados en un nido serán parte de la nueva generación de cucos.

Autores: Jorge Mario Cruz–Duarte, DICIS, Universidad de Guanajuato (UG), Salamanca, Guanajuato, 36885, México, jorge.cruz@ugto.mx; Ignacio Martín–Díaz, DICIS, Universidad de Guanajuato (UG), Salamanca, Guanajuato, 36885, México, i.martindiaz@ugto.mx; Iván Cruz–Aceves, CONACYT, Centro de Investigación en Matemáticas (CIMAT), A.C., Guanajuato, Guanajuato, 36000, México, ivan.cruz@cimat.mx.

3. Las aves huéspedes descubren a los demás huevos con una probabilidad p_D dada.

Por simplicidad, la probabilidad p_D se asume como la porción de nidos a ser reemplazados, luego $p_D = N_A/N_T$, siendo N_A la cantidad de nidos abandonados y N_T el total de nidos.

Ahora bien, se procede a describir detalladamente los pasos del método *Cuckoo Search*, necesarios para especificar su algoritmo y realizar su implementación.

2.1. Definición del problema

Un problema de optimización se define comúnmente de la siguiente manera:

$$\vec{x}_* = \arg\min_{\vec{x}\in\Omega} \{f(\vec{x})\}, \tag{1}$$

en donde se dice que $\vec{x}_*$ minimiza a $f(\vec{x})$, siendo $f(\vec{x}) : \mathbb{R}^D \to \mathbb{R}$ la función objetivo (o de costo en problemas de minimización), y $\Omega \subseteq \mathbb{R}^D$ la región factible. En los problemas sencillos Ω se define por restricciones simples, como:

$$\Omega = \{\vec{x} \in \mathbb{R}^D : \vec{x}_{inf} \preceq \vec{x} \preceq \vec{x}_{sup}\} \tag{2}$$

siendo $\vec{x}_{inf}$ y $\vec{x}_{sup}$ respectivamente los límites inferiores y superiores de las restricciones. Con esta información es posible implementar cualquier algoritmo de optimización, en este caso, un método iterativo como lo es *Cuckoo Search.* Adicionalmente, estos métodos requieren de, por lo menos, un criterio de parada, como lo es el número máximo de generaciones (M).

2.2. Inicialización de las variables

Una vez que es definido el problema, se procede a identificar el primer conjunto de N nidos (o de candidatos a solución), definido como $\mathfrak{X}^0 = \{\vec{x}_1^0, \vec{x}_2^0, \ldots, \vec{x}_N^0\}$. La manera más sencilla de realizar esto consiste en distribuir los puntos ($\vec{x}_k^0$) aleatoriamente dentro del espacio de búsqueda Ω, de acuerdo con:

$$\vec{x}_k^0 = \vec{x}_{inf} + \vec{u} \circ (\vec{x}_{sup} - \vec{x}_{inf}), \tag{3}$$

donde $k \in \{1, 2, \ldots, N\}$, $\vec{x}_{inf}$ y $\vec{x}_{sup}$ son las fronteras inferiores y superiores de Ω, y $\vec{u}$ es un vector de elementos aleatorios e independientes con distribución uniforme entre cero y uno. El operador $\circ$ es la multiplicación vectorial elemento–a–elemento.

Además de inicializar los agentes de búsqueda, se deben definir los parámetros propios del método como: la probabilidad de que un huevo sea descubierto (p_D), el número de nidos por generación (N), y el número de generaciones de cucos (M).

2.3. Identificación de la mejor solución

Para cualquier generación t, el conjunto de huevos escondidos en los nidos ($\mathfrak{X}^t$) es evaluado en la función objetivo $f(\vec{x})$, $f(\mathfrak{X}^t) = \{f(\vec{x}_1^t), f(\vec{x}_2^t), \ldots, f(\vec{x}_N^t)\}$, con lo que es posible identificar la mejor solución ($\vec{x}_*^t$) de la generación, como,

$$\vec{x}_*^t = \arg\min_{\vec{x}_k^t}\{f(\mathfrak{X}^t)\} \quad : \quad f(\vec{x}_k^t) \leq f(\vec{x}_j^t), \quad j \in \{1, 2, \ldots N\}\backslash k. \tag{4}$$

En otras palabras, $\vec{x}_*^t$ corresponde a la posición que evaluada en $f(\vec{x})$ entrega el menor valor, comparado con los valores generados por las otras $N - 1$ posiciones.

