Memoria
Práctica 1

FUNDAMENTOS DE APRENDIZAJE AUTOMÁTICO

Diego Pareja Serrano

INTRODUCCIÓN

Esta práctica se basó en la implementación de distintos algoritmos de búsqueda y la evaluación de estos, nos centramos en el entorno del juego Pacman para representarlos. Esta práctica tiene la finalidad de que aprendamos el comportamiento y realicemos un análisis de las diferentes estrategias de búsqueda que hemos estudiado en clase.

Se estudiaron algoritmos de búsqueda no informada, concretando más, vimos la DFS (Búsqueda en profundidad), BFS (Búsqueda en amplitud), A* (Búsqueda A*). Nos dedicamos a implementar los algoritmos en el juego de Pacman para poder observar y analizar su comportamiento, rendimiento, efectividad…

DESARROLLO DE LA PRÁCTICA (Resultados algoritmos y análisis propio)

Búsqueda en profundidad (DFS)

DFS explora el espacio de estados de manera recursiva, avanzando hasta el nodo más profundo antes de retroceder, utilizando una pila (estructura LIFO). En el ejercicio se ve que aunque es rápido en ciertos casos, no garantiza encontrar la solución más corta y tiende a consumir demasiada memoria en laberintos grandes. Su exploración excesiva de caminos innecesarios lo hizo ineficiente en escenarios complejos.

Búsqueda en amplitud (BFS)

BFS garantizó la ruta más corta en términos de movimientos, pero su elevado consumo de memoria se convirtió en una limitación en laberintos de mayor tamaño. He visto que a pesar de su eficiencia en la calidad de la solución, su escalabilidad quedó comprometida por la necesidad de almacenar múltiples estados simultáneamente.

Búsqueda de costo uniforme (UCS)

UCS resultó ser útil cuando los costos de movimiento eran variables, asegurando la solución más barata. Sin embargo, en escenarios con costos homogéneos, su desempeño fue similar al de BFS con un mayor coste computacional, lo que lo hizo innecesariamente lento en algunos casos.

Búsqueda A*

A* fue el algoritmo más eficiente, reduciendo la cantidad de nodos explorados mediante una heurística informada. No obstante, su desempeño dependió en gran medida de la calidad de la heurística utilizada; cuando la heurística no era adecuada, el rendimiento de A* se acercaba al de UCS

Respuestas a preguntas del enunciado

1. **¿Es el orden de exploración lo que hubieras esperado? ¿Pacman realmente va a todas las casillas exploradas en su camino a la meta? ¿Es esta una solución de menor coste?¿Es el orden de exploración lo que hubieras esperado? ¿Pacman realmente va a todas las casillas exploradas en su camino a la meta? ¿Es esta una solución de menor coste? PREGUNTAS DEL EJERCICIO 1.**

    - El orden de exploración sigue la estrategia del algoritmo utilizado, pero en algunos casos puede parecer poco intuitivo.

- Pacman no explora todas las casillas si encuentra una solución antes, ya que los algoritmos optimizan el camino.
- No siempre es la solución de menor coste, depende del algoritmo empleado.

2. **¿BFS encuentra una solución de menor coste? Si no es así, comprueba la implementación. PREGUNTAS DEL EJERCICIO 2.**

   - Sí, BFS encuentra una solución de menor coste en términos de cantidad de movimientos, ya que expande los nodos en orden de profundidad creciente.

3. **¿Qué sucede en openMaze para las diversas estrategias de búsqueda? PREGUNTA DEL EJERCICIO 4.**

   - En *openMaze*, las estrategias de búsqueda exploran grandes espacios sin obstáculos, lo que puede hacer que algunos algoritmos sean ineficientes en términos de exploración y tiempo de cómputo.

Complicaciones y sus soluciones

Manejo ineficiente de estructuras de datos en DFS: Inicialmente, la implementación de DFS no verificaba si un nodo ya había sido explorado, lo que provocaba ciclos y un uso innecesario de memoria. Se resolvió implementando una estructura de conjunto (set()) para almacenar los nodos visitados y evitar repeticiones.
Dificultades en UCS al manejar nodos ya visitados: Al principio, la implementación de UCS no revisaba si un estado ya explorado podía haber encontrado un camino más barato. Esto resultó en expansiones innecesarias y tiempos de ejecución más altos. Se corrigió verificando si el costo acumulado era menor antes de actualizar la estructura visitados.
Problemas con la actualización de prioridades en A*: En versiones iniciales, la prioridad en la cola de A* no se actualizaba correctamente cuando un estado ya visitado encontraba un camino más barato. Se solucionó permitiendo reinsertar el nodo con una prioridad menor en la cola en lugar de descartarlo.

CONCLUSIÓN

Esta práctica me permitió comprender cómo la elección del algoritmo de búsqueda influye en la eficiencia y calidad de las soluciones obtenidas. Observé que no existe un algoritmo universalmente mejor, sino que la elección depende del contexto y las restricciones del problema.
Uno de los aprendizajes más valiosos fue la importancia del equilibrio entre exploración y eficiencia computacional. Mientras que BFS y UCS garantizan soluciones óptimas, pueden ser costosos en términos de memoria y tiempo. A* destacó por su capacidad de reducir el número de nodos explorados, pero su éxito depende de la calidad de la heurística empleada.
Además, entendí que la heurística es un factor clave en la optimización de la búsqueda. Un diseño adecuado de la heurística puede hacer que A* sea altamente eficiente, mientras que una mala elección puede hacer que pierda su ventaja.