



COMPUTACIÓN BIOINSPIRADA

ACTIVIDAD 1

Algoritmos de adaptación social

Conocer la existencia de otros algoritmos de adaptación social no
vistos en clase

Profesor: Javier Martínez Torres

Ernesto González Pradas
ernesto.gonzalez023@comunidadunir.net

Índice

Contenido

Índice.....	1
Introducción.....	2
Descripción del algoritmo elegido y motivo	3
Descripción del problema sobre el que se ha aplicado	5
Análisis de los resultados obtenidos	6
Conclusión.....	7
Bibliografía	8

Introducción

El objetivo de esta actividad es conocer la existencia de otros algoritmos de adaptación social no vistos en clase. Además, poder comprobar la aplicación real de un algoritmo bio-inspirado y tener la posibilidad de ejecutar dicho algoritmo probando su funcionamiento.

En primer lugar, realizaré una búsqueda de un algoritmo de adaptación social (*swarm intelligence*) no visto en clase, es decir, que no sea ni el algoritmo basado en colonia de hormigas ni el algoritmo basado en nubes de partículas.

Descripción del algoritmo elegido y motivo

Antes de describir el algoritmo elegido, nos hemos basado en una librería escrita en Python, que contiene una colección de distintos algoritmos inspirados en la naturaleza. Esta librería se llama “*NiaPy*” y se puede encontrar el link de descarga junto con su documentación en el apartado de bibliografía.

Lo primero, definiremos un algoritmo de adaptación social como todos aquellos algoritmos que traten de simular o imitar el comportamiento colectivo y colaborativo que tienen muchas especies de animales concretas con el objetivo de cumplir ciertas tareas que los benefician mutuamente.

Como mencionábamos en el primer párrafo, “*NiaPy*” nos proporciona una colección bastante extensa de dichos algoritmos, todos ellos muy sorprendentes, como por ejemplo, *algoritmo de Optimización de Arrecifes de Coral*, *algoritmo de luciérnaga*, *algoritmo de lobo gris* o *algoritmo optimización de los halcones de Harris* entre muchos otros. En nuestro caso, hemos elegido el **algoritmo de murciélago**.

Y es que este algoritmo se basa en la capacidad avanzada que tienen los murciélagos de ecolocalización. La mayoría de estos mamíferos voladores, utilizan unas señales de frecuencia modulada como si de un sonar se tratara con el que pueden identificar los objetos de su entorno o sus presas (incluso la velocidad de movimiento de dicha presa). La forma en que utilizan la ecolocalización es usando el retraso de tiempo que se produce entre la emisión y la detección del eco, la diferencia de tiempo entre sus dos oídos y las variaciones de volumen de los ecos construyendo así un escenario tridimensional del entorno.

Dicho comportamiento de ecolocalización de los murciélagos se puede formular de tal manera que se pueda asociar con la función objetivo a optimizar. Las reglas simplificadas que utiliza dicho algoritmo son:

- Todos los murciélagos usan la ecolocalización para detectar la distancia y también “conocen” la diferencia entre su presa y los objetos u obstáculos del entorno.
- Los murciélagos vuelan al azar con velocidad v_i en la posición x_i con una frecuencia fija f_{\min} variando la longitud de onda λ y el volumen A_0 para buscar presas.
- Aunque el volumen puede variar de muchas maneras, asumimos que el volumen oscila desde un A_0 positivo hasta un valor constante mínimo A_{\min} .

A continuación, se muestra el pseudocódigo y una breve explicación de cómo funciona el algoritmo:

Función objetivo $f(x)$, siendo $x = (x_1, \dots, x_d)^T$

Inicializamos la población de murciélagos x_i siendo $i = (1, 2, \dots, n)$ y v_i

Definimos la frecuencia de pulso f_i en x_i

Inicializamos las frecuencias de pulso r_i y el volumen A_i

while ($t < \text{Número máximo de iteraciones}$) {

 Genera nuevas soluciones ajustando la frecuencia y actualizando velocidades y ubicaciones/soluciones.

if ($\text{rand} > r_i$) {

 Selecciona una solución entre las mejores.

 Genera una solución local alrededor del final de la mejor solución.

 }

 Genera una nueva solución volando aleatoriamente.

if ($\text{rand} < A_i \ \&\& \ f(x_i) < f(x^*)$) {

 Acepta nueva solución.

 Incrementamos r_i y reducimos A_i

 }

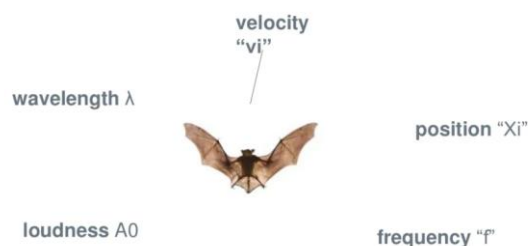
 Clasifica los murciélagos y encuentra el mejor actual x^* ($x_{\text{sub } *}$)

}

Se muestran los resultados del postproceso y visualización


Pseudocódigo del algoritmo de murciélago (BA)

El motivo de la elección de este algoritmo es que me parece fascinante que a alguien se le haya ocurrido la idea de utilizar la ecolocalización de los murciélagos para encontrar la solución de un problema.



