Análisis Comparativo del Algoritmo del Murciélago vs. Algoritmo Genético, Método De Descenso con Pendiente Máxima y Quasi-Newton-BFGS

Uriel David Tello Padilla[1]

[1]Escuela Superior de Física y Matemáticas, Instituto Politécnico Nacional
[2]Algoritmos bio-inspirados

Abstract

En este estudio se presentan los resultados de la comparación de desempeños de los algoritmos del murciélago, genético, descenso con máxima pendiente y quasi-Newton-BFGS. Los parámetros considerados para tal comparación son: evaluaciones de la función objetivo y precisión de la solución.

Keywords: algoritmos bio-inspirados, algoritmos genéticos.

1 Introducción

Existe una gran cantidad de algoritmos inspirados en la naturaleza. El propósito de este trabajo es analizar el funcionamiento del algoritmo del murciélago(**AM**) y compararlo con un algoritmo genético(**AG**), el método de descenso con pendiente máximo(**DPM**) y el método Broyden-Fletcher-Goldfarb-Shanno(**BFGS**). Comenzaremos con la descripción de las ideas básicas, los pasos principales, el pseudocódigo y los detalles de implementación de **AM**. Finalmente se realiza la ejecución de cada método para tres funciones de prueba y se muestran los resultados en una tabla comparativa. Todos algoritmos utilizados en este trabajo han sido codificados en el lenguaje de programación **Julia**. Con la intención de que el lector pueda comprender cómo se llevan a cabo las implementaciones y cómo funciona realmente el algoritmo se proporciona el código de **AM** en el apéndice.

1.1 El algoritmo del murciélago(Yang 2020).

Los murciélagos, especialmente los micro murciélagos, usan la eco-localización para navegar y buscar alimento. Estos murciélagos emiten una serie de ráfagas ultrasónicas cortas en el rango de frecuencia de 25 kHz a aproximadamente 150 kHz, y estos pulsos cortos suelen durar unos pocos milisegundos. La sonoridad de tales ráfagas y la tasa de emisión de pulsos varían durante la caza, especialmente cuando se busca una presa. El aumento de frecuencia reducirá la longitud de onda de los pulsos de ultrasonido, y así aumentar la resolución y precisión de la detección de las presas. Estas características de ecolocalización conocidas de los murciélagos se pueden simular en **AM**, que fue desarrollado por Xin-She Yang en 2010 (Yang X.S. Cruz C. González J.R. Pelta 2010). **AM** usa la sintonización de frecuencia como una fuerza impulsora aleatoria, en combinación con el uso de una tasa de emisión de pulso r y un sonoridad A para la población de murciélagos.

En **AM**, hay n murciélagos que forman una población de evolución iterativa. La ubicación de cada murciélago se denota por $x_i (i = 1, 2, \dots, n)$, que puede considerarse como el vector solución al problema de optimización en cuestión. Como los murciélagos vuelan en el espacio de búsqueda, cada murciélago también está asociado con un vector de velocidad v_i.

Un vector de posición se considera como un vector de solución a un problema de optimización en

Computer Science (2022)

DOI: 666

Corresponding author
Author One

Edited by
David Tello

un espacio de búsqueda de dimensión D con D variables de diseño independientes,

$$\mathbf{x} = [x_1, x_2, x_3, \ldots, x_D]. \tag{1}$$

Durante las iteraciones, cada murciélago puede variar su tasa de emisión de pulsos r_i, su sonoridad A_i y su frecuencia f_i. Las variaciones de frecuencia o la sintonización se pueden realizar mediante la ecuación

$$f_i = f_{\min} + \beta(f_{\max} - f_{\min}), \tag{2}$$

donde $f_{\min}$ y $f_{\max}$ son los rangos mínimo y máximo, respectivamente, de la frecuencia f_i para cada murciélago i. Aunque las variaciones de frecuencias no son uniformes en realidad, usaremos variaciones uniformes por simplicidad.

