Instituto Politecnico Nacional
Escuela Superior de Computo
Ing. Sistemas Computaciones

BUSQUEDA ALEATORIA

Practica 1

Materia:
Inteligencia Artificial

Grupo:
6CV3

Profesor:
Garcia Floriano Andres

Autores:
Aldo Alcántara Martínez
Sebastián Reyes Núñez
Lagarza Ortega Ana Karen
Estrada Chávez Dilan Daniel
Flores Vicencio Marco Isai

Fecha:
8 de Septiembre del 2024

Introducción

En el campo de la Inteligencia Artificial, uno de los desafíos más significativos es el desarrollo de algoritmos capaces de encontrar soluciones óptimas o aceptables en espacios de búsqueda complejos. Dentro de este contexto, la búsqueda aleatoria representa una técnica elemental que, aunque es básica, juega un rol importante en la comprensión y aplicación de estrategias de búsqueda más sofisticadas.

La búsqueda aleatoria, como su nombre lo indica, consiste en explorar el espacio de soluciones de manera no sistemática, seleccionando opciones al azar con la esperanza de encontrar una solución válida. Si bien a primera vista puede parecer ineficiente, este tipo de búsqueda resulta útil en situaciones donde no se tiene suficiente información sobre el espacio de soluciones o en problemas donde la estructura de la solución es muy compleja o estática.

Marco Teórico

La búsqueda de soluciones en problemas de IA suele involucrar la exploración de un espacio de búsqueda, que puede ser finito o infinito dependiendo de la naturaleza del problema. Las técnicas de búsqueda tienen como objetivo encontrar soluciones que maximicen o minimicen una función objetivo, o bien que cumplan con ciertas restricciones impuestas por el problema. Dentro de estas técnicas, la búsqueda aleatoria se distingue por no seguir un patrón predefinido, lo que implica que selecciona posibles soluciones de manera aleatoria y las evalúa en función de su idoneidad.

Ventajas y Desventajas

- Ventajas: Simplicidad, facilidad de implementación y flexibilidad en problemas desconocidos.
- Desventajas: Ineficiencia y falta de garantías de encontrar soluciones óptimas o válidas en un tiempo razonable.

Aplicaciones

Si bien la búsqueda aleatoria rara vez se utiliza por sí sola en aplicaciones prácticas de IA debido a su ineficiencia, sirve como base para comprender técnicas más avanzadas, como la búsqueda heurística o la optimización basada en algoritmos genéticos. En particular, es un componente básico en algoritmos de Monte Carlo, que combinan la aleatoriedad con evaluaciones probabilísticas para encontrar soluciones aproximadas.

Además, la búsqueda aleatoria puede ser útil en la generación de soluciones iniciales para algoritmos más complejos, o en situaciones donde el espacio de búsqueda cambia dinámicamente, ya que su falta de rigidez le permite adaptarse a estos cambios de manera flexible.

Desarrollo

En esta práctica se llevaron a cabo dos ejercicios utilizando el algoritmo de búsqueda aleatoria, aplicándolo en distintos contextos.

Ejercicio 1: Búsqueda de los Valores Mínimos de una Función

En este ejercicio, se empleó la búsqueda aleatoria para encontrar los valores mínimos de una función matemática. El algoritmo realizó múltiples pruebas con valores aleatorios, evaluando la función en cada caso, todo dentro de un límite de 10,000 iteraciones. El objetivo fue explorar el espacio de búsqueda y aproximarse a los valores mínimos de la función.

