Practica 1. Algoritmo genético para espacios continuos

Algoritmos Bioinspirados

Diego Castillo Reyes
Marthon Leobardo Yañez Martinez
Aldo Escamilla Resendiz

20 de marzo de 2024

Introducción

En esta practica hemos implementado un algoritmo genético para resolver un problema de optimización en un espacio continuo. El problema consiste en encontrar el mínimo de las funciones de Rosenbrock y Ackley, representadas por las siguientes ecuaciones respectivamente:

$$f(x)_{Rosenbrock} = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \tag{1}$$

$$f(x)_{Ackley} = -20exp(-0{,}2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)) + 20 + e \tag{2}$$

Para llevar a cabo esta tarea, se optó por el lenguaje de programación Python debido a su versatilidad, facilidad de uso y a la gran cantidad de librerías que ofrece para el manejo de arreglos y operaciones matemáticas. Además, se utilizó la librería *matplotlib* para graficar las funciones y el comportamiento del algoritmo genético.

Desarrollo

El algoritmo se dividió en 6 funciones principales:

Función de inicialización de población

```python
def start_poblation(f, poblation_size, dimension, lower_limit,
upper_limit):
    poblation = []

    for _ in range(poblation_size):
        individual = []
        for _ in range(dimension):
            individual.append(random.uniform(lower_limit,
upper_limit))

        if f == "rosenbrock":
            evaluation = rosenbrock(individual)
        elif f == "ackley":
            evaluation = ackley(individual)

        poblation.append([individual, evaluation])

    return poblation
```

Figura 1: Función población

Esta función se utiliza para inicializar una población de individuos con valores aleatorios en un rango específico dado por la función. La función toma cinco parámetros: el número de individuos, el número de variables, f, tamaño de la población, dimensión, limite inferior y limite superior.

- f: Es una cadena que especifica la función de aptitud que se utilizará, ya sea Rosenbrock o Ackley.
- tamaño de la población: Es el número de individuos que se generarán.
- dimensión: Es el número de variables que tendrá cada individuo.
- limite inferior: Es el valor mínimo que pueden tomar las variables de cada individuo.
- limite superior: Es el valor máximo que pueden tomar las variables de cada individuo.

La función comienza inicializando una población de lista vacía para contener la población de individuos. Luego ingresa a un bucle que ejecuta las veces de individuos seleccionados. En cada iteración de este ciclo, se crea un nuevo individuo.

Un individuo se representa como una lista de números de dimensión. Cada número es un flotante aleatorio generado por la función random.uniform, que devuelve un flotante aleatorio dentro del rango especificado por el límite inferior y límite superior. Una vez que se crea un individuo, su aptitud se evalúa utilizando la función de aptitud especificada. Si f es Rosenbrock, la función Rosenbrock se llama con el individuo como argumento. Si f es Ackley, en su lugar se llama a la función Ackley. La evaluación de aptitud se almacena en la evaluación de variables.

Finalmente, el individuo y su evaluación de aptitud se agregan como un par a la lista de población. Este proceso se repite hasta que se hayan creado y evaluado los individuos del tamaño de la población. La función devuelve la lista de población, que ahora contiene pares del tamaño de la población, cada uno de los cuales consta de un individuo y su evaluación de aptitud.

Función que define las parejas

```
def define_couples(poblacion, poblacion_size):
    fitness = [poblacion[i][1] for i in range(poblacion_size)]
    selected = []

    # Calcular la suma total de fitness
    total_fitness = sum(fitness)

    while len(selected) < poblacion_size:
        # Generar un número aleatorio entre 0 y la suma total de fitness
        selection = random.uniform(0, total_fitness)

        acumulated = 0
        for i, fit in enumerate(fitness):
            acumulated += fit

# Verificar si el acumulado supera el número aleatorio y si el índice no ha sido seleccionado previamente
            if acumulated >= selection:
                if i not in selected:
                    selected.append(i)
                    break
                else:
                    # Si el índice ya fue seleccionado, generar un nuevo número aleatorio y reiniciar el proceso
                    selection = random.uniform(0, total_fitness)
                    acumulated = 0

    # Asegurar que la cantidad de índices seleccionados sea par para formar parejas
    if len(selected) % 2 != 0:
        selected.pop()

    return selected
```

Figura 2: Función parejas

La función recibe como parámetros la población y el número de parejas que se seleccionarán.

- población: Es la lista de individuos que se cruzarán.
- número de parejas: Es el número de parejas que se seleccionarán.

Esta función se utiliza para seleccionar las parejas de individuos que se cruzarán para generar una nueva población. La función toma dos parámetros: la población y el número de parejas que se seleccionarán. La función empieza creando la lista fitness, que contiene los valores de ajuste de la población. Posteriormente crea la lista de seleccionados donde se almacenarán los indices de los individuos seleccionados. Luego, se ingresa a un bucle que se ejecuta el número de veces que se seleccionarán parejas. En cada iteración de este ciclo, se selecciona una pareja de individuos y se agrega a la lista de acumulados. Después de seleccionar una pareja, comprueba que la lista de las parejas que se cruzarán sea un número par y si no lo es, se elimina al ultimo individuo de la lista de seleccionados. Finalmente, la función devuelve la lista de seleccionados, que contiene los índices de los individuos seleccionados para cruzarse.

Función de cruza

```python
def crossover(parent1, parent2):
    # Seleccionar un punto de cruce aleatorio evitando extremos
    crossover_point = random.randint(1, len(parent1) - 1)

    # Sumamos las listas padre para obtener las listas hijas
    child1 = parent1[:crossover_point] + parent2[crossover_point:]
    child2 = parent2[:crossover_point] + parent1[crossover_point:]

    return child1, child2
```

Figura 3: Función cruza

La función cruza los individuos seleccionados para formar una nueva población.

- padre 1: Es el primer individuo seleccionado para cruzarse.
- padre 2: Es el segundo individuo seleccionado para cruzarse.

La función esta diseñada para realizar una cruza entre dos individuos y generar dos hijos. Esta empieza seleccionando un punto de cruza aleatorio, excluyendo el primero y el último índice de los individuos. Luego, crea dos hijos, el hijo 1 se crea utilizando la primera parte del padre 1 y la segunda parte del padre 2, mientras que el hijo 2 se crea utilizando la primera parte del padre 2 y la segunda parte del padre 1. Finalmente, la función devuelve una lista con los dos hijos generados.

Función de mutación

```python
def mutation(child, delta, mutation_percentage):
    mutated_child = []

    for gen in child:
        if random.uniform(0, 1) <= mutation_percentage:
            if random.uniform(0, 1) <= 0.5:
                gen += delta
            else:
                gen -= delta

        mutated_child.append(gen)

    return mutated_child
```

Figura 4: Función mutación

La función muta a los individuos de la población para mantener la diversidad en la población.

- hijo: Es el individuo que se mutará.
- delta: Es el valor que se sumará o restará a cada variable del individuo.
- porcentaje de mutación: Es el porcentaje de individuos que se mutarán.

La función toma tres argumentos como entrada: el hijo, el delta y el porcentaje de mutación. El hijo es un individuo que se mutará, el delta es el valor que se sumará o restará a cada variable del individuo y el porcentaje de mutación es el porcentaje de individuos que se mutarán.

La función comienza creando una lista donde se guardarán los hijos mutados