Practica 4. Optimización por Enjambre de Particulas.

Algoritmos Bioinspirados

Diego Castillo Reyes
Marthon Leobardo Yañez Martinez
Aldo Escamilla Resendiz

17 de mayo de 2024

Índice general

1. Introducción

La Optimización por Enjambre de Partículas (PSO por sus siglas en inglés, Particle Swarm Optimization) es una técnica de optimización inspirada en el comportamiento social de los animales, particularmente en el vuelo de las aves y el movimiento de los peces en bancos. Esta técnica se basa en un enfoque colaborativo, donde un conjunto de soluciones candidatas, representadas como "partículas", navegan a través de un espacio de búsqueda buscando la óptima global siguiendo reglas simples.

Cada partícula en el enjambre ajusta su posición y velocidad de acuerdo con su propia experiencia pasada y la información compartida por las otras partículas. A medida que el algoritmo avanza en el tiempo, las partículas convergen gradualmente hacia las regiones de alta calidad del espacio de búsqueda, buscando mejorar el desempeño de la solución.

El PSO ha demostrado ser efectivo en una amplia gama de problemas de optimización, incluyendo la optimización de funciones matemáticas, diseño de ingeniería, problemas de programación, entre otros. Su simplicidad conceptual y su capacidad para escapar de óptimos locales lo hacen una herramienta valiosa en la caja de herramientas de los investigadores y profesionales en campos como la ingeniería, la ciencia de datos y la inteligencia artificial.

2. Estadisticas Descriptivas

Conf	Estadíst.	Funciones de prueba	
		Ackley	Rosenbrock
1 (0.25,0.0,0.0)	min	18.515644131362567	498.35182704181545
	max	21.95	1,084
	promedio	21.090003122849488	4553.570892237594
	des. est.		
2 (0.25,0.0,1.0)	min	9.517998536467005	16.59913616876128
	max	22.02	1,093
	promedio	9.517998536467003	16.59913616876128
	des. est.		
3 (0.25,0.0,2.0)	min	4.976418408338478	10.667890800223562
	max	22.09	1,095
	promedio	4.9764184083384775	10.667890800223566
	des. est.		
4 (0.25,1.0,0.0)	min	18.515644131362567	498.35182704181545
	max	21.94	1,088
	promedio	21.090003122849488	4553.570892237594
	des. est.		
5 (0.25,1.0,1.0)	min	9.517998536467005	16.59913616876128
	max	21.85	1,088
	promedio	9.517998536467003	16.59913616876128
	des. est.		
6 (0.25,1.0,2.0)	min	4.976418408338478	10.667890800223562
	max	21.87	1,095
	promedio	4.9764184083384775	10.667890800223566
	des. est.		
7 (0.25,2.0,0.0)	min	18.515644131362567	498.35182704181545
	max	21.86	1,088
	promedio	21.090003122849488	4553.570892237594
	des. est.		
8 (0.25,2.0,1.0)	min	9.517998536467005	16.59913616876128
	max	21.96	1,085
	promedio	9.517998536467003	16.59913616876128
	des. est.		
9 (0.25,2.0,2.0)	min	4.976418408338478	10.667890800223562
	max	21.83	1,095
	promedio	4.9764184083384775	10.667890800223566
	des. est.		
10 (0.5,0.0,0.0)	min	0.1668246681319836	498.35182704181545
	max	22.09	1,085
	promedio	0.1668246681319836	4553.570892237594
	des. est.		

Cuadro 1: Resultados de las funciones de prueba

Conf	Estadíst.	Funciones de prueba	
		Ackley	Rosenbrock
11 (0.5,0.0,1.0)	min	2.031481808228964	9.861834458823816
	max	22.05	1,085
	promedio	2.031481808228964	9.86183445882382
	des. est.		
12 (0.5,0.0,2.0)	min	0.1668246681319836	8.778958901717523
	max	21.93	7,915
	promedio	0.1668246681319836	8.77895890171752
	des. est.		
13 (0.5,1.0,0.0)	min	18.515644131362567	498.35182704181545
	max	21.93	1,084
	promedio	21.090003122849488	4553.570892237594
	des. est.		
14 (0.5,1.0,1.0)	min	2.031481808228964	9.861834458823816
	max	22.0	1,081
	promedio	2.031481808228964	9.86183445882382
	des. est.		
15 (0.5,1.0,2.0)	min	0.1668246681319836	8.778958901717523
	max	21.96	7,915
	promedio	0.1668246681319836	8.77895890171752
	des. est.		
16 (0.5,2.0,0.0)	min	18.515644131362567	498.35182704181545
	max	21.95	1,081
	promedio	21.090003122849488	4553.570892237594
	des. est.		
17 (0.5,2.0,1.0)	min	2.031481808228964	9.861834458823816
	max	21.83	1,085
	promedio	2.031481808228964	9.86183445882382
	des. est.		
18 (0.5,2.0,2.0)	min	0.1668246681319836	8.778958901717523
	max	21.96	7,896
	promedio	0.1668246681319836	8.77895890171752
	des. est.		