Las variaciones de frecuencias se utilizan luego para modificar las velocidades de los murciélagos en la población de modo que

$$v_i^{t+1} = v_i^t + \left(x_i^t - x_*\right) f_i, \tag{3}$$

donde $x*$ es la mejor solución obtenida por la población en la iteración t. Una vez que se actualiza la velocidad de un murciélago, su posición (o vector solución) x_i puede ser actualizado por

$$x_i^{t+1} = x_i^t + (\Delta t) v_i^{t+1}, \tag{4}$$

donde Δt es la iteración o el incremento de tiempo. Vale la pena señalar que todos los algoritmos iterativos se actualizan de manera discreta, lo que significa que podemos establecer Δt. Por lo tanto, podemos simplemente considerar los vectores sin unidades físicas y luego escribir la ecuación de actualización como

$$x_i^{t+1} = x_i^t + v_i^{t+1}. \tag{5}$$

Para la modificación local en torno a la mejor solución, podemos utilizar

$$x_{\text{nuevo}} = x_{\text{anterior}} + \sigma \epsilon_t A^{(t)}, \tag{6}$$

donde ϵ_t es un número aleatorio extraído de una distribución normal $N(0,1)$ y σ es la desviación estándar que actúa como factor de escala. Aquí, $A^{(t)}$ es la sonoridad promedio en la iteración t. Para simplificar, podemos usar $\sigma = 0.1$ en nuestra implementación posterior.

1.2 Emisión de pulsos y volumen

Las variaciones de la sonoridad y la tasa de emisión de pulsos se pueden incluir en el **AM** para que influyan en la forma de exploración y explotación. En el mundo real, sus variantes son muy complicadas, dependiendo de la especie de murciélago. Sin embargo, usamos una variación monótona aquí. La tasa de emisión de pulsos r_i puede monótonamente aumenta desde un valor más bajo r_i^0, mientras que el volumen puede disminuir desde un valor más alto i (como $A^0 = 1$) a un valor mucho más bajo. Entonces tenemos

$$A_i^{t+1} = \alpha A_i^t, \tag{7}$$

$$r_i^{t+1} = r_i^{(0)}[1 - \exp(-\gamma t)], \tag{8}$$

donde $0 < \alpha < 1$ y $\gamma > 0$ son constantes.

Con base en las fórmulas anteriores, podemos ver que cuando t es lo suficientemente grande ($t \to \infty$), obtenemos $At \to 0$ y $r_i^t \to r_i^{(0)}$. Para simplificar, podemos usar $\alpha = 0.97$ y $\gamma = 0.1$ en esta implementación simple. Además, utilizaremos la misma frecuencia de emisión de pulsos y volumen para todos los murciélagos, lo que simplifica mucho las implementaciones de los códigos de demostración presentados en este capítulo.

1.3 Pseudocódigo y parámetros

Hay bastantes parámetros en **AM**. Aparte del tamaño de la población n y los rangos de frecuencia f_{min} y $f_{\max}$, hay dos parámetros (α, β) y dos valores iniciales (A_0^0 y r_i^0). Usaremos los siguientes valores en nuestra implementación: $\alpha = 0.97$, $\gamma = 0.1$, $f_{\min} = 0$, $f_{\max} = 2$ y $A_i^{(0)} = r_i^{(0)} = 1$ para todos los murciélagos. Para el tamaño de la población n, se usará $n = 10$ en la implementación de la siguiente demostración.

Algorithm 1: El Algoritmo del Murciélago

function AM (f, n);
Input : Función, tamaño de la población n, parámetros α y γ
Output : Mejor posición global
Generar población inicial x_i y $v_i (i = 1, 2, ..., n)$;
Configure las frecuencias f_i, la frecuencia del pulso r_i y el volumen A_i;
Inicializar $iter = 0$ (contador de iteraciones);
while *(iter < $t_{\max}$)* **do**
    Variar r_i y A_i ; Ajustar frecuencias; Generar nuevas soluciones mediante las ecuaciones (3) y (5)
    if *rand* > r_i **then**
        Seleccionar una solución del conjunto de las mejores soluciones; Modificar localmente alrededor de la mejor solución seleccionada;
    end
    Hacer un vuelo aleatorio para generar una nueva solución;
    if *rand* > A_i *y* $f(x_i) < f(x_j)$*(para minimizar)* **then**
        Aceptar la nueva solución;
    end
    Clasificar la población actual de murciélagos y encuentre la mejor solución x*;
end
return la mejor solución

2 Comparación de los métodos

Se consideraron las funciones Esfera, Rosenbrock y Ackley para la comparación de los métodos.