Comenzamos importante la librería de numpy en Python para poder usar las funciones de números aleatorios.

```
import numpy as np
```

Después definimos la función dada en el ejercicio:$((1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2)$ con límites de $-4.5 \leq x, y \leq 4.5$.

```
# AAqui se difne la función
def f(x, y):
    return ((1.5 - x + x * y) ** 2 +
            (2.25 - x + x * y ** 2) ** 2 +
            (2.625 - x + x * y ** 3) ** 2)

# Limites del Dominio
x_min, x_max = -4.5, 4.5
y_min, y_max = -4.5, 4.5
```

Seguimos con la inicialización de las variables donde se almacenarán los valores mínimos encontrados. Los valores se inician en “None” porque aún no hemos encontrado ningún valor. La siguiente línea inicializa la variable de valor mínimo con “float(‘inf’)” ya que la idea es que cualquier valor encontrado en la función sea menos que un infinito positivo.

```
# Inicialización de variables para valores minimos
best_x, best_y = None, None
min_value = float('inf')
```

Después, definimos la variable “iterations” para establecer el número máximo de iteraciones que se harán para la búsqueda aleatoria. Seguido de esto, comenzamos un bucle que se repetirá el número de iteraciones.

```
# Busqueda Aleatoria con 10000 iteraciones
iterations = 10000
for _ in range(iterations):
```

Dentro de este bucle, se generan números aleatorios para x,y dentro del rango establecido previamente ($-4.5 \leq x, y \leq 4.5$). Una vez que se asignan los valores aleatorios a x,y se calcula el valor de la función f(x,y) en esos puntos.

```
x = np.random.uniform(x_min, x_max)
y = np.random.uniform(y_min, y_max)
current_value = f(x, y)
```

El código procede a hacer la comparación si el valor actual "current_value" de la función es el valor más pequeño que ha encontrado. Si el valor actual es más pequeño que el que está guardado, "min_value" se actualiza con el nuevo valor mínimo. A su vez, se guardan los valores de x,y en donde se encontró el valor más pequeño.

```
# Se actualiza el valor minimo si se encuentra uno nuevo
if current_value < min_value:
    min_value = current_value
    best_x, best_y = x, y
```

Finalmente, ya que se completan las 10,000 iteraciones, el algoritmo sale del bucle e imprime el valor más pequeño que encontró al igual que los valores de xy.

```
# Impresion de los valores
print(f"Valores mínimos encontrados:")
print(f"x = {best_x}, y = {best_y}, f(x, y) = {min_value}")
```

Resultados:

```
Valores mínimos encontrados:
x = 3.1225905882897154, y = 0.5253919220321812, f(x, y) = 0.002437698906995421
PS C:\Users\Windows PC\OneDrive\Inteligencia Artificial> & "C:/Users/Windows PC/
ligencia Artificial/Lab1Busqueda.py"
Valores mínimos encontrados:
x = 2.864629692415254, y = 0.4666277563475125, f(x, y) = 0.0035072609396143172
PS C:\Users\Windows PC\OneDrive\Inteligencia Artificial>
```

Ejercicio 2: Implementación de un Juego de Gato 4x4

El segundo ejercicio consistió en programar un juego de gato en un tablero de 4x4. En este caso, la computadora utilizó búsqueda aleatoria para seleccionar una celda vacía del tablero de forma arbitraria durante su turno. La implementación permite que el algoritmo realice movimientos válidos sin seguir una estrategia definida.

Comenzamos importando las librerías necesarias para la generación de números aleatorios.

```
import numpy as np
import random
```

Definimos una función que inicializa el tablero vacío. El tablero está conformado por una lista de listas, en donde cada lista interna representa una fila del tablero. Cada casilla vacía esta representada por (“_“).

```
# Inicializacion del tablero vacio
def inicializar_tablero():
    return [[" " for _ in range(4)] for _ in range(4)]
```

Continuamos definiendo una función para imprimir el tablero en consola. Encima del tablero se imprimen las etiquetas de las columnas para que sea fácil de entender para el jugador. Después se imprime una línea separadora para poder organizar las filas del tablero.

Esto es seguido de un bucle el cuál recorre cada fila del tablero, la función de enumerate() proporciona tanto el índice de la fila como la fila completa. La siguiente línea se encarga de imprimir el tablero con sus respectivos bordes laterales y también imprimir las etiquetas de cada fila para que se pueda entender mejor el tablero. De igual forma, “.join(fila) coloca separadores entre cada elemento de la fila.

