

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Ingeniería en Sistemas Computacionales

Inteligencia Artificial

Profesor:

Andrés García Floriano

Laboratorio 1:

Búsqueda aleatoria

Alumnos:

Carmona Santiago Yeimi Guadalupe

Hernández Suárez Diego Armando

Flores Osorio Adolfo Ángel

Grupo:6CV3

INSTITUTO POLITÉCNICO NACIONAL



ESCOM

Introducción

En el ámbito de la Inteligencia Artificial (IA), los métodos de búsqueda juegan un papel fundamental en la resolución de problemas, especialmente cuando se trata de explorar grandes espacios de soluciones donde la optimización o el análisis exhaustivo no son viables. Este reporte aborda el concepto de búsqueda aleatoria como una técnica utilizada en dos escenarios diferentes: la minimización de una función no lineal y el desarrollo de un juego de gato en un tablero de 4x4, donde los movimientos de la computadora son generados de forma aleatoria.

La búsqueda aleatoria es una técnica que selecciona soluciones potenciales sin seguir un patrón específico, esperando que algunas de estas resulten óptimas o cercanas a una solución ideal. Aunque es una técnica básica comparada con algoritmos más complejos, su simplicidad la hace útil en contextos simples.

En este reporte, se exploran dos aplicaciones prácticas de la búsqueda aleatoria:

Encontrar valores mínimos de una función: Este problema implica la búsqueda de pares de números dentro de un rango $-4.5 \leq x, y \leq 4.5$, con el objetivo de minimizar el valor de una función no lineal. El algoritmo genera soluciones al azar y selecciona aquellas que resulten en el menor valor posible.

Juego de gato de 4x4: En este caso, se diseña un juego de gato donde los movimientos de la computadora no siguen una estrategia basada en lógica o predicción, sino que se eligen al azar. Esto permite analizar el impacto de la aleatoriedad en la toma de decisiones y su viabilidad en un entorno controlado como el de un juego.

Ambos ejercicios permiten no solo explorar la efectividad de la búsqueda aleatoria en diferentes dominios, sino también reflexionar sobre sus limitaciones dentro del campo de la Inteligencia Artificial. A través de esta implementación, se busca ofrecer una visión práctica del comportamiento de esta técnica en la resolución de problemas, enfatizando su simplicidad y adaptabilidad.

Valores mínimos

Asignación. Encuentra los valores mínimos de la función:

$$f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$$

En el rango: $-4.5 \leq x, y \leq 4.5$

Algoritmo

Este código genera de manera aleatoria pares de números dentro de un rango específico y calcula el resultado de una función en estos números. El objetivo del código es encontrar el par de números que minimiza el valor de esta función después de realizar 10,000 iteraciones.

```
#encontrar los valores mínimos de la función

import random #Esta librería se usa para generar números aleatorios

#Se crea un bucle se ejecuta 10000 veces, cada iteración genera dos números aleatorios
for i in range(10000):
    num1=round(random.uniform(-4.5,4.5),2)
    num2=round(random.uniform(-4.5, 4.5),2)

    #Se asegura que los dos números generados hayan sido diferentes
    while num1==num2:
        num2=random.uniform(-4.5,4.5)
    resultado=(1.5-num1+(num1*num2))**2+(2.25-num1+(num1*(num2**2)))**2+(2.625-num1+num1*(num2**3))**2 #Cálculo de la función

    #Comparación para encontrar el menor
    if i == 0:
        resultado_aux=resultado
    else:
        if resultado<resultado_aux:
            resultado_aux=round(resultado,5)

#Resultado final
print(f"Par de números final: {num1}, {num2}={resultado_aux}")
```

Explicación del algoritmo:

- Primero, el código genera pares de números aleatorios en el rango de -4.5 a 4.5.
- Luego, asegura que los dos números en cada par sean diferentes.

- Para cada par de números, calcula el valor de una función específica. Esta función está compuesta por tres términos que dependen de los números generados, y cada término es elevado al cuadrado y sumado para obtener un resultado total.
- A lo largo de 10,000 iteraciones, el código compara el valor de la función para cada par de números con el valor mínimo encontrado hasta ese momento.
- Al final de todas las iteraciones, el código muestra el par de números que dio el valor mínimo de la función.

Resultados

Podemos observar que el algoritmo está diseñado para buscar exhaustivamente dentro del rango definido, aumentando la probabilidad de encontrar un valor cercano al mínimo global. Aunque no garantiza encontrar el mínimo global, puede proporcionar soluciones adecuadas en un tiempo razonable para problemas de tamaño moderado.

