Josue Santana Robledo Corona 325073061

Maestría en Ciencias de la Robótica e Inteligencia Artificial

Algoritmos Bio-Inspirados

Centro Universitario de Ciencias Exactas e Ingeniería.

Mtro. Carlos Alberto López Franco

1. Introducción

Este reporte presenta la implementación del algoritmo Hill Climbing con paso ascendente y reinicios aleatorios, aplicado a funciones de prueba en n dimensiones.
El objetivo es encontrar máximos locales/globales de una función mediante búsqueda local, evaluando varias posiciones iniciales para mejorar la probabilidad de hallar el óptimo global.

2. Descripción del código

El código implementa el algoritmo Hill Climbing paso ascendente generalizado para cualquier número de dimensiones:

- Evalúa la función en la posición actual.
- Genera vecinos incrementando o decrementando cada dimensión en un paso fijo.
- Selecciona el vecino que mejora la función.
- Repite hasta que no se encuentre un mejor vecino o se cumplan las iteraciones máximas.

Se añaden reinicios aleatorios, lo que permite ejecutar el algoritmo varias veces desde posiciones iniciales diferentes y conservar el mejor resultado global.

3. Componentes del Código

Funciones de Prueba Implementadas

```
def funcion(x):
  return -np.sum(x**2) + 4*np.sum(x)

def hill_climbing_paso_ascendente(funcion_obj, x_inicial, max_iter=1000, paso=0.5):
  x_actual = np.array(x_inicial)
  mejor_x = x_actual.copy()
  mejor_valor = funcion_obj(x_actual)
  historia = [x_actual.copy()]

  for i in range(max_iter):
    valor_actual = funcion_obj(x_actual)
    mejor_vecino = None
    mejor_valor_vecino = valor_actual

    for dim in range(len(x_actual)):
      x_positivo = x_actual.copy()
      x_positivo[dim] += paso
      valor_positivo = funcion_obj(x_positivo)

      x_negativo = x_actual.copy()
```

```
            x_negativo[dim] -= paso
            valor_negativo = funcion_obj(x_negativo)

            if valor_positivo > mejor_valor_vecino:
                mejor_valor_vecino = valor_positivo
                mejor_vecino = x_positivo

            if valor_negativo > mejor_valor_vecino:
                mejor_valor_vecino = valor_negativo
                mejor_vecino = x_negativo

        if mejor_vecino is not None and mejor_valor_vecino > valor_actual:
            x_actual = mejor_vecino
            mejor_valor = mejor_valor_vecino
            mejor_x = x_actual.copy()
            historia.append(x_actual.copy())
        else:
            break

    return mejor_x, mejor_valor, np.array(historia).
```

funcion_obj: función a maximizar.

x_inicial: vector inicial aleatorio.

max_iter: número máximo de iteraciones por reinicio.

paso: magnitud del movimiento en cada dimensión.

```
for i in range(num_reinicios):
    x_inicial = np.random.uniform(-10,10,size=n_dim)
    x_optimo, valor_optimo, historia = hill_climbing_paso_ascendente(funcion, x_inicial, max_iter, paso)
```

Esta función es la que permite realizar los reinicios.

Resultados Obtenidos

Primero vamos con ascendente y descendente en dimensión 2

Al igual que el programa anterior. Para hacerlo descendente y ascendente solo hay que cambiar estas líneas

Minimización

if valor_positivo < mejor_valor_vecino:

if valor_negativo < mejor_valor_vecino:

if mejor_vecino is not None and mejor_valor_vecino < valor_actual:

if valor_optimo < mejor_global_valor:

mejor_global_valor = np.inf

Maximización

if valor_positivo> mejor_valor_vecino:

if valor_negativo> mejor_valor_vecino:

if mejor_vecino is not None and mejor_valor_vecino > valor_actual:

if valor_optimo > mejor_global_valor:

mejor_global_valor = -np.inf

Para cambiar entre minimización y maximización hay que cambiar el signo de esas 4 líneas.

Minimización 2D

Reinicio 1/5 → Mejor valor local: -254498.8423

Reinicio 2/5 → Mejor valor local: -257491.4933

Reinicio 3/5 → Mejor valor local: -257423.7880

Reinicio 4/5 → Mejor valor local: -262058.2548

Reinicio 5/5 → Mejor valor local: -259596.0797

Mejor punto global encontrado: [7.58673427 -509.89358577]

Mejor valor global: -262058.255

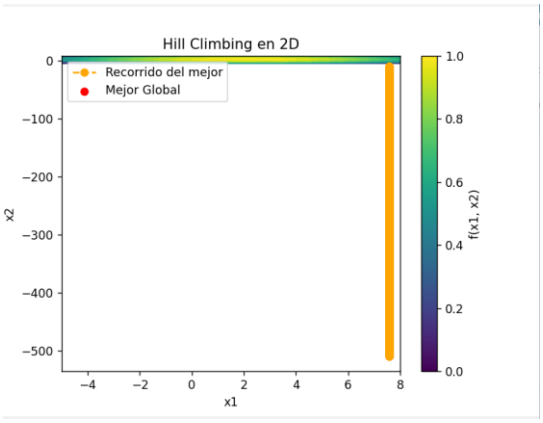

Imagen 1. Descendente en 2D

Maximización 2D

Reinicio 1/5 → Mejor valor local: 7.9428

Reinicio 2/5 → Mejor valor local: 7.9494

Reinicio 3/5 → Mejor valor local: 7.9863

Reinicio 4/5 → Mejor valor local: 7.9460

Reinicio 5/5 → Mejor valor local: 7.9531

Mejor punto global encontrado: [1.97824442 2.11502851]