Finalmente se imprime otra línea separadora entre cada fila, esto para que el formato del tablero sea entendible.

```
# Imprimir tablero en consola
def mostrar_tablero(tablero):
    print("   0   1   2   3")
    print("  -------------")
    for i, fila in enumerate(tablero):
        print(f"{i} | " + " | ".join(fila) + " |")
        print("  -------------")
```

Continuamos con la función que se encarga de verificar si el jugador ha ganado el juego. Dentro de la función tenemos un bucle que recorre las filas y columnas. Después se verifica si todas las casillas de la fila i contienen el mismo símbolo del jugador, de igual forma se verifica si todas las casillas de la columna i contienen el mismo símbolo del jugador. Si alguna de estas condiciones es verdadera, el jugador ha ganado y la función retorna True.

Después de esta verificación, se procede a verificar si las diagonales del tablero contienen el mismo símbolo del jugador. Si se cumple alguna de las condiciones, retorna True para indicar que el jugador ha ganado.

Si ninguna de las condiciones anteriores se cumple, significa que el jugador no ha ganado, por lo que se retorna False.

```python
# Verificacion si el jugador gano
def verificar_ganador(tablero, jugador):
    # Revisa filas y columnas
    for i in range(4):
        if all([tablero[i][j] == jugador for j in range(4)]) or all([tablero[j][i] == jugador for j in range(4)]):
            return True
    # Revisa las diagonales
    if all([tablero[i][i] == jugador for i in range(4)]) or all([tablero[i][3 - i] == jugador for i in range(4)]):
        return True
    return False
```

La siguiente función es encargada de revisar si hubo un empate, el criterio de verificación es si todas las celdas están llenas en el tablero. Si todas las celdas están llenas, retorna True para indicar que el juego terminó en empate, de otra forma retorna False.

```python
# Revisa si hay empate
def verificar_empate(tablero):
    return all([tablero[i][j] != " " for i in range(4) for j in range(4)])
```

Procedemos con otra función la cuál se encarga de elegir un movimiento aleatorio para la computadora. Para poder llevar esto a cabo, se crea una lista de todos los posibles movimientos que puede hacer la computadora. Los movimientos posibles son determinados por las casillas que se encuentren vacías en el tablero. Una vez que tiene todos los posibles movimientos en la lista, se usa “random.choice()” para seleccionar el siguiente movimiento de la computadora.

```python
# Movimiento aleatorio de la computadora
def movimiento_computadora(tablero):
    movimientos_posibles = [(i, j) for i in range(4) for j in range(4) if tablero[i][j] == " "]
    return random.choice(movimientos_posibles)
```

Finalmente tenemos la función principal la cuál ejecuta el juego. Se manda a llamar a la función para iniciar un tablero nuevo. Una vez que se inicializa el tablero, se imprime en consola con la función de mostrar tablero. Después el juego entra en un bucle infinito que se ejecutará hasta que haya un ganador o un empate.

Comenzamos con el turno del jugador, se imprime un mensaje el cual pide al jugador ingresar la fila en la cual quiere colocar su símbolo. De la misma forma, procede a pedir al jugador que ingrese la columna donde quiere poner su símbolo.

Una vez que el jugador ingresa las “coordenadas” de donde quiere poner su símbolo, se verifica primero si esa casilla ya está ocupada. Si es el caso, se pide al jugador que ingrese nuevamente los valores de fila y columna.

El juego procede a colocar el símbolo en la casilla indicada y después imprime el tablero actualizado. Una vez que hace esto, verifica si el jugador ganó o si el juego terminó en un empate.

El juego sigue con el turno de la computadora, manda a llamar a la función encargada de escoger aleatoriamente el movimiento de la computadora. Procede a insertar el símbolo de la computadora en el tablero y vuelve a imprimir el tablero actualizado.

