Algoritmos Bioinspirados
Artificial Bee Colony (ABC)

Junnior Santiago Ravelo

Estudiante

Henry David Suarez Serrano
Estudiante

Javier Andrés Villarreal Amaya
Estudiante

Juan Camilo Arciniegas
Estudiante

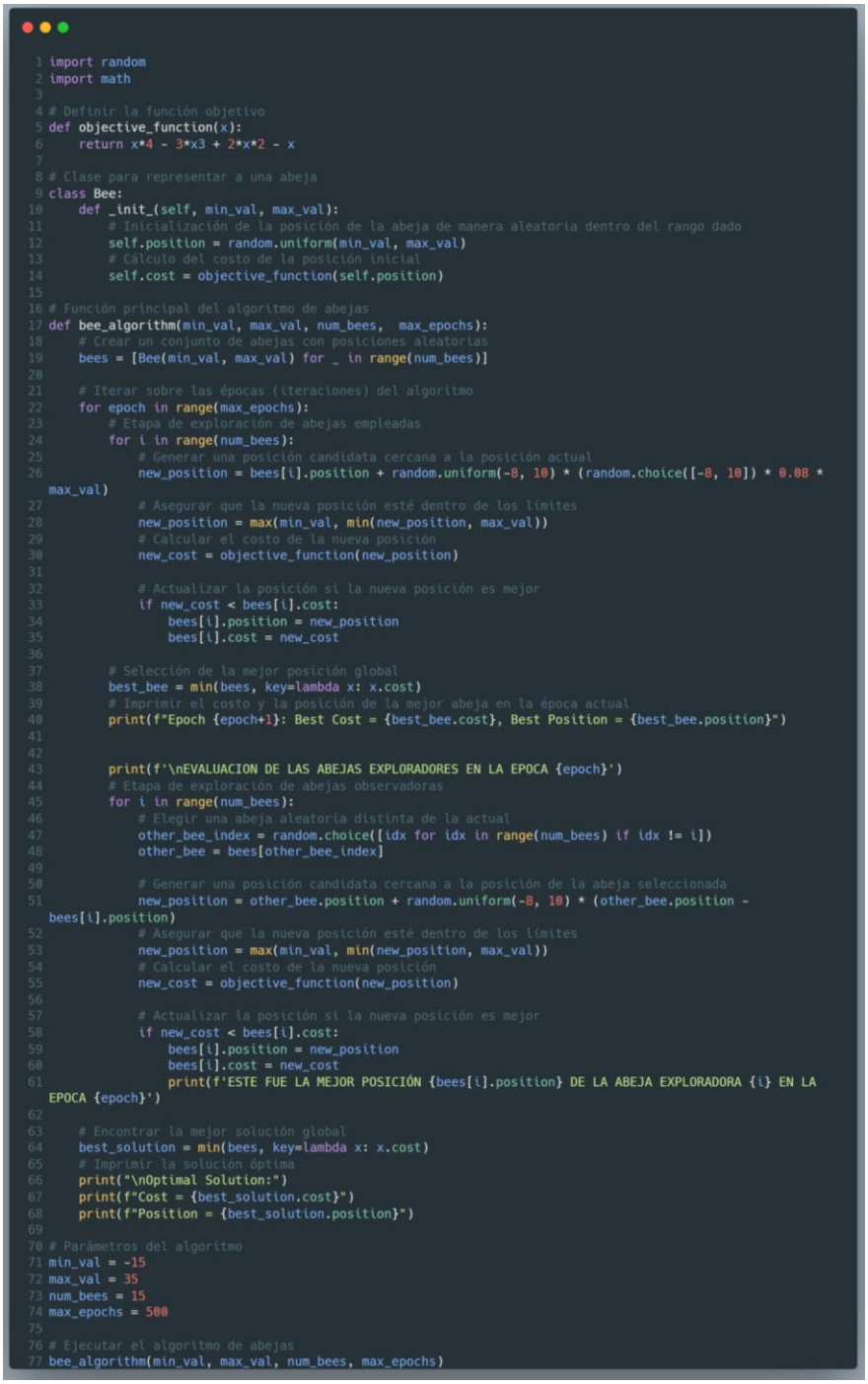
José Gerardo Chacón Rangel

Docente

Universidad de Pamplona

2024-1

Imagen del codigo



```python
import random
import math

# Definir la función objetivo
def objective_function(x):
    return x**4 - 3*x3 + 2*x**2 - x

# Clase para representar a una abeja
class Bee:
    def _init_(self, min_val, max_val):
        # Inicialización de la posición de la abeja de manera aleatoria dentro del rango dado
        self.position = random.uniform(min_val, max_val)
        # Cálculo del costo de la posición inicial
        self.cost = objective_function(self.position)

# Función principal del algoritmo de abejas
def bee_algorithm(min_val, max_val, num_bees,  max_epochs):
    # Crear un conjunto de abejas con posiciones aleatorias
    bees = [Bee(min_val, max_val) for _ in range(num_bees)]

    # Iterar sobre las épocas (iteraciones) del algoritmo
    for epoch in range(max_epochs):
        # Etapa de exploración de abejas empleadas
        for i in range(num_bees):
            # Generar una posición candidata cercana a la posición actual
            new_position = bees[i].position + random.uniform(-8, 10) * (random.choice([-8, 10]) * 0.08 *
max_val)
            # Asegurar que la nueva posición esté dentro de los límites
            new_position = max(min_val, min(new_position, max_val))
            # Calcular el costo de la nueva posición
            new_cost = objective_function(new_position)

            # Actualizar la posición si la nueva posición es mejor
            if new_cost < bees[i].cost:
                bees[i].position = new_position
                bees[i].cost = new_cost

        # Selección de la mejor posición global
        best_bee = min(bees, key=lambda x: x.cost)
        # Imprimir el costo y la posición de la mejor abeja en la época actual
        print(f"Epoch {epoch+1}: Best Cost = {best_bee.cost}, Best Position = {best_bee.position}")


        print(f'\nEVALUACION DE LAS ABEJAS EXPLORADORES EN LA EPOCA {epoch}')
        # Etapa de exploración de abejas observadoras
        for i in range(num_bees):
            # Elegir una abeja aleatoria distinta de la actual
            other_bee_index = random.choice([idx for idx in range(num_bees) if idx != i])
            other_bee = bees[other_bee_index]

            # Generar una posición candidata cercana a la posición de la abeja seleccionada
            new_position = other_bee.position + random.uniform(-8, 10) * (other_bee.position -
bees[i].position)
            # Asegurar que la nueva posición esté dentro de los límites
            new_position = max(min_val, min(new_position, max_val))
            # Calcular el costo de la nueva posición
            new_cost = objective_function(new_position)

            # Actualizar la posición si la nueva posición es mejor
            if new_cost < bees[i].cost:
                bees[i].position = new_position
                bees[i].cost = new_cost
                print(f'ESTE FUE LA MEJOR POSICIÓN {bees[i].position} DE LA ABEJA EXPLORADORA {i} EN LA
EPOCA {epoch}')

    # Encontrar la mejor solución global
    best_solution = min(bees, key=lambda x: x.cost)
    # Imprimir la solución óptima
    print("\nOptimal Solution:")
    print(f"Cost = {best_solution.cost}")
    print(f"Position = {best_solution.position}")

# Parámetros del algoritmo
min_val = -15
max_val = 35