2.4. Determinación de la nueva generación

Para hallar la nueva generación de cucos, $\mathfrak{X}^{t+1}$, cada nuevo candidato $\vec{x}_k^{t+1} \in \mathfrak{X}^{t+1}$ debe cumplir que $f(\vec{x}_k^{t+1}) \leq f(\vec{x}_k^t)$. El conjunto de nuevos candidatos se determina con las estrategias propias

del algoritmo *Cuckoo Search*, estas son, el vuelo de Lévy y la probabilidad de supervivencia. Estas estrategias se detallan a continuación.

2.5. Estrategia 1: El vuelo de Lévy

Para un agente cualquiera k, su nueva posición $\vec{x}_k^{t+1}$ se obtiene mediante

$$\vec{x}_k^{t+1} = \vec{x}_k^t + \delta_x \vec{\lambda}_L \circ (\vec{x}_k^t - \vec{x}_*^t), \tag{5}$$

donde δ_x es el tamaño del paso, comúnmente definido como 0,1 para problemas con $D \leq 3$, y 0,01 para $D > 3$. $\vec{\lambda}_L$ es un vector de números aleatorios independientes con distribución de Lévy simétrica y estable [8]. Las componentes de este vector, $\vec{\lambda}_L = (\lambda_{L,1}, \ldots, \lambda_{L,D})$, se calculan mediante el algoritmo de Mantegna [5] como,

$$\lambda_{L,d} = z|y|^{-1/\beta}, \tag{6}$$

siendo β un parámetro propio de la distribución ($1 \leq \beta < 2$), z e y variables aleatorias con distribución normal $\mathcal{N}(0, \sigma_z)$ y $\mathcal{N}(0, 1)$, respectivamente. El parámetro σ_z se obtiene con

$$\sigma_z = \left[\frac{\Gamma(\beta+1)\sin(\pi\beta/2)}{\beta\Gamma((\beta+1)/2)\, 2^{(\beta-1)/2}} \right]^{1/\beta}, \tag{7}$$

donde $\Gamma(\cdot)$ es la función Gamma.

2.6. Estrategia 2: Supervivencia

Para un agente cualquiera k, con una probabilidad p_D, su nueva posición $\vec{x}_k^{t+1}$ se obtiene mediante,

$$\vec{x}_k^{t+1} = \vec{x}_k^{t+1} + \vec{u} \circ \mathfrak{H}(\vec{u} - p_D) \circ (\vec{x}_i^{t+1} - \vec{x}_j^{t+1}), \tag{8}$$

donde $\vec{u} \in \mathbb{R}^D$ es un vector de elementos aleatorios e independientes con distribución uniforme entre cero y uno, $\mathfrak{H}$ es la forma multidimensional de la función escalón unitario, y $\vec{x}_i^{t+1}, \vec{x}_j^{t+1} \in \mathfrak{X}^{t+1}$ con $i \neq j$, donde i y j son seleccionados aleatoriamente.

2.7. Verificación de las restricciones simples

En la versión original de *Cuckoo Search* no se describe un mecanismo específico para verificar y corregir a los nidos que se posicionan por fuera del espacio de búsqueda. Si bien existen muchas maneras de tratar este problema [12], en este documento se describe una solución sencilla para esta situación. Sin embargo, para satisfacer las necesidades de un problema de optimización en particular, es recomendable revisar y ajustar esta metodología.

Ahora bien, tras encontrar una nueva generación de cucos, se procede a verificar si cada nido, $\vec{x}_k^t = (x_{k,1}^t, x_{k,2}^t, \ldots, x_{k,D}^t)^\intercal$, se encuentra en el espacio de búsqueda Ω, a través de las restricciones simples definidas en (2) de la Sección 2.1. En caso de que un nido se encuentre por fuera, éste será reintegrado a, por ejemplo, la frontera más próxima ($\vec{x}_{inf}$ ó $\vec{x}_{sup}$) como:

$$x_{k,d}^t = \begin{cases} x_{inf,d} & \text{si} \quad x_{k,d}^t \leq x_{inf,d}, \\ x_{sup,d} & \text{si} \quad x_{k,d}^t \geq x_{sup,d}, \\ x_{k,d}^t & \text{si} \quad x_{inf,d} < x_{k,d}^t < x_{sup,d}, \end{cases} \qquad \forall \quad d = 1, 2, \ldots, D. \tag{9}$$

Se debe observar que (9) corresponde a la posición de cada componente d del k–ésimo nido en la generación t, dentro de la región de búsqueda permitida.

