INSTITUTO POLITECNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
NOMBRE: Cerda García Gustavo
Materia: Inteligencia Artificial

Ejercicio de Laboratorio 4. BFS, A* y SA

Ejercicio 1. BFS o A*

Empleando búsqueda informada (coste uniforme, A* o una variante) resuelve el problema del 8-puzzle.

¿Qué algoritmo utilizamos?

El algoritmo A* es un algoritmo de búsqueda informada que utiliza una función heurística para encontrar el camino más corto desde un nodo inicial hasta un nodo objetivo. En el caso del problema del 8-puzzle, el algoritmo A* puede utilizarse para encontrar la secuencia de movimientos necesarios para llegar desde una disposición inicial del tablero hasta una disposición objetivo.

¿Qué heurística utilizamos?

Se utiliza la heurística de la distancia Manhattan para calcular el costo estimado desde un estado del tablero hasta el estado objetivo. El algoritmo A* combina este costo estimado con el costo acumulado de los movimientos realizados para tomar decisiones informadas sobre qué nodos explorar primero en la búsqueda.

Código

Clase de estado del tablero

```python
# Clase para representar un estado del tablero del 8-puzzle
# Atributos:
#   - tablero (Lista): Estado del tablero
#   - movimiento (String): Movimiento que llevó a este estado
#   - costo (int): Costo del estado
#   - padre (Estado): Estado padre
class Estado:
    def __init__(self, tablero, movimiento, costo, padre):
        self.tablero = tablero
        self.movimiento = movimiento
        self.costo = costo
        self.padre = padre

    # Método para comparar dos estados por su costo
    # Se utiliza para la cola de prioridad
    # Entrada:
    #   - other (Estado): Estado a comparar
    # Salida:
    #   - bool: True si el costo de este estado es menor que el costo del otro estado, False en otro caso
    def __lt__(self, other):
        return self.costo < other.costo
```

Función `obtener_posicion_espacio(tablero)`

```python
# Función para obtener la posición del espacio en blanco en el tablero
# Entrada:
#   - tablero (Lista): Tablero del 8-puzzle
# Salida:
#   - i (int): Fila del espacio en blanco
#   - j (int): Columna del espacio en blanco
def obtener_posicion_espacio(tablero):
    for i in range(3):
        for j in range(3):
            if tablero[i][j] == 0:
                return i, j
```

Función `generar_movimientos(estado)`

```python
# Función para generar los movimientos válidos a partir de un estado del tablero
# Entrada:
#   - estado (Estado): Estado actual del tablero
# Salida:
#   - movimientos (Lista): Lista de estados resultantes de los movimientos válidos
def generar_movimientos(estado):
    movimientos = []
    i, j = obtener_posicion_espacio(estado.tablero)

    # Movimientos válidos
    # Arriba, abajo, izquierda, derecha

    # Arriba
    if i > 0:
        nuevo_tablero = [fila[:] for fila in estado.tablero]
        nuevo_tablero[i][j], nuevo_tablero[i - 1][j] = nuevo_tablero[i - 1][j], nuevo_tablero[i][j]
        movimientos.append(Estado(nuevo_tablero, "Arriba", estado.costo + 1, estado))

    # Abajo
    if i < 2:
        nuevo_tablero = [fila[:] for fila in estado.tablero]
        nuevo_tablero[i][j], nuevo_tablero[i + 1][j] = nuevo_tablero[i + 1][j], nuevo_tablero[i][j]
        movimientos.append(Estado(nuevo_tablero, "Abajo", estado.costo + 1, estado))

    # Izquierda
    if j > 0:
        nuevo_tablero = [fila[:] for fila in estado.tablero]
        nuevo_tablero[i][j], nuevo_tablero[i][j - 1] = nuevo_tablero[i][j - 1], nuevo_tablero[i][j]
        movimientos.append(Estado(nuevo_tablero, "Izquierda", estado.costo + 1, estado))

    # Derecha
    if j < 2:
        nuevo_tablero = [fila[:] for fila in estado.tablero]
        nuevo_tablero[i][j], nuevo_tablero[i][j + 1] = nuevo_tablero[i][j + 1], nuevo_tablero[i][j]
        movimientos.append(Estado(nuevo_tablero, "Derecha", estado.costo + 1, estado))

    return movimientos
```