2.1 Detalles Técnicos de Implementación de los Algoritmos.

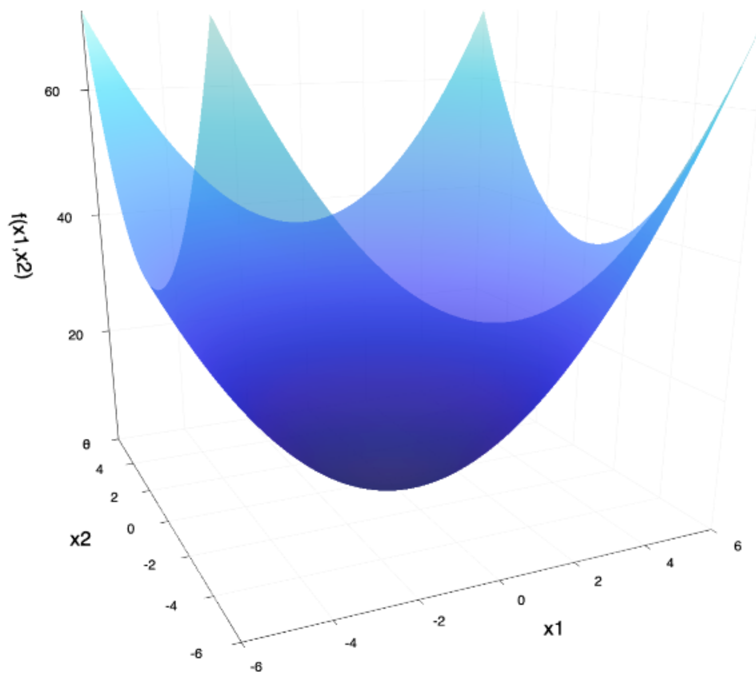
Con propósito de hacer una comparación justa entre todos los métodos, se ha modificado el criterio de paro del algoritmo (1), esto es, el criterio de paro que consistía originalmente en llegar a un número máximo de iteraciones ha sido reemplazado por una tolerancia para el valor de la función como se muestra a continuación en las líneas 18 a 20 de (2).

Algorithm 2: El Algoritmo del Murciélago(modificado)

```
1  function AM (f, n);
   Input   :Función, tamaño de la población n, parámetros α y γ
   Output  :Mejor posición global
2  Generar población inicial x_i y v_i (i = 1, 2, ..., n);
3  Configure las frecuencias f_i, la frecuencia del pulso r_i y el volumen A_i;
4  Inicializar iter = 0 (contador de iteraciones);
5  while (true) do
6      Variar r_i y A_i ; Ajustar frecuencias; Generar nuevas soluciones mediante las ecuaciones
         (3) y (5)
7      if rand > r_i then
8          Seleccionar una solución del conjunto de las mejores soluciones;
9          fmin = min(Aptitud) ;
10         Modificar localmente alrededor de la mejor solución seleccionada;
11     end
12     Hacer un vuelo aleatorio para generar una nueva solución;
13     fnew=F(Soluciones)
14     if rand > A_i y f(x_i) < f(x_j)(para minimizar) then
15         Aceptar la nueva solución;
16     end
17     Clasificar la población actual de murciélagos y encuentre la mejor solución x*;
18     if fmin <= Eps then
19         break
20     end
21 end
22 return la mejor solución
```

En este experimento consideramos vectores de dimensión $d = 2$, sin embargo el código de cada uno de los métodos es extensible a dimensiones superiores. Para los algoritmos **AM** y **AG** se consideró una población de $n = 10$. El espacio de búsqueda en el caso de **AM** se consideró como cota inferior -1 y cota superior 1. En cuanto al **AG** se consideró el intervalo $[-10, 10]$ para crear una población con distribución uniforme. Para los métodos **DPM** y **BFGS** se implementó un proceso multi-start el cual genera diez puntos aleatorios alrededor de $[-10, 10]$, por lo tanto los valores correspondientes a dichos métodos son calculados mediante el promedio de los valores obtenidos del multi-start.