2.8. Algoritmo de Cuckoo Search

El Algoritmo 1 muestra el proceso de búsqueda de CS con los pasos descritos anteriormente.

Algoritmo 1: Cuckoo Search (CS)

Entrada: La función objetivo $f(\vec{x})$; el espacio de búsqueda Ω; los parámetros $N > 2$, $\delta_x > 0$, $\beta \in [1; 2)$, y $p_D \in [0; 1]$; y el criterio de parada $M \gg 1$, (Secciones 2.1 y 2.2)
Salida: La mejor solución $\vec{x}_*^t$ hallada en la última generación t (Sección 2.3)
Hacer $t \leftarrow 0$, Inicializar $\mathfrak{X}^0$ en Ω (Sección 2.2) y Encontrar $\vec{x}_*^0$ de $\mathfrak{X}^0$ (Sección 2.3)
mientras *($t \leq M$) & (otro criterio de parada no se satisface)* **hacer**
  Calcular $\mathfrak{X}^{t+1}$ de acuerdo con el vuelo de Lévy (Sección 2.5)
  Verificar las restricciones con $\mathfrak{X}^{t+1}$ (Sección 2.7) y Actualizar $\mathfrak{X}^{t+1}$ (Sección 2.4)
  Calcular $\mathfrak{X}^{t+1}$ de acuerdo con la supervivencia (Sección 2.6)
  Verificar las restricciones (Sección 2.7) y Actualizar $\mathfrak{X}^{t+1}$ (Sección 2.4)
  Encontrar $\vec{x}_*^{t+1}$ de $\mathfrak{X}^{t+1}$ (Sección 2.3) y Hacer $t \leftarrow t + 1$
fin

3. IMPLEMENTACIÓN

El método de optimización de *Cuckoo Search* puede ser implementado en cualquier lenguaje de programación, al igual que cualquier otro método similar. La única razón aparente para elegir una plataforma de cómputo sobre otra, yace en la naturaleza de la aplicación práctica en la que este método será implementado. Por tanto, en la Fig. 1 se describen los pasos del algoritmo de CS usando la representación de diagrama de bloques. Esta estructura es, en cierto modo, modular y se propone para fines pedagógicos, brindándole al lector las herramientas básicas para modificar el método y ajustarlo a las necesidades de su aplicación en particular. Cada uno de los bloques se relaciona con los procedimientos descritos en la Sección 2 y el Algoritmo 1. Además, en este artículo se presenta la implementación de CS en el lenguaje de programación científica GNU Octave v4.2.1, disponible libremente en www.gnu.org/software/octave/. Por ello, los bloques en la Fig. 1 son escritos independientemente en Octave, como se muestra en el Apéndice. Adicionalmente, para ejecutar correctamente el método de *Cuckoo Search* de acuerdo con la Fig. 1, se debe tener en cuenta que:

- La función objetivo a resolver debe ser un bloque, en donde la entrada es un candidato a solución y la salida corresponde a la función evaluada en ese punto.
- El espacio de búsqueda dado por las restricciones simples debe estar en una matriz de $D \times 2$, esto es, las dimensiones como filas y los límites por dimensión como columnas.