Cuadro 2: Resultados de las funciones de prueba

3. Graficas de convergencia

4. Analisis sobre el comportamiento del algoritmo de PSO

(a) ¿Que efecto tiene aumentar los parametros r1 y r2?
Aumentar los parámetros r1 y r2 puede aumentar la diversidad de las partículas en el espacio de búsqueda, lo que puede ayudar a explorar nuevas regiones y evitar que las partículas se estanquen en óptimos locales. Sin embargo, un aumento excesivo en r1 y r2 puede llevar a una exploración excesiva y a una convergencia más lenta.
(b) En general, ¿a que valor de w le fue mejor?
En general, el valor de w que mejor funcionó fue 0.25.
(c) ¿Cual es la peor configuracion de parametros para cada problema?
La peor configuracion de parametros para el problema de Ackley es la configuracion 10 (0.5,0.0,0.0) y para el problema de Rosenbrock es la configuracion 10 (0.5,0.0,0.0).
(d) ¿Cual es el problema que converge mas rapido al optimo global y cual mas lento?
El problema que converge mas rapido al optimo global es el problema de Ackley y el problema que converge mas

lento es el problema de Rosenbrock.
(e) ¿Que mejoras realizaria en el algoritmo de PSO?
Introducir mecanismos específicos para mejorar la convergencia, como estrategias de búsqueda local que se activan cuando las partículas se estancan, puede aumentar la eficiencia del PSO en la búsqueda de soluciones óptimas.

5. Regresión lineal

```python
# Regresion lineal simple con PSO

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Leer el archivo CSV
file_path = 'Salary.csv'
data = pd.read_csv(file_path)

# Preparar los datos
X = data['YearsExperience'].values
y = data['Salary'].values

# Definir la función de error cuadrático (MSE corregido)
def mse(params):
    a, b = params
    y_pred = a + b * X
    return np.sum((y - y_pred) ** 2)


def pso(num_particles, dimension, objective_function, w, c1, c2, generations, L, U):
    # Asignamos a n_a la función np.array para no tener que escribir np.array continuamente
    nA = np.array
    # Inicializar la posición y la velocidad de las partículas
    particles = [nA([np.random.uniform(L, U) for _ in range(dimension)]) for _ in range(num_particles)]
    velocities = [nA([0 for _ in range(dimension)]) for _ in range(num_particles)]

    # Establecemos el primer pbest y gbest
    pbest = particles
    gbest = min(pbest, key=objective_function)
    print(f"Posicion inicial de las partículas: {particles}")
    print(f"Mejor posición global inicial: {gbest} fitness: {objective_function(gbest)}")

    for _ in range(generations):
        for i in range(num_particles):
            r1, r2 = np.random.uniform(0, 1, 2)
            # Actualizamos la velocidad
            velocities[i] = w * velocities[i] + c1 * r1 * (pbest[i] - particles[i]) + c2 * r2 * (gbest - particles[i])
            # Actualizamos la posición
            particles[i] = particles[i] + velocities[i]
            # Actualizamos el pbest
            if objective_function(particles[i]) < objective_function(pbest[i]):
                pbest[i] = particles[i]

        # Actualizamos el gbest
        gbest = min(pbest, key=objective_function)

    print(f"Particulas: {particles}")
    print(f"Mejor posición por partícula: {pbest}")
    print(f"Mejor posición global: {gbest}")
    print(f"Fitness de la mejor posicion: {objective_function(gbest)}")
    # print(f"Fitness promedio de las mejores posiciones por particula: {np.mean([objective_function(p) for p in pbest])}")

    return gbest


```

```
# Parámetros
num_particles = 100
dimension = 2
#    ={0.25,0.5}
w = 0.5
# c1={0.0,1.0,2.0}
c1 = 2
# c2={0.0,1.0,2.0}
c2 = 2
generations = 500
fun = mse

# Definir límites uniformes para los parámetros a y b
L = 30000
U = 50000

# Se establece la semilla para reproducibilidad
np.random.seed(0)

gbest = pso(num_particles, dimension, fun, w, c1, c2, generations, L, U)

a, b = gbest
print(f"Mejor solución encontrada: a = {a}, b = {b}")

# Calcular valores predichos
y_pred = a + b * X

# Graficar los datos y la regresión lineal
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Datos reales')
plt.plot(X, y_pred, color='red', label='Regresión lineal')
plt.title('Regresión lineal simple con PSO')
plt.xlabel('Años de experiencia')
plt.ylabel('Salario')
plt.legend()
plt.grid(True)
plt.show()
```