↔ Par de números final: 2.07, 4.44 = 0.00121

↔ Par de números final: 2.97, -2.13 = 0.00016

↔ Par de números final: 2.29, 3.87 = 0.00502

↔ Par de números final: -1.81, 2.84 = 0.0067

↔ Par de números final: 3.43, -2.28 = 0.00375

Juego de gato

Asignación. Diseña un juego de gato de 4x4, en donde los movimientos de la computadora sean movimientos elegidos al azar.

Algoritmo

Este código implementa un juego de **Gato** en un tablero de 4x4, está construido alrededor de verificaciones condicionales y chequeo de arreglos, donde un jugador humano juega contra la computadora. El jugador usa "X" y la computadora usa "O". El juego continúa hasta que uno de los jugadores gana o el tablero se llena, resultando en un empate.

Código

```
gato.py X
C: > Users > Adolfo > Desktop > gato.py > mostrar_tablero
1 import random
2 import time
3
4 from os import system
5
6 # Imprime el tablero
7 def mostrar_tablero(tablero):
8     for fila in tablero:
9         print(" ".join(fila))
10    print("-" * 7)
11
12 # Verifica si hay un ganador
13 def winner_check(tablero, jugador):
14     # Verificar filas y columnas
15     for i in range(4):
16         if all([tablero[i][j] == jugador for j in range(4)]) or all([tablero[j][i] == jugador for j in range(4)]):
17             return True
18
19     # Verificación de diagonales
20     if tablero[0][0] == jugador and tablero[1][1] == jugador and tablero[2][2] == jugador and tablero[3][3] == jugador:
21         return True
22     if tablero[0][3] == jugador and tablero[1][2] == jugador and tablero[2][1] == jugador and tablero[3][0] == jugador:
23         return True
24     return False
25
26 # Verifica si el tablero está lleno
27 def tablero_lleno(tablero):
28     return all([cell != " " for fila in tablero for cell in fila])
29
30 # Movimiento del jugador
31 def turno_jugador(tablero):
32     while True:
33         fila = int(input("Elige una fila (1, 2, 3, 4): ")) - 1
34         columna = int(input("Elige una columna (1, 2, 3, 4): ")) - 1
35         if tablero[fila][columna] == " ":
36             tablero[fila][columna] = "X"
37             break
38         else:
39             print("Posición ocupada, intenta nuevamente.")
```

```

40
41 # Movimiento de la computadora (al azar)
42 def turno_computadora(tablero):
43     movimientos_computadora = [(i, j) for i in range(4) for j in range(4) if tablero[i][j] == " "]
44     movimiento = random.choice(movimientos_computadora)
45     tablero[movimiento[0]][movimiento[1]] = "O"
46
47
48
49 tablero = [[" " for _ in range(4)] for _ in range(4)]
50 mostrar_tablero(tablero)
51
52 while True:
53     # Movimiento del jugador
54     print("Turno del jugador (X):")
55     turno_jugador(tablero)
56     mostrar_tablero(tablero)
57
58     # Verificar si el jugador ganó
59     if winner_check(tablero, "X"):
60         print("Ganaste mi pa!")
61         break
62
63     # Verificar si el tablero está lleno
64     if tablero_lleno(tablero):
65         print("Empataron mi pa!")
66         break
67
68     # Movimiento de la computadora
69     print("Turno de la computadora (O):")
70     turno_computadora(tablero)
71     print("Procesando jugada...")
72     time.sleep(3)
73     system("cls")
74     mostrar_tablero(tablero)
75

```

```

75
76     # Verificar si la computadora ganó
77     if winner_check(tablero, "O"):
78         print("La computadora te ganó mi pa!")
79         break
80
81     # Verificar si el tablero está lleno
82     if tablero_lleno(tablero):
83         print("Empataron mi pa!")
84         break

```

Impresión del Tablero

La función `mostrar_tablero(tablero)` se encarga de mostrar el estado actual del tablero de juego en la consola. Se imprimen las filas con separadores para que el jugador pueda visualizar el tablero fácilmente.

Verificación de Ganador

La función `winner_check(tablero, jugador)` verifica si el jugador (humano o computadora) ha ganado el juego. Se verifican tres condiciones:

- Alineación de cuatro fichas en cualquier fila.
- Alineación de cuatro fichas en cualquier columna.

- Alineación de cuatro fichas en cualquiera de las dos diagonales principales.

Verificación de Tablero Lleno (Empate)

La función `tablero_lleno(tablero)` comprueba si todas las posiciones del tablero están ocupadas. Si es así, el juego termina en empate.

Movimiento del Jugador

La función `turno_jugador(tablero)` permite al jugador humano ingresar las coordenadas (fila y columna) para su movimiento. Si el lugar está ocupado, se solicita al jugador que elija otra posición.

