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 reemplazo (First Improvement), una variación del método clásico de búsqueda local.
El algoritmo explora un conjunto de vecinos generados aleatoriamente en cada iteración y reemplaza la solución actual únicamente si alguno de ellos mejora el valor de la función objetivo.

2. **Descripción del código**

El código implementa un algoritmo de **Hill Climbing con reemplazo aleatorio**, donde en cada paso se generan varios candidatos (vecinos) alrededor del punto actual.
Se evalúan todos y se selecciona el que tenga el mejor valor.
Si este mejora el valor de la solución actual, el algoritmo se mueve hacia él; en caso contrario, se detiene.

3. **Componentes del Código**

Función Objetivo

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

```
def hill_climbing_con_reemplazo(funcion_obj, x_inicial, max_iter=1000, paso=0.5, num_vecinos=20):
  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):
    vecinos = [x_actual + np.random.uniform(-paso, paso, size=len(x_actual)) for _ in range(num_vecinos)]
    valores = [funcion_obj(v) for v in vecinos]

    idx_mejor_vecino = np.argmax(valores)
    mejor_vecino = vecinos[idx_mejor_vecino]
    mejor_valor_vecino = valores[idx_mejor_vecino]
```

```
    if mejor_valor_vecino > mejor_valor:
        x_actual = mejor_vecino
        mejor_x = x_actual.copy()
        mejor_valor = mejor_valor_vecino
        historia.append(x_actual.copy())
    else:
        break

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

- Se parte de un punto inicial aleatorio.
- En cada iteración se generan varios vecinos aleatorios alrededor del punto actual.
- Se evalúa la función objetivo en todos los vecinos.
- Se selecciona el vecino con mejor valor.
- Si mejora el resultado actual, se reemplaza la solución.
- Si no hay mejora, el algoritmo se detiene.

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

```
idx_mejor_vecino = np.argmin(valores)

if mejor_valor_vecino < mejor_valor:
    x_actual = mejor_vecino
    mejor_x = x_actual.copy()
    mejor_valor = mejor_valor_vecino
    historia.append(x_actual.copy())
```

Maximización

```
idx_mejor_vecino = np.argmax(valores)
if mejor_valor_vecino > mejor_valor:  # Maximización
   x_actual = mejor_vecino
   mejor_x = x_actual.copy()
   mejor_valor = mejor_valor_vecino
   historia.append(x_actual.copy())
```

Minimización 2D

Punto inicial: x = [-8.38711002 -7.32026244], f(x) = -186.759

Resultado optimizado: x = [-373.31803258 -363.26756477], f(x) = -274276.019

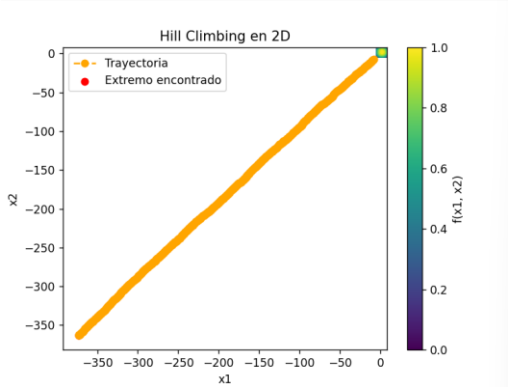

Imagen 1. Descendente en 2D

Maximización 2D

Punto inicial: x = [-7.63215179 8.3458107], f(x) = -125.048

Resultado optimizado: x = [2.0138364 2.01101492], f(x) = 8.000

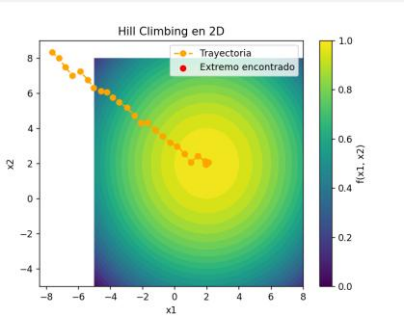

Imagen 2. Ascendente en 2D

Minimización 3D

Punto inicial: x = [-3.54837847 -2.97946517 -6.34160673], f(x) = -113.162

Resultado optimizado: x = [-301.96412379 -266.88669474 -347.18460766], f(x) = -286612.133

Maximización 3D

Punto inicial: x = [-4.82079601 -1.69802598 -0.73178623], f(x) = -55.661

Resultado optimizado: x = [1.92993648 2.05462967 1.85565964], f(x) = 11.971

4. Ventajas del Método

Permite explorar el espacio de búsqueda de forma aleatoria, evitando caer rápidamente en óptimos locales.
La generación de varios vecinos por iteración mejora la calidad del movimiento.

5. Limitaciones

A pesar de generar vecinos aleatorios, no garantiza alcanzar el óptimo global si la función tiene múltiples máximos.
Requiere elegir cuidadosamente el número de vecinos y el tamaño del paso para obtener buenos resultados.
Puede detenerse prematuramente si no encuentra una mejora, incluso si aún hay mejores soluciones cercanas.

6. Conclusiones

El algoritmo de Hill Climbing con reemplazo representa una mejora sobre la versión determinista, al incorporar aleatoriedad en la exploración.
En la función de prueba implementada, logra encontrar el máximo global con éxito.Este código realiza la búsqueda local con reemplazo del mejor vecino.

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_con_reemplazo(funcion_obj, x_inicial, max_iter=1000,
paso=0.5,num_vecinos=20):
  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):
    vecinos = [x_actual + np.random.uniform(-paso, paso, size=len(x_actual)) for _ in
range(num_vecinos)]
    valores=[funcion_obj(v) for v in vecinos]

    idx_mejor_vecino=np.argmin(valores)
    mejor_vecino=vecinos[idx_mejor_vecino]
    mejor_valor_vecino=valores[idx_mejor_vecino]

    if mejor_valor_vecino < mejor_valor:
      x_actual = mejor_vecino
      mejor_x=x_actual.copy()
      mejor_valor = mejor_valor_vecino
      historia.append(x_actual.copy())
    else:
      break


  return mejor_x, mejor_valor, np.array(historia)


n_dim = 3
x_inicial = np.random.uniform(-10, 10, size=n_dim)
max_iter = 1000
paso = 0.5
num_vecinos = 20

x_optimo, valor_optimo, historia = hill_climbing_con_reemplazo(funcion, x_inicial, max_iter,
paso,num_vecinos)

print(f"Punto inicial: x = {x_inicial}, f(x) = {funcion(x_inicial):.3f}")
print(f"Resultado optimizado: x = {x_optimo}, f(x) = {valor_optimo:.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(historia[:, 0], historia[:, 1], 'o--', color='orange', label='Trayectoria')
plt.scatter(x_optimo[0], x_optimo[1], color='red', label='Extremo encontrado')
plt.title("Hill Climbing en 2D")
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.colorbar(label='f(x1, x2)')
plt.show()
```