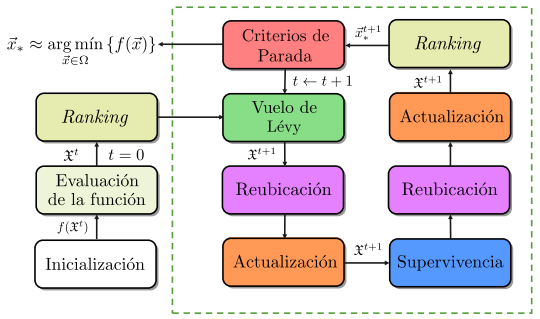


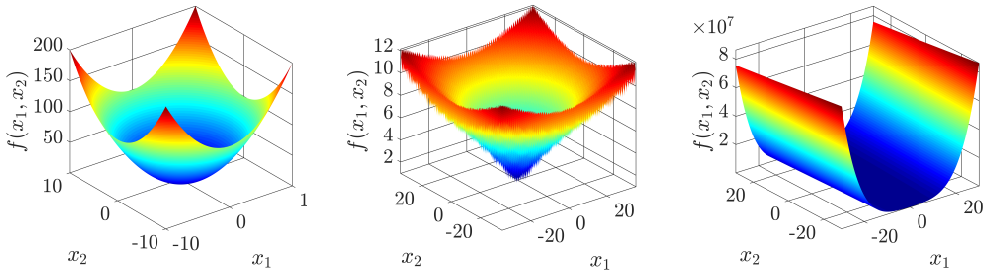
Fig. 1. Propuesta de *Cuckoo Search* como diagrama de bloques.

4. PRUEBAS PRELIMINARES

Una vez se codifica el método de optimización *Cuckoo Search*, es pertinente realizar algunas pruebas preliminares para verificar su correcto funcionamiento. Estas pruebas suelen ser sencillas desde el punto de vista numérico, sin embargo son necesarias para proceder con pruebas más rigurosas, como las realizadas para el ajuste de los parámetros del método en una aplicación específica. En consecuencia, se deben seleccionar problemas de optimización comunes en la literatura, con soluciones y comportamientos bien conocidos. Como ejemplo ilustrativo, la Tabla 1 presenta tres funciones de prueba comunes, junto con la información básica para su implementación. Además, la Fig. 2 muestra la representación de estas funciones.

Tabla 1. Funciones de prueba comunes para el estudio preliminar de la implementación de un método de optimización [4]. Todas las funciones presentadas poseen un mínimo global igual a cero, $f_k(\vec{x}_*) = 0$.

k	Nombre	Dominio	$\vec{x}_*^\intercal$	Fórmula matemática, $f_k(\vec{x})$
1	Sphere	$[-10, 10]$	$(0, \ldots, 0)$	$f_1(\vec{x}) = \|\|\vec{x}\|\|^2$
2	Ackley	$[-35, 35]$	$(0, \ldots, 0)$	$f_2(\vec{x}) = 20\left(1 - e^{-0{,}02\|\|\vec{x}\|\|/\sqrt{D}}\right) + e - e^{\|\|\cos(2\pi\vec{x})\|\|^2/D}$
3	Rosenbrock	$[-30, 30]$	$(1, \ldots, 1)$	$f_3(\vec{x}) = \sum_{d=1}^{D-1}\left[100(x_{d+1} - x_d^2)^2 + (x_d - 1)^2\right]$



(a) Sphere (b) Ackley (c) Rosenbrock

Fig. 2. Representación tridimensional de las funciones de prueba descritas en la Tabla 1.

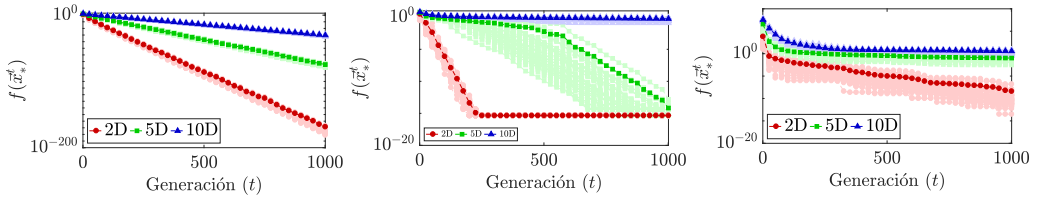
Los valores de los parámetros de CS usados en todas las ejecuciones presentadas en este documento fueron: $N = 25$, $M = 1000$, $p_D = 0{,}25$, $\delta_x = 1$ y $\beta = 1{,}5$; seleccionados de acuerdo con [9, 12, 15]. La primera prueba realizada consistió en una ejecución simple del método, para verificar que éste no presentase errores en su codificación. Para ello se seleccionó la función Sphere de la Tabla 1, considerada tradicionalmente como el problema de optimización más sencillo a resolver, y se ejecutó la secuencia de comandos mostrada en el Código 1 en la terminal de Octave. Como resultado de la implementación con $D = 2$, se obtuvo que $\vec{x}_*^M \approx$ (1e-89, 1e-89)$^\intercal$ y $f(\vec{x}_*^M) \approx$ 1e-178, siendo fácil de reconocer que es una aproximación sustancialmente cercana al valor esperado (ver Tabla 1). Así, se puede concluir parcialmente, que la implementación del método funciona correctamente.

