Assignment 3: Symbolic Planning

Name: Aman Manish Chulawala
Andrew ID: `achulawa`

1 Algorithm Discussion

The Planner class is a C++ implementation for planning in dynamic environments using the A* search algorithm. To use the planner, instantiate an Env object representing your planning environment and initialize the Planner with it.

The preprocessing method is crucial as it generates all grounded actions based on the defined symbolic actions and symbols in the environment. This step prepares the planner for efficient A* search.

After preprocessing, call the Astar method to execute the A* search. This method explores the state space, considering valid grounded actions and applying heuristics. If a goal state is reached, the optimal path is stored in the path stack.

Customization is possible by adjusting the heuristic type (whichHeur) and extending or modifying methods for specific planning needs. Dependencies include the Env class and standard C++ data structures.

1.1 Preprocessing

The `preprocessing` method plays a crucial role in preparing the `Planner` for efficient A* search. It involves the following steps:

1.1.1 Retrieve Actions and Symbols:

- Obtain all possible actions (`allActions`) from the planning environment (`Env`).
- Extract symbols from the environment and store them in `allSymbols`.

1.1.2 Generate Combinations and Permutations:

- For each action, generate all combinations of its arguments.
- Generate permutations for each combination to cover all possible grounded action instances.
- Populate `allGroundedActions` with the resulting grounded actions.

1.1.3 Heuristic Calculation Support:

- The method sets up the groundwork for heuristic calculations by populating `allGroundedActions` with all possible combinations of action instances.
- Heuristic 1 : Number of Missing Literals in the state
- Heuristic 2 : Empty-Delete-List Heuristic
- Heuristic 3 : Set heuristic of each state to 0

1.2 A* Search (Astar)

The `Astar` method executes the A* search algorithm to find the optimal path from the initial state to the goal state. The process involves:

1.2.1 Initialize Open and Closed Lists:

- Maintain an open list, sorted by the estimated total cost (*f-value*), and a closed list to track expanded states.

1.2.2 Initialize Initial Node:

- Create a starting node representing the initial state.
- Compute a hashed string representation of the initial state (*nodeStr*).
- Add the initial node to the open list.

1.2.3 Expand States:

- While the open list is not empty, select the node with the lowest total cost (*f-value*) for expansion.
- Check if the state represented by the node satisfies the goal conditions. If yes, the search is complete.

1.2.4 Generate Valid Grounded Actions:

- For each grounded action, check if its preconditions are satisfied by the current state.
- If satisfied, apply the grounded action to generate a new state.

1.2.5 Update Node Information:

- Update the node information (parent, cost, heuristic, etc.) for the new state if it has not been encountered before or if a lower-cost path is found.

1.2.6 Add to Open List:

- Add the new state to the open list for further exploration.

1.2.7 Repeat:

- Repeat the process until the goal state is reached or the open list is exhausted.

1.2.8 Backtrack for Optimal Path:

- If the goal state is reached, backtrack from the goal state to the initial state, storing the optimal path in the `path` stack.

The `Astar` method leverages heuristics, defined by the `getHeuristic` method, to guide the search efficiently.

2 Results

The tables below summarises the performance of the planners for 4 different domains.

Heuristic	Time to Plan (s)	Number of Expansion	Length of Path
No. of Missing Literals	0.0565	13	3
Empty Delete List	0.020373	12	3
Heuristic 0	0.009471	16	3

Table 1: Statistics of **Blocks.txt** domain

Heuristic	Time to Plan (s)	Number of Expansion	Length of Path
No. of Missing Literals	0.171906	210	6
Empty Delete List	1.89551	112	6
Heuristic 0	2.6131	2211	6

Table 2: Statistics of **BlocksTriangle.txt** domain

Heuristic	Time to Plan (s)	Number of Expansion	Length of Path
No. of Missing Literals	0.367829	380	21
Empty Delete List	548.568	292	21
Heuristic 0	0.418082	392	21

Table 3: Statistics of **FireExtinguisher.txt** domain

Heuristic	Time to Plan (s)	Number of Expansion	Length of Path
No. of Missing Literals	0.002156	4	3
Empty Delete List	0.002964	4	3
Heuristic 0	0.00211	4	3

Table 4: Statistics of **Custom.txt** domain

3 Discussion

In the context of A* search, three heuristics are employed: Heuristic 1 counts the number of missing literals, Heuristic 2 utilizes the Empty Delete List (EDL) heuristic by tracking deleted literals, and Heuristic 3 sets all heuristic values to 0. Heuristic 2 outperforms Heuristics 1 and 3 in terms of the number of nodes expanded, indicating its effectiveness in guiding the search efficiently. However, the computational expense of Heuristic 2 is higher due to the complexity of maintaining the list of deleted literals. Heuristic 1, though simpler, may result in a larger number of expanded nodes, and Heuristic 3, while computationally efficient, sacrifices the benefits of heuristic guidance, potentially leading to a less efficient search. The choice of heuristic involves a trade-off between computational efficiency and the heuristic's ability to guide the search effectively toward the goal state.

I would prefer to use Heuristic 1. It is not as time consuming, and only expands a marginally more number of nodes.

4 Extra Credit

4.1 Custom Domain

A custom domain was created and is present with the name custom.txt in the folder along with the other domains. The domain defines a robot which needs to go from room A to room B. The door between the rooms is locked and should remain locked except when the robot is crossing the rooms. This initial state and goal state are defined in the domain, along with the possible actions, preconditions and effects. The domain can be executed by just calling the executable with this domain as an input, like the other domains.

4.2 Empty Delete Heuristic

The Empty-Delete-heuristic has been implemented in the planner. You can call the planner with the Empty Delete Heuristic by changing the `whichHeur` to 2. This causes the planner to use the EDL heuristic. The Heuristic implementation makes it inherently slower, but reduces the amount of nodes which are expanded.