2.2 Función Esfera



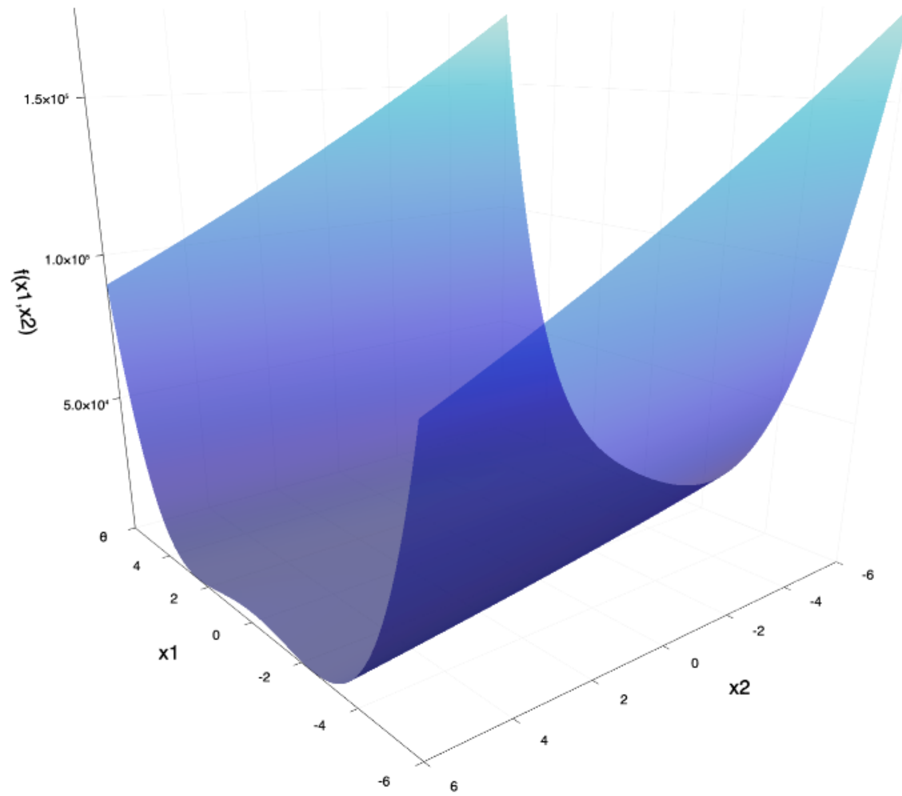
$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2. \quad (9)$$

Figure 1. La función Esfera tiene d mínimos locales excepto el global. Es continua, convexa y unimodal. La trama muestra su forma bidimensional. La función generalmente se evalúa en el hipercubo $x_i \in [-5.15, 5.12]$, para todo $i = 1, \ldots, d$. **Mínimo global**: $f(x) = 0$, en $x = (0, \ldots, 0)$.

Table 1. Tabla comparativa que muestra los resultados del desempeño del algoritmo del murciélago, algoritmo genético, el método de descenso con pendiente máxima y el método quasi Newton BFGS para encontrar el mínimo de la función (9).

Método/Algoritmo	Llamadas a la Función Objetivo	Precisión de la Solución	Tiempo de Ejecución
AM	234	$(8.5939e^{-}5, 2.5939e^{-}5)$	0.191061 s
AG	48,233	$(-0.00012, 0.00018)$	0.726477 s
DPM	14.0	$(-4.955e^{-}8, -1.651e^{-}8)$	0.011141 s
BFGS	20.1	$(3.6642e^{-}8, 1.6834e^{-}8)$	0.015317 s