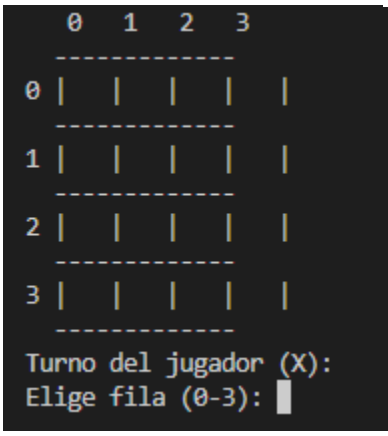
Finalmente, vuelve a hacer la verificación de ganador o empate para saber si la computadora ganó el juego o por defecto terminó en empate.

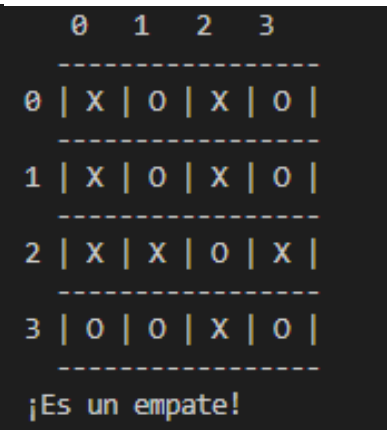
```
# Juego
def jugar():
    tablero = inicializar_tablero()
    mostrar_tablero(tablero)

    while True:
        # Turno del jugador
        print("Turno del jugador (X):")
        fila = int(input("Elige fila (0-3): "))
        columna = int(input("Elige columna (0-3): "))

        if tablero[fila][columna] != " ":
            print("Esa casilla ya está ocupada. Intenta de nuevo.")
            continue

        tablero[fila][columna] = "X"
        mostrar_tablero(tablero)

        if verificar_ganador(tablero, "X"):
            print("¡Felicidades, has ganado!")
            break

        if verificar_empate(tablero):
            print("¡Es un empate!")
            break
```

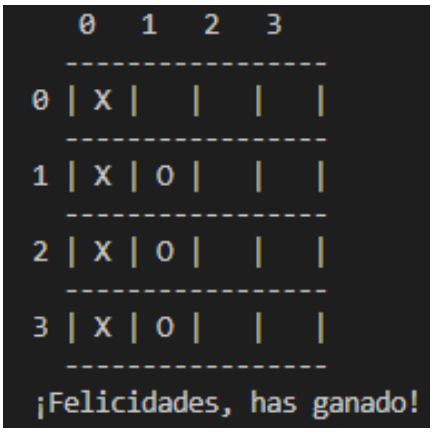
```
        # Turno de la computadora
        print("Turno de la computadora (O):")
        fila, columna = movimiento_computadora(tablero)
        tablero[fila][columna] = "O"
        mostrar_tablero(tablero)

        if verificar_ganador(tablero, "O"):
            print("La computadora ha ganado.")
            break

        if verificar_empate(tablero):
            print("¡Es un empate!")
            break

# Ejecuta el juego
jugar()
```

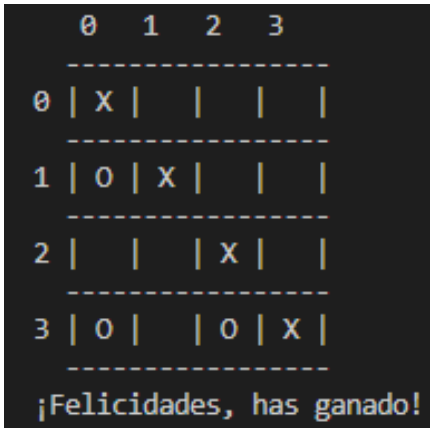
Resultados:

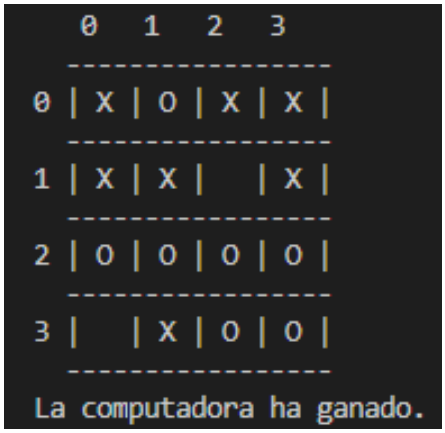
```
   0   1   2   3
 -------------
0| X | X | X | X |
 -------------
1|   |   |   |   |
 -------------
2|   | O | O |   |
 -------------
3|   | O |   |   |
```