num_bees = 15
max_epochs = 500

# Ejecutar el algoritmo de abejas
bee_algorithm(min_val, max_val, num_bees, max_epochs)
```

Codigo Escrito : Python

```
import random
import math


# Definir la función objetivo
def objective_function(x):
    return x*4 - 3*x3 + 2*x*2 - x


# Clase para representar a una abeja
class Bee:
    def _init_(self, min_val, max_val):
        # Inicialización de la posición de la abeja de manera aleatoria dentro del rango dado
        self.position = random.uniform(min_val, max_val)
        # Cálculo del costo de la posición inicial
        self.cost = objective_function(self.position)


# Función principal del algoritmo de abejas
def bee_algorithm(min_val, max_val, num_bees,  max_epochs):
    # Crear un conjunto de abejas con posiciones aleatorias
    bees = [Bee(min_val, max_val) for _ in range(num_bees)]


    # Iterar sobre las épocas (iteraciones) del algoritmo
```

```
    for epoch in range(max_epochs):
        # Etapa de exploración de abejas empleadas
        for i in range(num_bees):
            # Generar una posición candidata cercana a la posición actual
            new_position = bees[i].position + random.uniform(-8, 10) * (random.choice([-8, 10]) *
0.08 * max_val)
            # Asegurar que la nueva posición esté dentro de los límites
            new_position = max(min_val, min(new_position, max_val))
            # Calcular el costo de la nueva posición
            new_cost = objective_function(new_position)

            # Actualizar la posición si la nueva posición es mejor
            if new_cost < bees[i].cost:
                bees[i].position = new_position
                bees[i].cost = new_cost

        # Selección de la mejor posición global
        best_bee = min(bees, key=lambda x: x.cost)
        # Imprimir el costo y la posición de la mejor abeja en la época actual
        print(f"Epoch {epoch+1}: Best Cost = {best_bee.cost}, Best Position = {best_bee.position}")


        print(f'\nEVALUACION DE LAS ABEJAS EXPLORADORES EN LA EPOCA {epoch}')
```

```
        # Etapa de exploración de abejas observadoras
        for i in range(num_bees):
            # Elegir una abeja aleatoria distinta de la actual
            other_bee_index = random.choice([idx for idx in range(num_bees) if idx != i])
            other_bee = bees[other_bee_index]

            # Generar una posición candidata cercana a la posición de la abeja seleccionada
            new_position = other_bee.position + random.uniform(-8, 10) * (other_bee.position - bees[i].position)
            # Asegurar que la nueva posición esté dentro de los límites
            new_position = max(min_val, min(new_position, max_val))
            # Calcular el costo de la nueva posición
            new_cost = objective_function(new_position)

            # Actualizar la posición si la nueva posición es mejor
            if new_cost < bees[i].cost:
                bees[i].position = new_position
                bees[i].cost = new_cost
                print(f'ESTE FUE LA MEJOR POSICIÓN {bees[i].position} DE LA ABEJA EXPLORADORA {i} EN LA EPOCA {epoch}')

    # Encontrar la mejor solución global
    best_solution = min(bees, key=lambda x: x.cost)
    # Imprimir la solución óptima
```

```
    print("\nOptimal Solution:")
    print(f"Cost = {best_solution.cost}")
    print(f"Position = {best_solution.position}")

# Parámetros del algoritmo
min_val = -15
max_val = 35
num_bees = 15
max_epochs = 500

# Ejecutar el algoritmo de abejas
bee_algorithm(min_val, max_val, num_bees, max_epochs)
```

Repositorio en GitHub: [Ver repositorio]