Movimiento de la Computadora (Aleatorio)

La función `turno_computadora(tablero)` genera movimientos al azar para la computadora. Busca todas las posiciones disponibles en el tablero y elige una aleatoriamente para colocar la ficha "O".

Dinámica del Juego

1. **Inicio:** Se inicializa el tablero vacío (4x4) y se muestra al jugador.
2. **Turno del Jugador:** El jugador elige una fila y columna para colocar su "X". Se verifica si ha ganado o si el tablero está lleno.
3. **Turno de la Computadora:** La computadora selecciona una posición vacía al azar para colocar su "O". Después de cada movimiento, se verifica si la computadora ha ganado o si el tablero está lleno.
4. **Final del Juego:** El juego termina cuando:

- Un jugador (humano o computadora) gana al alinear 4 fichas en fila, columna o diagonal.
- El tablero se llena y el resultado es un empate.

Resultados

```

C:\Users\Adolfo\AppData\Loc  X  +  v
| | |
-----
| | |
-----
| | |
-----
| | |
-----
Turno del jugador (X):
Elige una fila (1, 2, 3, 4): |

```

```

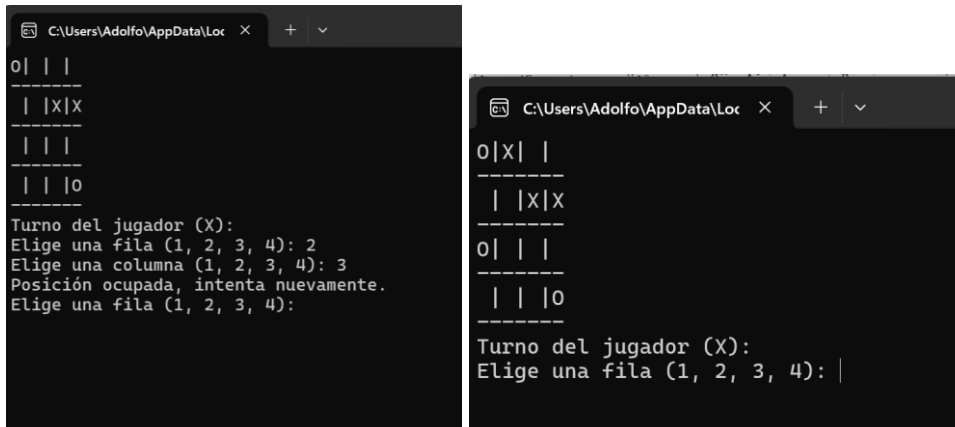
C:\Users\Adolfo\AppData\Loc  X  +  v
| | |
-----
| | |X
-----
| | |
-----
| | |O
-----
Turno del jugador (X):
Elige una fila (1, 2, 3, 4): 2
Elige una columna (1, 2, 3, 4): 3
| | |
-----
| |X|X
-----
| | |
-----
| | |O
-----
Turno de la computadora (O):
Procesando jugada...

```

```

C:\Users\Adolfo\AppData\Loc  X  +  v
O| | |
-----
| |X|X
-----
| | |
-----
| | |O
-----
Turno del jugador (X):
Elige una fila (1, 2, 3, 4): |

```

```
C:\Users\Adolfo\AppData\Local\Programs\Python\Python39\python.exe C:\Users\Adolfo\AppData\Local\Programs\Python\Python39\python.exe
0| | |
-----
| |X|X
-----
| | |
-----
| | |O
-----
Turno del jugador (X):
Elige una fila (1, 2, 3, 4): 2
Elige una columna (1, 2, 3, 4): 3
Posición ocupada, intenta nuevamente.
Elige una fila (1, 2, 3, 4):

C:\Users\Adolfo\AppData\Local\Programs\Python\Python39\python.exe C:\Users\Adolfo\AppData\Local\Programs\Python\Python39\python.exe
0|X| |
-----
| |X|X
-----
0| | |
-----
| | |O
-----
Turno del jugador (X):
Elige una fila (1, 2, 3, 4): |
```

Referencias

W3Schools.com. (s. f.). https://www.w3schools.com/python/python_variables.asp

Montiel, O. (2022, 22 febrero). *La guía para principiantes de Git y Github*. freeCodeCamp.org. <https://www.freecodecamp.org/espanol/news/guia-para-principiantes-de-git-y-github/>

Hola mundo - Documentación de GitHub. (s. f.). GitHub Docs. <https://docs.github.com/es/get-started/start-your-journey/hello-world>

math — Mathematical functions. (s. f.). Python Documentation. <https://docs.python.org/es/3/library/math.html>