0 1 2 3
0 | | | | |
1 | | | | |
2 | | | | |
3 | | | | |
Turno del jugador (X):
Elige fila (0-3):


0 1 2 3
0 | X | O | X | O |
1 | X | O | X | O |
2 | X | X | O | X |
3 | O | O | X | O |
¡Es un empate!


0 1 2 3
0 | X | | | |
1 | X | O | | |
2 | X | O | | |
3 | X | O | | |
¡Felicidades, has ganado!


0 1 2 3
0 | X | | | |
1 | O | X | | |
2 | | | X | |
3 | O | | O | X |
¡Felicidades, has ganado!


0 1 2 3
0 | X | O | X | X |
1 | X | X | | X |
2 | O | O | O | O |
3 | | X | O | O |
La computadora ha ganado.

Conclusiones

Aldo Alcántara Martínez

En esta práctica se implementaron dos ejercicios que destacan la versatilidad del uso de algoritmos de búsqueda aleatoria en distintos contextos. En el primer ejercicio, se logró aproximar los valores mínimos de una función matemática compleja mediante la generación de valores aleatorios dentro de un rango definido, lo que permitió explorar eficientemente el espacio de búsqueda. A través de un número elevado de iteraciones, el algoritmo fue capaz de identificar los valores mínimos de manera efectiva.

En el segundo ejercicio, la búsqueda aleatoria se utilizó en un juego de gato de 4x4 para simular los movimientos de la computadora. Aunque los movimientos no seguían una estrategia predefinida, el algoritmo fue capaz de garantizar que las jugadas de la computadora fueran válidas, demostrando que la búsqueda aleatoria puede ser aplicada incluso en escenarios lúdicos donde se requiere que las decisiones sean inmediatas y válidas.

Ambos ejercicios resaltan la utilidad de los algoritmos de búsqueda aleatoria en problemas donde el espacio de soluciones es amplio y no existe un patrón evidente para seguir. Aunque estos enfoques pueden no ser los más eficientes en términos de tiempo en todos los casos, brindan una solución sencilla y efectiva cuando se necesita explorar múltiples posibilidades.

Sebastián Reyes Núñes

En este laboratorio se exploró la aplicación de la búsqueda aleatoria en diferentes contextos, desde la minimización de funciones hasta la implementación de un juego de gato. A través de estos ejercicios, se pudo observar que, aunque la búsqueda aleatoria no es la técnica más eficiente ni garantiza la obtención de soluciones óptimas en un tiempo razonable, su simplicidad y facilidad de implementación la convierten en una herramienta útil en situaciones donde el espacio de búsqueda es poco conocido o altamente dinámico.

En el primer ejercicio, la búsqueda aleatoria permitió aproximar los valores mínimos de una función compleja mediante la evaluación de múltiples iteraciones. Aunque no se obtuvo necesariamente el valor mínimo absoluto, los resultados demostraron que esta técnica puede ser útil cuando no se dispone de información detallada sobre la estructura del problema.

El segundo ejercicio mostró cómo la búsqueda aleatoria puede emplearse en la toma de decisiones en juegos simples, como el gato, donde la computadora seleccionó movimientos de forma aleatoria. Este enfoque resalta la flexibilidad de la técnica, aunque su falta de estrategia conduce a resultados impredecibles y, en algunos casos, subóptimos.

Lagarza Ortega Ana Karen

La búsqueda aleatoria, explorada en este laboratorio a través de dos ejercicios distintos, demuestra su capacidad para abordar problemas donde no se tiene información clara sobre la estructura del espacio de soluciones. Si bien esta técnica no es la más eficiente en términos de tiempo, su simplicidad y facilidad de implementación resultan útiles, especialmente en situaciones donde se requiere explorar múltiples posibilidades rápidamente. En el primer ejercicio, su aplicación permitió aproximar los valores mínimos de una función compleja, resaltando cómo, a pesar de no siempre encontrar soluciones óptimas, el método puede ser efectivo al enfrentar problemas con restricciones poco definidas.