Función `heuristica(tablero_actual, tablero_objetivo)`

```python
# Función para calcular la heurística (en este caso, distancia Manhattan)
# Entrada:
#   - tablero_actual (Lista): Tablero actual
#   - tablero_objetivo (Lista): Tablero objetivo
# Salida:
#   - distancia (int): Distancia Manhattan
def heuristica(tablero_actual, tablero_objetivo):
    distancia = 0
    for i in range(3):
        for j in range(3):
            if tablero_actual[i][j] != tablero_objetivo[i][j] and tablero_actual[i][j] != 0:
                valor = tablero_actual[i][j]
                objetivo_i, objetivo_j = divmod(valor - 1, 3)
                distancia += abs(i - objetivo_i) + abs(j - objetivo_j)
    return distancia
```

Función `resolver_8_puzzle(inicial, objetivo)`

```python
# Función para resolver el 8-puzzle utilizando A*
# Entrada:
#   - inicial (Lista): Tablero inicial
#   - objetivo (Lista): Tablero objetivo
# Salida:
#   - solucion (Lista): Lista de tuplas con el movimiento y el tablero resultante
def resolver_8_puzzle(inicial, objetivo):
    # Cola de prioridad para los estados a explorar
    frontera = []
    heapq.heappush(frontera, Estado(inicial, "", 0, None))

    # Conjunto de nodos ya explorados
    explorados = set()

    # Mientras haya nodos en la frontera
    while frontera:
        estado_actual = heapq.heappop(frontera)

        # Verificar si se ha encontrado la solución
        if estado_actual.tablero == objetivo:
            # Se ha encontrado la solución
            solucion = []
            while estado_actual.padre:
                solucion.append((estado_actual.movimiento, estado_actual.tablero))
                estado_actual = estado_actual.padre
            solucion.reverse()
            return solucion

        # Agregar el estado actual a los explorados
        explorados.add(tuple(map(tuple, estado_actual.tablero)))

        # Generar los movimientos válidos a partir del estado actual
        for movimiento in generar_movimientos(estado_actual):
            if tuple(map(tuple, movimiento.tablero)) not in explorados:
                movimiento.costo += heuristica(movimiento.tablero, objetivo)
                heapq.heappush(frontera, movimiento)

    return None
```

Función `generar_tablero()`

```python
# Función para generar un tablero con números random
# Salida:
#   - tablero (Lista): Tablero generado
def generar_tablero():
    tablero = [[0, 0, 0] for _ in range(3)]
    numeros = list(range(1, 9))
    numeros.append(0)

    for i in range(3):
        for j in range(3):
            numero = random.choice(numeros)
            numeros.remove(numero)
            tablero[i][j] = numero

    return tablero
```

Programa Principal

```python
if __name__ == "__main__":
    # Generar tablero inicial y objetivo
    estado_inicial = generar_tablero()

    estado_objetivo = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 0]
    ]

    # Imprimir tablero inicial y objetivo
    print("Tablero inicial:")
    for fila in estado_inicial:
        print(fila)
    print("**********************")

    # Resolver el 8-puzzle
    solucion = resolver_8_puzzle(estado_inicial, estado_objetivo)

    # Imprimir la solución y los movimientos
    if solucion:
        print("Solución encontrada:")
        for movimiento, tablero in solucion:
            print(f"--- Movimiento: {movimiento} ---")
            for fila in tablero:
                print(fila)
            print()
    else:
        print("No se encontró solución.")
```

Salida

Si no hay solución

En el problema del 8-puzzle, específicamente en el contexto de la búsqueda de soluciones utilizando algoritmos como A* o BFS, puede haber situaciones en las que no exista una solución válida. Estas situaciones se deben a las restricciones del problema y la configuración inicial del tablero. Algunas son inversión de

paridad, bloqueo de espacio, configuración no conectada al tablero objetivo o estado inalcanzable.

```
[Running] python -u "c:\Users\Gus\Desktop\ESCOM\6 Semestre\Inteligencia
Artificial\Practicas\IA-6CV2-GCG\Practica-4\Ejercicio-1-Puzzle-8.py"
Tablero inicial:
[1, 8, 5]
[0, 4, 7]
[3, 2, 6]
************************
No se encontr♦ soluci♦n.
```