6. Pruebas Realizadas

7. Lectura

En los últimos años, el número de enfoques de optimización bioinspirados ha crecido considerablemente, alcanzando niveles sin precedentes que ensombrecen las perspectivas futuras de este campo de investigación. Este artículo aborda este problema proponiendo dos taxonomías exhaustivas y basadas en principios que permiten a los investigadores organizar los desarrollos algorítmicos existentes y futuros en categorías bien definidas, considerando dos criterios diferentes: la fuente de inspiración y el comportamiento de cada algoritmo.
Utilizando estas taxonomías, revisamos más de trescientas publicaciones sobre algoritmos bioinspirados y propuestas que caen dentro de cada una de estas categorías, llevando a un resumen crítico de las tendencias de diseño y similitudes entre ellas, y la identificación del algoritmo clásico más similar para cada trabajo revisado. A partir de nuestro análisis, concluimos que a menudo se encuentra una relación pobre entre la inspiración natural de un algoritmo y su comportamiento. Además, las similitudes en términos de comportamiento entre diferentes algoritmos son mayores de lo que se afirma en su divulgación pública: específicamente, mostramos que más de un tercio de los solucionadores bioinspirados revisados son versiones de algoritmos clásicos. Basándonos en las conclusiones de nuestro análisis crítico, damos varias recomendaciones y puntos de mejora para mejores prácticas metodológicas en este campo de investigación activo y en crecimiento.
En los últimos años, se ha reportado una gran variedad de algoritmos bioinspirados en la literatura. Estos algoritmos se utilizan en problemas de optimización complejos donde los solucionadores exactos no son aplicables debido a su coste computacional o tiempo de resolución. Inspirados en los procesos biológicos observados en la naturaleza, estos algoritmos simulan procesos biológicos como la evolución natural y el comportamiento colectivo, dando lugar a conceptos como la *Swarm Intelligence*.

El artículo propone dos taxonomías principales:

- **Taxonomía basada en la fuente de inspiración:** Clasifica los algoritmos según su inspiración natural o biológica, permitiendo agrupar rápidamente los solucionadores publicados en la literatura.
- **Taxonomía basada en el comportamiento:** Clasifica los algoritmos según su comportamiento, es decir, cómo generan nuevas soluciones candidatas para la función a optimizar.

Del análisis de más de trescientas publicaciones, se concluye que existe una relación pobre entre la inspiración natural y el comportamiento del algoritmo. Además, muchas soluciones bioinspiradas son variantes de algoritmos clásicos. Se proporcionan recomendaciones para mejorar las prácticas metodológicas en la investigación, fomentando la aplicación de estos algoritmos a más problemas y la participación en competiciones para evaluar su rendimiento.

El artículo destaca la necesidad de organizar los algoritmos bioinspirados en taxonomías claras para facilitar su análisis y comparación. Se enfatiza que el comportamiento del algoritmo es más relevante que su inspiración natural, y se recomienda a los investigadores enfocarse en diferencias de comportamiento y evidencias de rendimiento verificables en problemas prácticos.

8. Conclusiones

Escamilla Reséndiz Aldo.
El algoritmo *Particle Swarm Optimization* (PSO) es un ejemplo destacado de la *Swarm Intelligence* y ha demostrado ser un enfoque efectivo y versátil para resolver problemas de optimización complejos. PSO simula el comportamiento social de los pájaros en busca de comida, permitiendo que las partículas (soluciones candidatas) en el espacio de búsqueda ajusten sus posiciones basándose en la experiencia personal y la de sus vecinos. Este enfoque ha inspirado muchos otros algoritmos y ha demostrado ser robusto en una amplia gama de aplicaciones.

Las funciones de prueba como la función de Rosenbrock y la función de Ackley son cruciales para evaluar y comparar el rendimiento de los algoritmos de optimización. La función de Rosenbrock, también conocida como el "valle de Rosenbrock.º "banana function", es una función no convexa utilizada comúnmente como un problema de prueba para los algoritmos de optimización debido a su forma de valle estrecho que lleva a un mínimo global. Por otro lado, la función de Ackley es conocida por su paisaje multimodal con muchos mínimos locales, lo que la hace particularmente desafiante y útil para probar la capacidad de los algoritmos para escapar de los óptimos locales.