Descripción del problema sobre el que se ha aplicado

En la página de github de Niapy, el código que viene para probar contiene numerosos tipos de problemas de optimización, como por ejemplo:

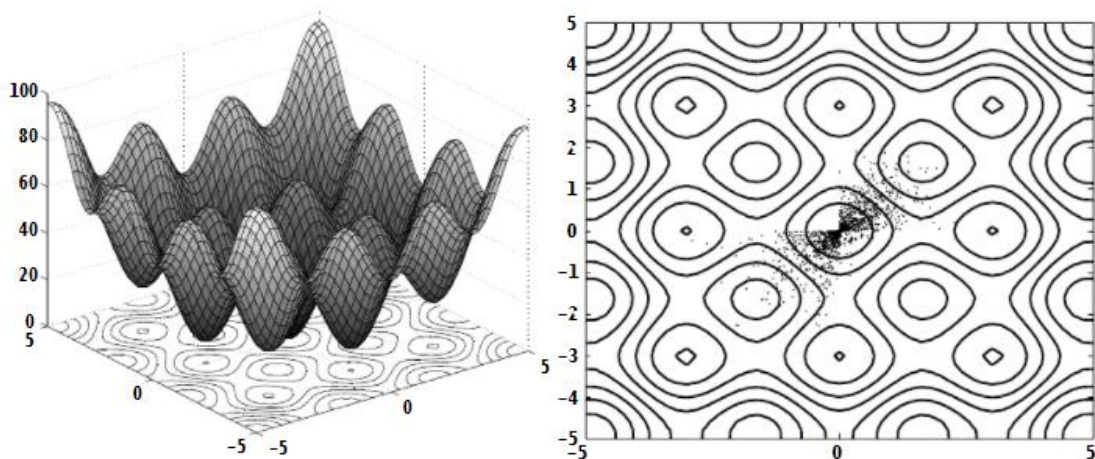


```
from niapy.problems.elliptic import Elliptic
from niapy.problems.griewank import Griewank, ExpandedGriewankPlusRosenbrock
from niapy.problems.happy_cat import HappyCat
from niapy.problems.hgbat import HGBat
from niapy.problems.katsuura import Katsuura
from niapy.problems.levy import Levy
from niapy.problems.michalewicz import Michalewicz
from niapy.problems.perm import Perm
from niapy.problems.pinter import Pinter
from niapy.problems.powell import Powell
from niapy.problems.qing import Qing
from niapy.problems.quintic import Quintic
from niapy.problems.rastrigin import Rastrigin
from niapy.problems.ridge import Ridge
from niapy.problems.rosenbrock import Rosenbrock
from niapy.problems.salomon import Salomon
from niapy.problems.schaffer import SchafferN2, SchafferN4, ExpandedSchaffer
from niapy.problems.schumer_steiglitz import SchumerSteiglitz
```

Hemos decidido utilizar el problema de Pinter que es el que venía en el propio ejemplo para implementar. Buscando un poco de documentación sobre este problema, vemos que es una función matemática compleja, la cual intentaremos resolver con el algoritmo seleccionado, en este caso el algoritmo de murciélago. La función matemática sería:

$$f(\mathbf{x}) = \sum_{i=1}^D ix_i^2 + \sum_{i=1}^D 20i \sin^2 A + \sum_{i=1}^D i \log_{10}(1 + iB^2); A = (x_{i-1} \sin(x_i) + \sin(x_{i+1})) \text{ and } B = (x_{i-1}^2 - 2x_i + 3x_{i+1} - \cos(x_i) + 1)$$

Es una función que se utiliza sobre todo para analizar problemas de optimización. Una ilustración de prueba nos quedaría tal que así:



Análisis de los resultados obtenidos

A continuación, pasamos a ver la implementación del código y a comentar los resultados. El código implementado sería el siguiente:

```
import sys

sys.path.append('../')
# End of fix

from niapy.algorithms.basic import BatAlgorithm
from niapy.task import Task

# we will run Bat Algorithm for 20 independent runs
for i in range(20):
    task = Task(problem="pinter", max_iters=100)
    algo = BatAlgorithm(population_size=25, loudness=1.0, pulse_rate=1.0, alpha=0.97, gamma=0.1, min_frequency=0.0,
                        max_frequency=2.0,)
    best = algo.run(task)
    print('%s -> %s' % (best[0], best[1]))
```

Observando en el código, vamos a lanzar 25 murciélagos virtuales (tamaño de la población), con un volumen inicial de 1.0 (loudness), una frecuencia de pulso de 1.0 (pulse rate), 0.97 para el parámetro Alpha que controla la disminución del volumen, 0.1 para gamma que controla el aumento de la frecuencia de pulso y dos parámetros opcionales que son la frecuencia mínima a 0.0 y la frecuencia máxima a 2.0.