Mejor valor global: 7.986

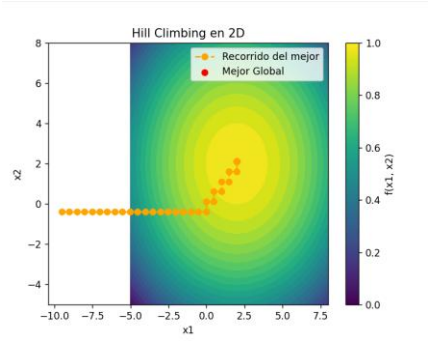

Imagen 2. Ascendente en 2D

Minimización 3D

Reinicio 1/5 → Mejor valor local: -259816.0239

Reinicio 2/5 → Mejor valor local: -254096.5210

Reinicio 3/5 → Mejor valor local: -259011.2617

Reinicio 4/5 → Mejor valor local: -254069.6050

Reinicio 5/5 → Mejor valor local: -255713.2729

Mejor punto global encontrado: [3.95767759 -507.702081 7.2896125]

Mejor valor global: -259816.024

Maximización 3D

Reinicio 1/5 → Mejor valor local: 11.9525

Reinicio 2/5 → Mejor valor local: 11.8938

Reinicio 3/5 → Mejor valor local: 11.9414

Reinicio 4/5 → Mejor valor local: 11.9393

Reinicio 5/5 → Mejor valor local: 11.9270

Mejor punto global encontrado: [1.90612934 1.97589047 1.80466921]

Mejor valor global: 11.952

4. **Ventajas del Método**

Simplicidad: fácil de entender e implementar.

Eficaz para espacios de baja a media dimensión: encuentra rápidamente máximos locales.

Reinicios aleatorios: aumentan la probabilidad de encontrar el máximo global.

5. **Limitaciones**

Puede quedarse atrapado en óptimos locales si la función tiene muchos picos.

La magnitud del paso y número de iteraciones afecta fuertemente el resultado.

Escalar a dimensiones muy altas puede requerir ajustes de parámetro y más reinicios.

6. **Conclusiones**

Este código permite explorar funciones en multiples dimensiones, e implementa la estructura del reinicio, como en otros programas no es difícil de implementar, se realizó en unas líneas de código ya explicadas en el reporte.

Código

```
import random
import numpy as np
import matplotlib.pyplot as plt


def funcion(x):
  return -np.sum(x**2) + 4*np.sum(x)

def hill_climbing_paso_ascendente(funcion_obj, x_inicial, max_iter=1000, paso=0.5):
  x_actual = np.array(x_inicial)
  mejor_x = x_actual.copy()
  mejor_valor = funcion_obj(x_actual)
  historia = [x_actual.copy()]

  for i in range(max_iter):
    valor_actual=funcion_obj(x_actual)
    mejor_vecino=None
    mejor_valor_vecino=valor_actual

    for dim in range(len(x_actual)):
      x_positivo=x_actual.copy()
      x_positivo[dim]+=paso
      valor_positivo=funcion_obj(x_positivo)

      x_negativo=x_actual.copy()
      x_negativo[dim]-=paso
      valor_negativo=funcion_obj(x_negativo)

      if valor_positivo> mejor_valor_vecino:
        mejor_valor_vecino = valor_positivo
        mejor_vecino=x_positivo

      if valor_negativo> mejor_valor_vecino:
        mejor_valor_vecino=valor_negativo
        mejor_vecino=x_negativo

    if mejor_vecino is not None and mejor_valor_vecino > valor_actual:
      x_actual=mejor_vecino
      mejor_valor=mejor_valor_vecino
      mejor_x=x_actual.copy()
      historia.append(x_actual.copy())
    else:
      break
```

```
    return mejor_x, mejor_valor, np.array(historia)


n_dim = 2
x_inicial = np.random.uniform(-10, 10, size=n_dim)
max_iter = 1000
paso = 0.5
num_reinicios=5

mejor_global_x = None
mejor_global_valor = -np.inf
mejor_historia = None

for i in range(num_reinicios):
    x_inicial = np.random.uniform(-10,10,size=n_dim)
    x_optimo, valor_optimo, historia = hill_climbing_paso_ascendente(funcion, x_inicial,
max_iter, paso)

    print(f"Reinicio {i + 1}/{num_reinicios} → Mejor valor local: {valor_optimo:.4f}")
    if valor_optimo > mejor_global_valor:
        mejor_global_valor = valor_optimo
        mejor_global_x = x_optimo
        mejor_historia = historia

print(f"Mejor punto global encontrado: {mejor_global_x}")
print(f"Mejor valor global: {mejor_global_valor:.3f}")

if n_dim == 2:
    x1 = np.linspace(-5, 8, 200)
    x2 = np.linspace(-5, 8, 200)
    X1, X2 = np.meshgrid(x1, x2)
    Z = -X1**2 - X2**2 + 4*X1 + 4*X2

    plt.contourf(X1, X2, Z, levels=30, cmap='viridis')
    plt.plot(mejor_historia[:, 0], mejor_historia[:, 1], 'o--', color='orange', label='Recorrido del
mejor')
    plt.scatter(mejor_global_x[0], mejor_global_x[1], color='red', label='Mejor Global')
    plt.title("Hill Climbing en 2D")
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.legend()
    plt.colorbar(label='f(x1, x2)')
    plt.show()
```