Código 1. Ejemplo de ejecución

```
>> f1 = @(x) sum(x.^2); % Definir la función Sphere, f1(x), en la Tabla 1
>> D = 2; % Definir el número de dimensiones del espacio de búsqueda, D
>> bnd = ones(D,1)*[-10 10]; % Definir el espacio de búsqueda, Ω, como en (2)
>> [BestNest,fBest]=CuckooSearch(f1,bnd); % Ejecutar el método de Cuckoo Search para minimizar la función
```

Seguidamente, se abordaron las tres funciones propuestas en la Tabla 1, variando las dimensiones en los espacios de búsqueda, $D = 2, 5$ y 10, y resolviendo cada combinación 50 veces, con fines

estadísticos. La Fig. 3 muestra la evolución promedio de CS por generación para las pruebas realizadas con las funciones, en donde se observa que las aproximaciones se alejan del valor teórico con el incremento de las dimensiones. Este efecto se hace más notorio en las funciones Ackley y Rosenbrock (Fig. 3b y 3c), que contienen mínimos locales y una región plana, respectivamente. Además, los resultados de CS presentaron mayor variación con la función Rosenbrock, comparados con las variaciones dadas por las funciones Sphere y Ackley. La Tabla 2 presenta los resultados que complementan el análisis anterior. Finalmente, varios estudios de la literatura han reportado que CS es más eficiente que los métodos *Genetic Algorithms* (GAs) y *Particle Swarm Optimisation* (PSO), en problemas de optimización no lineal [3, 8, 14].



(a) Sphere (b) Ackley (c) Rosenbrock

Fig. 3. Evolución promedio de la solución por generación para las funciones de prueba, con 2, 5 y 10 dimensiones. Los marcadores tenues representan las 50 repeticiones de cada ejecución.

Tabla 2. Resultados obtenidos en las pruebas preliminares con las funciones Sphere, Ackley y Rosenbrock, variando las dimensiones (D = 2, 5 y 10) y repitiendo cada ejecución del método 50 veces.

Función	D	Valores de la función objetivo $f(\vec{x}_*^M)$			
		Promedio	Desviación Estándar	Mejor	Peor
Sphere	2	6,2e-169	0	4,3e-179	2,6e-167
	5	4,9e-76	2e-75	6,7e-80	1,4e-74
	10	3,2e-31	4,1e-31	7,1e-33	1,6e-30
Ackley	2	4,4e-16	0	4,4e-16	4,4e-16
	5	1,3e-12	9,3e-12	4,4e-16	6,6e-11
	10	0,57	0,21	0,0021	0,93
Rosenbrock	2	6,7e-09	3,4e-08	1,4e-13	2,4e-07
	5	0,2	0,21	0,0053	1,1
	10	3,8	1,6	0,18	5,8

5. CONCLUSIONES

En este artículo se brindó al lector la información necesaria para comprender e implementar con éxito la versión más simple del método *Cuckoo Search* (CS). CS es reconocido en la literatura como una herramienta poderosa para diversas aplicaciones prácticas [1, 3, 7, 8]. Asimismo, este método cuenta con 1435 documentos científicos publicados, desde su aparición en 2009 hasta el presente año (2017), de acuerdo con la base de datos bibliográfica *Scopus* de Elsevier B.V. Entre estos documentos se encuentran aplicaciones en ingeniería (49 %), energía (12 %), física y astronomía (6 %), ciencias ambientales (4 %), medicina (3 %), entre otras. Además, se registran al menos 18 patentes que utilizan métodos de optimización del estado del arte, incluyendo entre ellos a *Cuckoo Search*. Por lo tanto, es oportuno considerar a CS como un algoritmo del estado del arte presto a ser usado y adaptado para cualquier aplicación particular.