Y la salida obtenida sería la siguiente:

```
C:\Users\ernes\AppData\Local\Programs\Python\Python310\python.exe C:/Users/ernes/IdeaProjects/CBAktividad1/AlgoritmoNiapy.py
[ 3.04136126  6.41674737 -0.59696611  2.59862629] -> 153.7613607976694
[-0.12382156  0.29176157  2.85545669  3.09145153] -> 85.63255172948345
[ 3.42286245 -1.54027036 -3.57035916  0.90250141] -> 104.99349363246945
[-2.50847148  1.40936233  3.80124314  1.30754781] -> 78.08791098807623
[-4.10152128 -4.19264535 -5.71970812 -1.24186075] -> 185.7872078004179
[-3.83705738 -0.75967483  2.52713406 -4.1760827 ] -> 149.65732673291834
[ 3.69202298 -3.32052101 -2.45310918  1.35400081] -> 99.18107097760871
[ 3.33081879 -0.24246984 -3.92760539  0.80989328] -> 103.67084437438723
[ 3.32088161  2.86378167 -1.36071624 -0.18270084] -> 59.709673319903985
[-2.25109596 -0.41619117 -1.8658053  3.51657979] -> 87.2941990448185
[-0.00014108 -0.00087096  0.00026002  0.00011135] -> 3.8445363825020216e-05
[ 0.5902219 -1.53057631 -3.74556846  1.33544657] -> 81.74901829140063
[ 7.53482214 -0.81722671 -2.0975544 -1.87150643] -> 124.51985295129904
[ 8.58468001 -3.96309575  3.18386168  3.35490321] -> 215.94280086519618
[-1.18258652  2.33486696  1.3147548  0.80301225] -> 38.94691983323399
[-3.18500837  3.8807028  0.96967811 -0.05128683] -> 76.74509938078705
[ 5.64296961  3.58073693 -0.77097377 -0.72667171] -> 88.91166381430614
[ 3.08663954e+00  9.06700760e-03 -2.37090627e-02 -6.48260399e-04] -> 25.69447271824533
[-0.37657978  2.72059561 -2.81461996  1.25650311] -> 78.57699951760391
[ 6.55369719 -5.94957421  1.45140269 -0.23900004] -> 157.65313475255155

Process finished with exit code 0
```

Conclusión

Me ha parecido una actividad muy interesante y buscando información sobre este algoritmo de adaptación social he visto que hay infinidad de ellos. Como comentaba en párrafos anteriores, me parece fascinante que al matemático Xin-She Yang, se le ocurriera basarse en la ecolocalización de los murciélagos para plantear este algoritmo de optimización.

Lo único que me ha faltado en esta actividad ha sido entender un poco mejor la implementación real en cuanto a la resolución del problema de Pintér. Entiendo cómo funciona el algoritmo de murciélago (creo que con el pseudocódigo mostrado es sencillo de entender y seguir) y entiendo los parámetros de entrada que tiene la función (es decir, entiendo a que se refieren cada uno de ellos) pero quizás me ha faltado profundizar un poco más en dicho problema para entenderlo realmente y poder describir si los resultados son buenos o si por el contrario son mejorables y como lo podrían ser.

Buscando información para realizar esta actividad, he encontrado que existen tres algoritmos modificados basados en el algoritmo descrito en esta actividad: *Algoritmo de murciélago híbrido*, *Algoritmo de murciélago auto adaptativo híbrido* y *Algoritmo de murciélago sin parámetros*. Esto me lleva a pensar que los resultados obtenidos no son los mejores y que se ha conseguido mejorar dicho algoritmo con estos tres últimos.

Bibliografía

- Pinter, J. (Enero de 2002). *researchgate.net*. Obtenido de https://www.researchgate.net/profile/Janos-Pinter/publication/221669684_Global_Optimization_Software_Test_Problems_and_Applications/links/59cbb5750f7e9bbfdc3b6c2b/Global-Optimization-Software-Test-Problems-and-Applications.pdf?origin=publication_detail
- Varios. (11 de Marzo de 2022). *github.com/NiaOrg/NiaPy*. Obtenido de <https://github.com/NiaOrg/NiaPy>
- Yang, X.-S. (Abril de 2010). *A New Metaheuristic Bat-Inspired Algorithm*. Obtenido de https://www.researchgate.net/publication/45913690_A_New_Metaheuristic_Bat-Inspired_Algorithm