INSTITUTO POLITÉCNICO NACIONAL

ESCOM

INTELIGENCIA ARTIFICIAL

LAB 6: BUSQUEDA INFORMADA P3 FUNCION DE HIMMELBLAU MEDIANTE RECOCIDO SIMULADO

MARTÍNEZ CHÁVEZ JORGE ALEXIS

6CV3

17 OCTUBRE 2024

Introducción

La función de Himmelblau es una función matemática utilizada comúnmente como un en problemas de optimización no lineal. Es una función bidimensional que tiene múltiples mínimos locales, lo que la hace un desafío interesante para algoritmos de optimización como el recocido simulado. El recocido simulado, es un algoritmo de optimización inspirado en el proceso físico de recocido de metales.

Busca soluciones cercanas a un óptimo global de una función objetivo, explorando el espacio de búsqueda de manera probabilística. El algoritmo comienza con una "temperatura" alta, lo que permite aceptar soluciones peores con cierta probabilidad para escapar de óptimos locales. A medida que la temperatura disminuye, el sistema se "enfría" y se vuelve más selectivo, favoreciendo soluciones mejores.

Código

Primero debemos de hacer una función en donde debemos de definir la función de Himmelblau, en donde tiene como parámetros el recibir el valor de “x” y de “y” para que regrese la evaluación de esta función.

```
def himmelblau(x, y):
    return (x**2 + y - 11)**2 + (x + y**2 - 7)**2
```

Inicializamos los valor de “x” y de “y” asi como asignar los ragos maximo y minimos que pueden obtener esos valores, estos valores iniciales sera un random que este dentro de estos parametros.

```
initial_x = random.uniform(-5, 5)
initial_y = random.uniform(-5, 5)
```

Iniciamos la ejecución del porgrama, en donde obtendremos los mejores valores de “x” y de “y” asi como la evaluación de estos mismos, estos valores son los mejores valores encontrados. Como parametros enviamos los valores iniciales, asi como la evaluación de los mismos.

```
best_x, best_y, best_value = simulated_annealing(himmelblau, initial_x, initial_y)
```

La función del algoritmos recibe como parametros la primera evaluación de la función, asi como los valores inicales, asignamos un valor de maximo de iteraciones que se realizaran para poder evaluar, en este caso seran 10000 iteraciones. Tambien definimos el valor de temperatura, en este caso oucparemos 1000 y el rango de enfriamiento, que lo ocuparemos en 0.003. Inicializamos otras variables, que seran: un variable para el valor actual de “x” y otra para el valor actual de “y”, el valor actual de la función que ira siendo la evaluación de las 2 variables. Y ocuparemos otras 3 varibales para ir resguardando el mejor valor, sera dos para las variables y una para el resultado.

```
def simulated_annealing(func, x0, y0, max_iters=10000, T=1000.0, cooling_rate=0.003):
    # Inicializamos las variables
    current_x = x0
    current_y = y0
    current_value = func(current_x, current_y)
    best_x, best_y = current_x, current_y
    best_value = current_value
```

Iniciamos el for para para hacer las iteraciones desde el 1 hasta el valor que hemos definido, en cada iteración reduciremos la temperatura en donde le iremos restando el valor del rango de enfriamiento. Despues gerenamos nuevas varibales, su valor

debemos hacer random, pero que esten cercanos a los valores actuales, y comprobamos que estos neuvos valores esten dentro de los rango definidos del algoritmo. Y estos nuevos valores los mandamos a evaluar y lo reservamos en la varibale.

```
for i in range(max_iters):
    # Enfriamos la temperatura
    T = T * (1 - cooling_rate)

    # Generamos nuevos valores de x, y cercanos
    new_x = current_x + random.uniform(-1, 1)
    new_y = current_y + random.uniform(-1, 1)

    # Aseguramos que estén dentro de los límites [-5, 5]
    new_x = max(min(new_x, 5), -5)
    new_y = max(min(new_y, 5), -5)

    new_value = func(new_x, new_y)
```

Creamos una nueva variable que sera la difereancia de las evaluaciones, entre la nueva evaluación y la evaluación actual. Ahora debemos de hacer una comparativa en donde si la diferencia es menor a “0” , es decir, es negativo, significa que el nuevo punto mejora la solución; en caso de ser positivo significa que es peor pero todavía hay una porbabilidad de poder aceptar ese punto, este punto lo aceptaremos si un numero random que este entre (0, 1), es menor que la relación entre el negativo de la diferencia de las evaluaciones y la temperatura. En caso que se cumplan estas dos condiciones, entonces aceptamos el valor entonces los valores actuales los actualizamos con los nuevos valores.

Ahora debemos de ver si estos nuevos valores son los mejores, hacemos la comparativa que es la evaluacion actual es menor a la mejor evaluacion, entonces actualizamos los valores definimos como mejores con los valores actuales. Una vez que se hayan terminado las iteraciones, regresamos los mejores valores encontrados.

```
    # Calculamos la diferencia entre las funciones
    delta_value = new_value - current_value

    # Aceptamos el nuevo punto con probabilidad
    if delta_value < 0 or random.uniform(0, 1) < math.exp(-delta_value / T):
        current_x, current_y = new_x, new_y
        current_value = new_value

    # Actualizamos el mejor valor encontrado
    if current_value < best_value:
        best_x, best_y = current_x, current_y
        best_value = current_value

return best_x, best_y, best_value
```

Una vez que lo ejecutamos tenemos el siguiente resultado.

Enlace

A continuación dejamos el enlace del github completo, la practica actual se encuentra en la carpeta de LAB_6_BUSQUEDA_INFORMADA_P3: https://github.com/Jorge300403/IA_6CV3_MartinezChavez/tree/main/LAB_6_BUSQUEDA_INFORMADA_P3