Si existe solución

```
[Running] python -u "c:\Users\Gus\Desktop\ESCOM\6 Semestre\Inteligencia
Artificial\Practicas\IA-6CV2-GCG\Practica-4\Ejercicio-1-Puzzle-8.py"
Tablero inicial:
[8, 3, 5]
[6, 2, 1]
[4, 0, 7]
************************
Soluci♦n encontrada:
--- Movimiento 1: Arriba ---
[8, 3, 5]
[6, 0, 1]
[4, 2, 7]

--- Movimiento 2: Derecha ---
[8, 3, 5]
[6, 1, 0]
[4, 2, 7]

--- Movimiento 3: Arriba ---
[8, 3, 0]
[6, 1, 5]
[4, 2, 7]
```

```
--- Movimiento 4: Izquierda ---
[8, 0, 3]
[6, 1, 5]
[4, 2, 7]

--- Movimiento 5: Abajo ---
[8, 1, 3]
[6, 0, 5]
[4, 2, 7]

--- Movimiento 6: Izquierda ---
[8, 1, 3]
[0, 6, 5]
[4, 2, 7]

--- Movimiento 7: Arriba ---
[0, 1, 3]
[8, 6, 5]
[4, 2, 7]

--- Movimiento 8: Derecha ---
[1, 0, 3]
[8, 6, 5]
[4, 2, 7]
```

```
--- Movimiento 9: Derecha ---
[1, 3, 0]
[8, 6, 5]
[4, 2, 7]

--- Movimiento 10: Abajo ---
[1, 3, 5]
[8, 6, 0]
[4, 2, 7]

--- Movimiento 11: Izquierda ---
[1, 3, 5]
[8, 0, 6]
[4, 2, 7]

--- Movimiento 12: Abajo ---
[1, 3, 5]
[8, 2, 6]
[4, 0, 7]
```

```
--- Movimiento 13: Derecha ---
[1, 3, 5]
[8, 2, 6]
[4, 7, 0]

--- Movimiento 14: Arriba ---
[1, 3, 5]
[8, 2, 0]
[4, 7, 6]

--- Movimiento 15: Arriba ---
[1, 3, 0]
[8, 2, 5]
[4, 7, 6]

--- Movimiento 16: Izquierda ---
[1, 0, 3]
[8, 2, 5]
[4, 7, 6]

--- Movimiento 17: Abajo ---
[1, 2, 3]
[8, 0, 5]
[4, 7, 6]
```

```
--- Movimiento 18: Izquierda ---
[1, 2, 3]
[0, 8, 5]
[4, 7, 6]

--- Movimiento 19: Abajo ---
[1, 2, 3]
[4, 8, 5]
[0, 7, 6]

--- Movimiento 20: Derecha ---
[1, 2, 3]
[4, 8, 5]
[7, 0, 6]

--- Movimiento 21: Arriba ---
[1, 2, 3]
[4, 0, 5]
[7, 8, 6]

--- Movimiento 22: Derecha ---
[1, 2, 3]
[4, 5, 0]
[7, 8, 6]
```

```
--- Movimiento 23: Abajo ---
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```

El algoritmo comienza explorando el estado inicial y expandiendo los movimientos posibles desde ese estado. Luego, evalúa la heurística para cada uno de estos estados expandidos y los agrega a una cola de prioridad. En cada iteración, el algoritmo selecciona el estado con el menor costo total estimado (suma del costo acumulado y la heurística) para explorar a continuación.

Así, el algoritmo A* continúa explorando y expandiendo estados hasta encontrar el estado objetivo, garantizando que el camino encontrado sea óptimo en términos de la cantidad de movimientos necesarios para llegar al estado objetivo desde el estado inicial.

Ejercicio 2. Simulated Annealing

Empleando templado simulado (simulated annealing) encuentra el valor mínimo de las siguientes funciones:

1. $f\ (x) = x^4 + 3x^3 + 2x^2 - 1$
2. $f(x) = x^2 - 3x - 8$

Solución

Para encontrar el valor mínimo de las funciones dadas utilizando el método de templado simulado (simulated annealing), necesitamos seguir estos pasos:

1. Definir la función objetivo $f(x)$.
2. Inicializar un punto aleatorio x_0 como punto de partida.
3. Definir el rango de valores de x en el que se buscara el mínimo.
4. Definir la función de probabilidad para aceptar un nuevo punto x basado en la diferencia de valores de la función objetivo.
5. Implementar el algoritmo de templado simulado con iteraciones y un esquema para reducir la “temperatura”.

Código

Funciones para minimizar

