Assignment-3

Uniform cost search:

Uniform cost search is an algorithm we use to find the minimum cost to reach the goal state from the start state.

Unlike breadth-first search, which explores nodes level by level, Uniform Cost Search prioritizes nodes based on the accumulated cost to reach them, which means it expands its search along the minimum cost.
In uniform search we create a priority queue i.e. minimum heap which gives priority for minimum cost.

nodes are organized based on their path costs. At each step, the algorithm extracts the node with the
lowest cost from the priority queue, expanding its successors and updating their path costs if a lower-cost path is discovered.
While Uniform Cost Search guarantees optimality by always selecting the lowest-cost path, its time complexity can be
high due to the need to store and manage the priority queue efficiently.
Additionally, if the search space contains
a large number of states or if there is a high variation in edge costs, Uniform Cost Search might explore a large portion
of the search space before reaching the goal.

How to solve:

First we take priority queue and add our state state, initial cost i.e 0 and its parent initially it will be none.

Now we apply a loop until the priority queue gets empty. Now we take the top priority element i.e our current state if our current state is our goal node then we create a path list where we add all corresponding parent nodes until our start state and we reverse the path list.

If our current state is not our goal state then If the current state has not been visited yet, mark it as visited and proceed.

Generate the successor states by applying valid actions (moves) to the current state. Now we calculate the cost of reaching each successor state by adding the action cost to the current path cost.

If the state is not in the visited set, enqueue it into the priority queue with its calculated cost.

If the state is already visited but the newly calculated cost is lower than the previously recorded cost, update the cost in the priority queue.

If the priority queue becomes empty without finding the goal state, the algorithm concludes that there is no solution to the problem.

Iterative deepening search:

Iterative deepening search is an approach to traversing graphs that strikes a balance between the dfs and bfs.instead of diving straight into deepest levels like dfs or fully exploring each level like bfs, iterative deepening employes series of dfs and bfs.it progressively increases the depth with each iteration until our goal reached.By delving deeper with each cycle, IDS retains the memory efficiency of depth-first search while also ensuring that optimal solutions are found.

How to solve:

Initially we set depth limit to 0 and then we perform dfs starting from our initial node until our current depth if we find our solution then we just return it.otherwise we just increment the depth by 1 in each iteration and repeat the same process.if we reach both level and if we don’t find our goal state then just conclude that there is no solution.

Completeness:

Both IDS and UCS are complete algorithms, meaning that if a solution exists within the search space, both algorithms will find it.

Optimality:

Iterative Deepening Search: IDS is not guaranteed to find the optimal solution if the edge costs are not uniform (i.e., not all edges have the same cost). It repeatedly explores the same nodes at different depth levels, which can lead to suboptimal solutions.

Uniform Cost Search: UCS is optimal when all edge costs are non-negative. It always expands the lowest-cost nodes first, ensuring that the shortest path to a solution is found.

Time Complexity:

Iterative Deepening Search: The time complexity of IDS is high compared to other search algorithms. In the worst case, it may visit the same nodes multiple times across different depth levels, resulting in a time complexity of O(b^d), where b is the branching factor and d is the depth of the shallowest solution.

Uniform Cost Search: The time complexity of UCS depends on the number of nodes and the cost of the optimal path. In the worst case, where edge costs are not bounded, UCS can be quite slow, but with reasonable cost bounds, it can be more efficient than IDS.

Space Complexity:

Iterative Deepening Search: IDS has a relatively low space complexity of O(bd) since it only needs to store nodes at the current depth level.

Uniform Cost Search: UCS can have a high space complexity due to the need to maintain a priority queue to store nodes based on their costs. In the worst case, it can be O(b^d) for both time and space complexity.

Edge Costs:

Iterative Deepening Search: IDS does not take into account the cost of edges while searching.

Uniform Cost Search: UCS takes edge costs into consideration and always explores the lowest-cost path first.

In summary, while both Iterative Deepening Search and Uniform Cost Search have their merits, Uniform Cost Search is generally preferred when edge costs are available and well-defined, as it guarantees optimality. Iterative Deepening Search is often used in scenarios where memory constraints are an issue or when the edge costs are not well-defined or uniform.

<u>Group members:</u>

Gnaneshwar 2001CS15

Devanand 2001CS19

Teja vardhan 2001CS26