REFERENCIAS

[1] Mohamed Abdel-Basset, Abdel-Naser Hessin, and Lila Abdel-Fatah. 2016. A comprehensive study of cuckoo-inspired algorithms. *Neural Computing and Applications* (2016), 1–17.

[2] H Berlanga, H Gómez de Silva, VM Vargas-Canales, V Rodríguez-Contreras, LA Sánchez-González, R Ortega-Álvarez, and R Calderón-Parra. 2015. *Aves de México: Lista actualizada de especies y nombres comunes.* CONABIO, México DF.

[3] I Fister Jr., D Fister, and I Fistar. 2013. A comprehensive review of Cuckoo search: Variants and hybrids. *International Journal of Mathematical Modelling and Numerical Optimisation* 4, 4 (2013), 387–409.

[4] Momin Jamil and Xin She Yang. 2013. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation* 4, 2 (2013), 150.

[5] Rosario N. Mantegna and H. Eugene Stanley. 1994. Stochastic Process with Ultraslow Convergence to a Gaussian: The Truncated Levy Flight. *Physical Review Letters* 73, 22 (1994), 2946–2949.

[6] E P Raposo, S V Buldyrev, M G E da Luz, G M Viswanathan, and H E Stanley. 2009. Lévy flights and random searches. *Journal of Physics A: Mathematical and Theoretical* 42, 43 (2009), 23.

[7] Hetal Soneji and Rajesh C. Sanghvi. 2014. Towards the improvement of Cuckoo search algorithm. *International Journal of Computer Information Systems and Industrial Management Applications* 6 (2014), 77–88.

[8] XS Yang and S Deb. 2014. Cuckoo search: recent advances and applications. *Neural Computing and Applications* (2014), 1–9.

[9] Xin-She Yang. 2009. Cuckoo Search via Lévy flights. In *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*. IEEE, 210–214.

[10] Xin-She Yang. 2009. Firefly algorithms for multimodal optimization. In *Stochastic algorithms: foundations and applications*. Lecture Notes in Computer Sciences, Vol. 5792. Springer, Berlin, Heidelberg, 169–178.

[11] Xin-she Yang. 2010. A New Metaheuristic Bat-Inspired Algorithm. In *International Workshop on Nature Inspired cooperative Strategies for Optimization (NICSO 2010)*. Springer, Granada, 65–74.

[12] Xin She Yang. 2010. *Engineering Optimization: An Introduction with Metaheuristic Application.* John Wiley & Sons, Inc. 378 pages.

[13] Xin She Yang. 2012. Flower pollination algorithm for global optimization. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7445 LNCS (2012), 240–249.

[14] Xin-She Yang and Suash Deb. 2010. Engineering Optimisation by Cuckoo Search. *Int. J. Mathematical Modelling and Numerical Optimisation* 1, 4 (2010), 330–343.

[15] Xin-She Yang, Suash Deb, Simon Fong, Xingshi He, and Yu-Xin Zhao. 2016. From Swarm Intelligence to Metaheuristics: Nature-Inspired Optimization Algorithms. *Computer* 49, 9 (sep 2016), 52–59.

APÉNDICE

A continuación se incluyen los códigos para cada uno de los bloques de la Figura 1.

