

hw1_search_q12_memory_efficient_graph_search

Question 12: Memory Efficient Graph Search

8/8 points (ungraded)

Recall from lecture the general algorithm for GRAPH-SEARCH reproduced below.

```
function GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe, strategy)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe ← INSERT(child-node, fringe)
      end
    end
  end
```

Using GRAPH-SEARCH, when a node is expanded it is added to the closed set. This means that even if a node is added to the fringe multiple times it will not be expanded more than once. Consider an alternate version of GRAPH-SEARCH, MEMORY-EFFICIENT-GRAPH-SEARCH, which saves memory by (a) not adding node n to the fringe if STATE[n] is in the closed set, and (b) checking if there is already a node in the fringe with last state equal to STATE[n]. If so, rather than simply inserting, it checks whether the old node or the new node has the cheaper path and then accordingly leaves the fringe unchanged or replaces the old node by the new node.

By doing this the fringe needs less memory, however insertion becomes more computationally expensive.

More concretely, MEMORY-EFFICIENT-GRAPH-SEARCH is shown below with the changes highlighted.

```

function MEMORY-EFFICIENT-GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe, strategy)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe ← SPECIAL-INSERT(child-node, fringe, closed)
      end
  end

```

```

function SPECIAL-INSERT(node, fringe, closed) return fringe
  if STATE[node] not in closed set then
    if STATE[node] is not in STATE[fringe] then
      fringe ← INSERT(node, fringe)
    else if STATE[node] has lower cost than cost of node in fringe reaching STATE[node] then
      fringe ← REPLACE(node, fringe)
  end if

```

Now, we've produced a more memory efficient graph search algorithm. However, in doing so, we might have affected some properties of the algorithm. Assume you run MEMORY-EFFICIENT-GRAPH-SEARCH with the A* node expansion strategy and a consistent heuristic, select all statements that are true.

☒ The EXPAND function can be called at most once for each state.

☒ The algorithm is complete.

☒ The algorithm will return an optimal solution.



Submit

✓ Correct (8/8 points)