2.3 Función de Rosenbrock



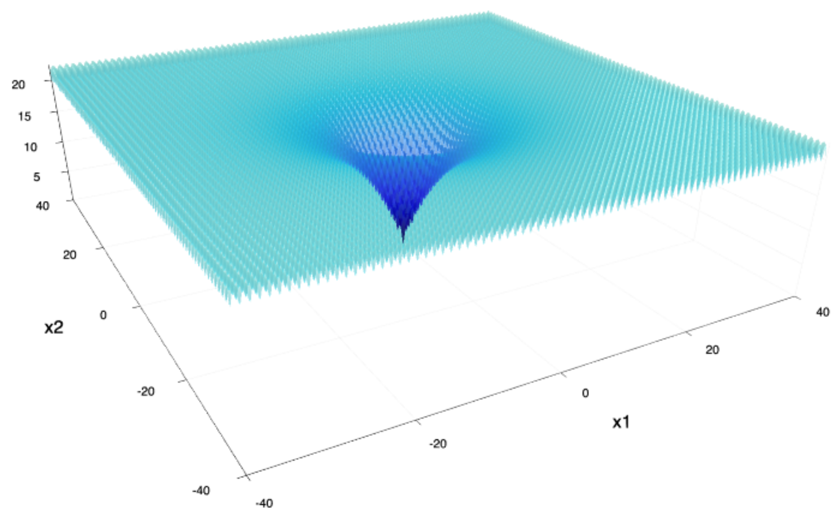
$$f(\mathbf{x}) = \sum_{i=1}^{d-1} \left[100 \left(x_{i+1} - x_i^2 \right)^2 + (x_i - 1)^2 \right]. \tag{10}$$

Figure 2. La función Rosenbrock, también conocida como función Valley o Banana. La función generalmente se evalúa en el hipercubo $x_i \in [-5, 10]$, para todo $i = 1, \ldots, d$, aunque puede estar restringida al hipercubo $x_i \in [-2.048, 2.048]$, para todo $i = 1, \ldots, d$. **Mínimo global**: $f(x) = 0$, en $x = (1, \ldots, 1)$.

Table 2. Tabla comparativa que muestra los resultados del desempeño del algoritmo del murciélago, algoritmo genético, el método de descenso con pendiente máxima y el método quasi Newton BFGS para encontrar el mínimo de la función (10).

Método/Algoritmo	Llamadas a la Función Objetivo	Precisión de la Solución	Tiempo de Ejecución
AM	11	(0.00905, −0.06643)	0.182204 s
AG	3,515,251	(0.9998, 0.9996)	38.488292 s
DPM	8.1	(3.74225, 5.21089)	0.0175868 s
BFGS	33.0	$(-3.58000e29, 2.57790e22)$	0.1884089 s

2.4 Función de Ackley



$$f(\mathbf{x}) = -a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1). \tag{11}$$

Figure 3. La función Ackley se usa ampliamente para probar algoritmos de optimización. En su forma bidimensional, como se muestra en el gráfico anterior, se caracteriza por una región exterior casi plana y un gran agujero en el centro. La función plantea el riesgo de que los algoritmos de optimización, en particular los algoritmos de escalada, queden atrapados en uno de sus muchos mínimos locales. La función generalmente se evalúa en el hipercubo $x_i \in [-32.768, 32.768]$, para todo $i = 1, \ldots, d$. **Mínimo global**: $f(x) = 0$, en $x = (0, \ldots, 0)$.

Table 3. Tabla comparativa que muestra los resultados del desempeño del algoritmo del murciélago, algoritmo genético, el método de descenso con pendiente máxima y el método quasi Newton BFGS para encontrar el mínimo de la función (11).

Método/Algoritmo	Llamadas a la Función Objetivo	Precisión de la Solución	Tiempo de Ejecución
AM	457	$(-2.08598e^{-}8, -1.23800e^{-}9)$	0.204535 s
AG	–	––	–
DPM	12.6	(4.13902, 3.92473)	0.0175133 s
BFGS	16.5	$(-3.49734e11, 79.11090)$	0.0172195 s

3 Conclusiones

En este estudio se ha logrado establecer que **AM** es potencialmente más eficiente que **AG**. Para **AG** se puede observar que tiene un número demasiado grande de llamadas a la función objetivo. Por otra parte, el tiempo de ejecución es mucho mayor que cualquiera de los otros métodos.

La precisión de la solución obtenida con **AM** es superior a la de todos los demás métodos, sin embargo, el número de llamadas a la función objetivo es bastante elevado comparado con **DPM** y **BFGS**.

En relación con el tiempo de ejecución, **AM** presenta un tiempo casi diez veces mayor que **DPM**y **BFGS**. Se puede ver que el **DPM** en cada caso, es el método que presenta un mejor tiempo de ejecución y también realiza un menor número de llamadas a la función objetivo. En cuanto a la precisión de la solución **DPM** tiene una buena precisión en el caso de la función Esfera(9), pero no así para la función de Rosenbrock(10) y la función de Ackley(11).