El segundo ejercicio, por su parte, aplicó la búsqueda aleatoria en el contexto de un juego de gato, evidenciando su utilidad incluso en escenarios lúdicos. Aunque los movimientos de la computadora no siguieron una estrategia clara, el algoritmo aseguró la validez de las jugadas, demostrando que puede ser utilizado en contextos dinámicos y no estructurados. En resumen, ambos ejercicios refuerzan la idea de que la búsqueda aleatoria, a pesar de sus limitaciones, ofrece una base para la comprensión de técnicas más avanzadas que combinan aleatoriedad con estrategias heurísticas.

Estrada Chávez Dilan Daniel

En esta práctica de laboratorio utilizamos algoritmos de búsqueda aleatoria para resolver problemas en dos contextos diferentes, en el primer ejercicio, utilizamos la búsqueda aleatoria para aproximarnos a obtener los valores mínimos en una función matemática generando una gran cantidad de valores para X y Y filtrando los valores mínimos encontrados, llegando a la conclusión que mientras mayor sea el numero de iteraciones, o en este caso, el mayor numero de valores aleatorios utilizados podremos encontrar unos valores para la función cada vez más precisos.

En el segundo ejercicio, utilizamos la búsqueda aleatoria como una función con menos importancia puesto que en el juego del gato 4x4 utilizamos la función aleatoria como una herramienta para marcar la selección de la IA en el juego, siendo su única función el escoger aleatoriamente un espacio vacio en el tablero, si bien no se podría considerar una opción decente para una IA de un juego, al menos en este ejercicio sirvió para simular un juego de gato.

Flores Vicencio Marco Isai:

Una vez finalizada la práctica, podemos realizar las siguientes conclusiones: Utilizar el algoritmo de búsqueda aleatoria para darle solución a este tipo de problemas no representa obtener la mejor solución posible, pues lo algoritmos básicos como este solo permiten obtener los mejores resultados locales. Esto se puede observar al obtener los diez mejores resultados, donde la variabilidad y precisión de resultados es mas variada que utilizando otro tipo de algoritmos,

como el de descenso de gradiente. Aunque para este tipo de problemas, su implementación resulta mucho más fácil que otro tipo de algoritmos, lo que la hace ideal para problemas que no requieren obtener los mejores resultados posibles o cuando las características del problema no permiten aplicar algoritmos de mayor complejidad.

Por otro lado, su implementación para resolver puzles como el famoso juego del gato resulta completamente ineficiente, ya que no es posible idealizar formas en las que alguno de los dos jugadores gane constantemente, dando lugar a juegos donde los movimientos y el resultados son demasiado erráticos.

Referencias

- Búsqueda aleatoria | Interactive Chaos. (s. f.). https://interactivechaos.com/es/manual/tutorial-de-machine-learning/busqueda-aleatoria

- Guerard_Guillaume. (2022, 3 diciembre). Búsqueda aleatoria: sistemas complejos e inteligencia artificial. Sistemas Complejos E IA. https://complex-systems-ai.com/es/algoritmos-estocasticos/busqueda-aleatoria/

- José Manuel Galán. (2020, 24 marzo). Introducción a las metaheurísticas (6 de 9). Búsqueda aleatoria [Vídeo]. YouTube. https://www.youtube.com/watch?v=Ohs0MkdVdMw

- ¿Qué es la simulación de Monte Carlo? - Explicación de la simulación de Monte Carlo - AWS. (s. f.). Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/monte-carlo-simulation/#:~:text=Las%20simulaciones%20de%20Monte%20Carlo%20son%20una%20t%C3%A9cnica%20matem%C3%A1tica%20que,de%20una%20elecci%C3%B3n%20de%20acci%C3%B3n.