Seguimiento 3

Seguimiento tarea 3

Daniel Tomas Gallego

Objetivo:

Esta tarea consiste en escribir una búsqueda general para encontrar la ruta que visita todas las ciudades en un grafo utilizando diferentes estrategias (búsqueda en amplitud, búsqueda en profundidad y costo uniforme). Para ello, se necesita implementar una capa de clases que maneje la frontera, los nodos en el árbol de búsqueda y el conjunto de estados visitados, y combinarla con las estructuras de preguntas anteriores.

Detalles del código utilizado:

Clase Frontera (frontera.py):
• Clase Frontera que mantiene nodos para el árbol de búsqueda utilizando una cola de prioridad: • Emplea heapq para una inserción ordenada eficiente por el valor del nodo, o resta el id del nodo si hay un empate.
• Métodos: insert (inserta un nodo), extract (devuelve un nodo con el valor más bajo), is_empty (si está vacío o no) y size (número de nodos).
Clase Nodo (nodo.py):
• La clase Node es una clase que representa un nodo en el árbol de búsqueda: • Atributos: id (único, incremental), parent, state, action, cost, depth, heuristic y value.
• Métodos: state_hash (devuelve los últimos 6 caracteres del hash MD5 del estado), **str** (representación textual en el formato especificado), path (reconstruir el camino desde el nodo raíz) y **lt** (comparación para ordenar en la frontera).

Clase VisitedStatesSet (conjuntoestadosvisitados.py):
• Esta clase, al igual que la otra, controlará los estados visitados para no entrar en ciclos:
• Se basa en un diccionario (states) para rastrear los hashes MD5 de los estados con un enfoque en la velocidad de pertenencia (contains) e inserción (add).

Clase Problema (problema.py):
• Las dependencias del problema están encapsuladas por la clase Problem:
• Atributos latentes: initial_state, adjacency_list, is_goal_func y heuristic_func.
• Métodos: is_goal (método para verificar si un estado es objetivo), calculate_heuristic (para calcular la heurística, si existe) y get_successors (para generar sucesores). La lista de adyacencia a utilizar es frija.tripathyQueries1.

Clase algoritmoBusqueda (algoritmoBusqueda.py):
• La búsqueda general se realiza de la siguiente manera:
• Inicia la construcción de la Frontera y el Conjuntoestadosvisitados, añade el nodo inicial y maneja el bucle de búsqueda.
• La computación del valor del nodo se realiza en función de la estrategia (búsqueda en amplitud: depth, búsqueda en profundidad: 1/(depth+1), costo uniforme: cost).
• Si los nodos no son más profundos que la profundidad máxima y no han sido visitados, se expanden extendiendo nodos mientras se actualizan la frontera y los nodos visitados.
• Devuelve la secuencia de estados para llegar al objetivo si se encuentra, o None en caso contrario.

Integración (main.py):
• El script main.py, que coordina la ejecución:
• Carga el grafo con parse_graphml desde creargrafo.py.
• Especifica un punto de partida y una heurística básica (el número de nodos que deben ser visitados).
• Genera un objeto Problem y llama al SearchAlgorithm utilizando una estrategia específica (por ejemplo, costo uniforme).
• Muestra el camino hacia la solución, o un mensaje si no hay solución.

Destacados:

• Optimización: La Frontier utilizará una cola de prioridad para las inserciones ordenadas, y el VisitedStatesSet utilizará un diccionario para tener operaciones rápidas.
• Flexibilidad: Esta implementación es genérica, puede adaptarse utilizando muchas estrategias de búsqueda (como A* no incluida en el código actual).
• Robustez: El manejo de estados visitados evita bucles y el límite de profundidad restringe la exploración excesiva.
• Integración: Las clases se integran con estructuras existentes en tareas anteriores (State, GraphMLHandler), asegurando compatibilidad.

Conclusión:

• En la Tarea 3 desarrollas un algoritmo de búsqueda general, que resuelve el problema de encontrar caminos en un grafo que describe Ciudad Real, utilizando representaciones eficientes que incluyen Frontera, Nodo y Conjuntosestadosvisitados. La conexión con el grafo y los estados definidos en tareas anteriores busca soluciones óptimas basadas en diferentes estrategias tanto con el grafo como con los estados, para cumplir con los requisitos del laboratorio.