Si bien **AM**no es el más rápido o con menor llamadas a la función objetivo queda claro que es una opción altamente competitiva.

Una continuación interesante y necesaria de este trabajo consiste en considerar una gama más amplia de algoritmos existentes que utilizan funciones de prueba mucho más exigentes en dimensiones más altas las cuales plantearán más desafíos para los algoritmos y, por lo tanto, tales comparaciones potencialmente revelarán las virtudes y debilidades de todos los algoritmos de interés.

Appendices

A Código del algoritmo del murciélago.

```
using Distributions

 function AM(F::Function, tamaño_de_poblacion::Int64)
    """ ===================================================
    # ENTRADAS:
    #
    # Function      función objetivo
    # tamaño_de_poblacion         tamaño de la población
    # =================================================== """
    f_contador = 0 ; # contador de llamadas a la función objetivo
    # Ajuste de parámetros
    n = tamaño_de_poblacion ;
    A = 1.0
    r = 1.0
    alpha = 0.97
    gamma = 0.1
    # Rango de frequencia
    freqMin = 0 ;
    freqMax = 2 ;
    # Tolerancia
    eps = 0.0000001 ;
    iter = 0;      # contador de iteraciones
    # Dimensión de las variables
    d = 2 ;
    # Cota inferior y superior, respectivamente
```

```
lB = -1
uB = 1
# Iniciando los arreglos
Freq = zeros(n,1) ; # Frequencia Inicial
v = zeros(n,d) ;     # Velocidades Iniciales
Lb = lB*ones(1,d) ;# Cotas Inferiores
Ub = uB*ones(1,d) ; # Cotas superiores

# Iniciando población de n murciélagos(soluciones)
Sol = Array{Float64}(undef, n,d)
Aptitud = Array{Float64}(undef, 1,n)
for i = 1: n
    Sol[i,:] = Lb + (Ub-Lb).*rand(1,d)
    Aptitud[i] = sum(F(Sol[i,:]));
end
f_contador = f_contador + n ;
# Mejor murciélago(solución) en la población inicial
fmin = findmin(Aptitud, dims = 2)[1][1] # valor de la función
index = findmin(Aptitud, dims = 2)[2][1][2] # índice
mejorcito = Sol[index,:]



# Comienzo de las iteraciones
while true
    # Variación de los parámetros
    r = r*(1 - exp(-gamma*iter)) ;
    A = alpha*A ;
    # Creación de arreglos para guardar los valores
    Freq = Array{Float64}(undef, n, 1) ;
    v = Array{Float64}(undef, n, d) ;
    S = Array{Float64}(undef, n, d) ;
    # Repetición sobre todos los murciélagos
    for i = 1: n
        Freq[i] = freqMin + (freqMax - freqMin)*rand() ;
        v[i,:] = v[i,:] + (Sol[i,:] - mejorcito)*Freq[i] ;
        S[i,:] = Sol[i,:] + v[i,:] ;
        # Verificación de la condición de cambio
        if rand() > r
            S[i,:] = mejorcito' + 0.1*randn(1,d)*A;
        end
        # Evaluación de las soluciones
        Fnew = sum(F(S[i,:])) ;
        # Si la solución mejora
        if ( Fnew <= Aptitud[i] ) & (rand() > A )
            Sol[i,:] = S[i,:] ;
            Aptitud[i] = Fnew ;
        end
        # Actualiza mejorcito en la población
        if Fnew <= fmin
          mejorcito = S[i,:] ;
          fmin = Fnew ;
        end
```

```
        end # for
    iter = iter + 1 ;
    f_contador = f_contador + 1 ;
    if fmin <= eps # Criterio de paro
        break
    end
    println("Iteraciones: ", iter," ,fmin: ", fmin) ;
    end # while
    return mejorcito, fmin, f_contador
end # function
```

References

Yang, X.-S. 2020. *Nature-Inspired Computation and Swarm Intelligence.* April. ISBN: 9780128226094.

Yang X.S. Cruz C. González J.R. Pelta, D. T. G. 2010. "A new metaheuristic bat-inspired algorithm." *Nature Inspired Cooperative Strategies for Optimization (NISCO 2010). In: Studies in Computational Intelligence* 284:65–74.