Código 2. Cuckoo Search.

```
function [BestNest,fBest]=CuckooSearch(fObj,bnd)
 % Ejemplo de uso: [BestNest,fBest]=CuckooSearch(@Sphere,[−10 10;−10 10]);
 %       donde y=Sphere(x) es la función a minimizar y, [−10 10;−10 10] es el espacio de búsqueda en 2D

N=25;  % Número de nidos (población)
M=1e3;  % Número máximo de generación o iteraciones
pD=0.25;  % Probabilidad de que un huevo sea descubierto
Delta=1.0;  % Tamaño del paso para los vuelos de Lévy
Beta=1.5;  % Parámetro para la distribución de Lévy

[D,bnd_1,bnd_2,Nests]=initialise(bnd,N);  % Bloque de inicialización (Código 3)
Fitness=evaluateFunction(N,fObj,Nests);  % Bloque de evaluación (Código 4)
[fBest,g]=min(Fitness); BestNest=Nests(g,:);  % Bloque de ranking (Código 9)

for generation = 1 : M  % Comienza el proceso iterativo hasta que alguno de los criterios de parada se cumplan
  newNests=LevyFlight(Nests,Delta,Beta,BestNest);  % Bloque de vuelo de Lévy (Código 7)
  newNests=simpleConstraints(newNests,bnd_1,bnd_2);  % Bloque de reubicación (Código 5)
  [Nests,Fitness]=updatePositions(N,D,fObj,Fitness,newNests,Nests);  % Bloque de actualización (Código 6)
  newNests=randomlyDiscoveringEggs(N,D,pD,Nests,Fitness);  % Bloque de supervivencia (Código 8)
  newNests=simpleConstraints(newNests,bnd_1,bnd_2);  % Bloque de reubicación (Código 5)
  [Nests,Fitness]=updatePositions(N,D,fObj,Fitness,newNests,Nests);  % Bloque de actualización (Código 6)
  [fBest,g]=min(Fitness); BestNest=Nests(g,:);  % Bloque de ranking (Código 9)
endfor
endfunction
```

Código 3. Bloque de inicialización.

```
function [Nd,bnd_1,bnd_2,positions]=initialise( bnd, Na)
Nd=size(bnd,1);
bnd=[min(bnd,[],2) max(bnd,[],2)];
bnd_1=ones(Na,1).*bnd(:,1)';
bnd_2=ones(Na,1).*bnd(:,2)';
 % Inicializa los nidos
positions=bnd_1+rand(Na,Nd).*(bnd_2−bnd_1);
endfunction
```

Código 4. Bloque de evaluación.

```
function evaluated_function=evaluateFunction( Na, the_function,points)
 % Inicializa el vector de las funciones evaluadas
evaluated_function=nan(Na,1);
 % Evalúa cada punto en la función
for s=1:Na
  evaluated_function(s)=the_function(points(s,:));
endfor
endfunction
```

Código 5. Bloque de reubicación.

```
function Nests=simpleConstraints( Nests, lowerBoundaries,
          upperBoundaries)
 % Verifica los nidos fuera de la región
check=Nests<lowerBoundaries;
 % Reubica los nidos a la frontera inferior
Nests=!check.*Nests+check.*lowerBoundaries;
 % Verifica los nidos fuera de la región
check=Nests>upperBoundaries;
 % Reubica los nidos a la frontera superior
Nests=!check.*Nests+check.*upperBoundaries;
endfunction
```

Código 6. Bloque de actualización.

```
function [new_positions,new_fitness] = updatePositions( Na, Nd,
          the_function, fitness, positions, best_positions)
evaluated_function = evaluateFunction(Na, the_function, positions);
condition = evaluated_function < fitness;
new_fitness = min(evaluated_function,fitness);
 % Actualiza los nidos requeridos
new_positions = (condition*ones(1,Nd)).*positions + ...
    (!condition*ones(1,Nd)).*best_positions;
endfunction
```

Código 7. Bloque de vuelo de Lévy.

```
function newNests=LevyFlight( Nests, Delta, Beta, BestNest)
sNests=size(Nests);
 % Calcula Sigma según el algoritmo de Mantegna
Sigma=(gamma(1+Beta)*sin(pi*Beta/2)/(Beta* ...
    gamma((1+Beta)/2)*2^((Beta−1)/2)))^(1/Beta);
 % Calcula los vuelos de Lévy
Lambda=(randn(sNests)*Sigma)./ ...
    (abs(randn(sNests)).^(1/Beta));
 % Actualizar la posición de los nidos
newNests=Nests+Delta*randn(sNests).* ...
     Lambda.*(Nests−ones(sNests(1),1)*BestNest);
endfunction
```

Código 8. Bloque de supervivencia.

```
function updated_Nests=randomlyDiscoveringEggs(Na, Nd, pD, Nest,
          fitness)
 % Modificar la porción pD de la población
updated_Nests=Nest+rand(Na,Nd).*double( ...
    rand(Na,Nd)>pD).*(Nest(randperm(Na),:)− ...
    Nest(randperm(Na),:));
endfunction
```

Código 9. Bloque de *ranking*.

```
[fBest,g]=min(Fitness); BestNest=Nests(g,:);
```