```python
# Funciones a minimizar
# Función 1: f(x) = x^4 + 3x^3 + 2x^2 - 1
def f1(x):
    return x**4 + 3*x**3 + 2*x**2 - 1

# Función 2: f(x) = x^2 - 3x - 8
def f2(x):
    return x**2 - 3*x - 8
```

Función simulated_annealing(función, temperatura_inicial, temperatura_final, alpha, iteraciones_por_temp)

```python
# Función para encontrar el mínimo de una función utilizando Simulated Annealing
# Entrada:
#   - funcion (Función): Función a minimizar
#   - temperatura_inicial (float): Temperatura inicial
#   - temperatura_final (float): Temperatura final
#   - alpha (float): Factor de reducción de la temperatura
#   - iteraciones_por_temp (int): Número de iteraciones por temperatura
# Salida:
#   - x (float): Valor de x que minimiza la función
def simulated_annealing(funcion, temperatura_inicial, temperatura_final, alpha,
iteraciones_por_temp):
    # Punto de partida aleatorio
    x_actual = random.uniform(-10, 10)
    mejor_x = x_actual # Mejor valor encontrado
    mejor_valor = funcion(x_actual) # Valor de la función en el mejor valor encontrado

    # Temperatura inicial
    temperatura = temperatura_inicial

    # Bucle principal
    # Mientras la temperatura sea mayor que la final
    while temperatura > temperatura_final:
        # Iteraciones por temperatura
        for _ in range(iteraciones_por_temp):
            # Generar un nuevo punto cercano al actual
            nuevo_x = x_actual + random.uniform(-0.5, 0.5)

            # Calcular diferencias de valores
            delta_valor = funcion(nuevo_x) - funcion(x_actual)

            # Decidir si se acepta el nuevo punto
            # Si el nuevo punto tiene un valor menor o se acepta con una probabilidad dada por
la temperatura
            if delta_valor < 0 or random.random() < math.exp(-delta_valor / temperatura):
                x_actual = nuevo_x

                # Actualizar el mejor valor encontrado
                if funcion(x_actual) < mejor_valor:
                    mejor_x = x_actual
                    mejor_valor = funcion(x_actual)

        # Reducir la temperatura
        temperatura *= alpha

    return mejor_x, mejor_valor
```

Programa Principal

```python
if __name__ == "__main__":
    temperatura_inicial = 1000
    temperatura_final = 1
    alpha = 0.95 # Factor de reducción de la temperatura
    iteraciones_por_temp = 100 # Número de iteraciones por temperatura

    print("Encontrar mínimo de f(x) = x^4 + 3x^3 + 2x^2 - 1:")
    min_x1, min_valor1 = simulated_annealing(f1, temperatura_inicial, temperatura_final, alpha,
iteraciones_por_temp)
    print("x =", min_x1)
    print("Valor mínimo =", min_valor1)

    print("\nEncontrar mínimo de f(x) = x^2 - 3x - 8:")
    min_x2, min_valor2 = simulated_annealing(f2, temperatura_inicial, temperatura_final, alpha,
iteraciones_por_temp)
    print("x =", min_x2)
    print("Valor mínimo =", min_valor2)
```

Salida

```
[Running] python -u "c:\Users\Gus\Desktop\ESCOM\6 Semestre\Inteligencia
Artificial\Practicas\IA-6CV2-GCG\Practica-4\Ejercicio-2-SimAnn.py"
Encontrar m♦nimo de f(x) = x^4 + 3x^3 + 2x^2 - 1:
x = -1.64032390643114448
Valor m♦nimo = -1.619684335447312

Encontrar m♦nimo de f(x) = x^2 - 3x - 8:
x = 1.5001183859825011
Valor m♦nimo = -10.249999985984758
```