hw1_search_q10_early_goal_checking_graph_search

Question 10: Early Goal Checking Graph Search

0.0/2.0 points (graded)

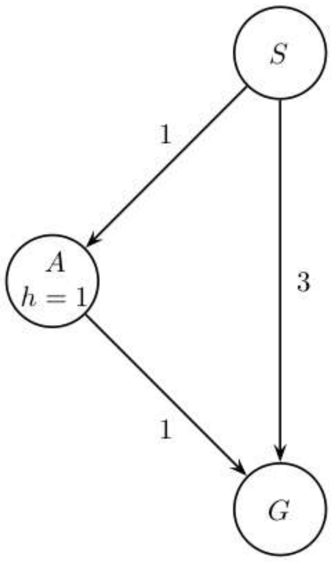
Recall from lecture the general algorithm for GRAPH-SEARCH reproduced below.

```
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
```

With the above implementation a node that reaches a goal state may sit on the fringe while the algorithm continues to search for a path that reaches a goal state. Let's consider altering the algorithm by testing whether a node reaches a goal state when inserting into the fringe. Concretely, we add the line of code highlighted below:

```
function EARLY-GOAL-CHECKING-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
                if GOAL-TEST(problem, STATE[child-node]) then return child-node
                fringe ← INSERT(child-node, fringe)
            end
    end
```

Now, we've produced a graph search algorithm that can find a solution faster. However, In doing so we might have affected some properties of the algorithm. To explore the possible differences, consider the example graph below.



If using EARLY-GOAL-CHECKING-GRAPH-SEARCH with a Uniform Cost node expansion strategy, which path, if any, will the algorithm return?

- [✓ S-G] ✓
- S-A-G
- EARLY-GOAL-CHECKING-GRAPH-SEARCH will not find a solution path.

The fringe starts out with the start state, S. We expand S. The successors of S are S-A and S-G. The path S-G passes the goal test, so the algorithm returns S-G.

Submit

Answers are displayed within the problem

problem

0.0/2.0 points (graded)

If using EARLY-GOAL-CHECKING-GRAPH-SEARCH with an A* node expansion strategy, which path, if any, will the algorithm return?

- [✓ S-G] ✓
- S-A-G
- EARLY-GOAL-CHECKING-GRAPH-SEARCH will not find a solution path.

Like in the previous question, the fringe starts out with the start state, S. We expand S. The successors of S are S-A and S-G. The path S-G passes the goal test, so the algorithm returns S-G.

Submit

Answers are displayed within the problem

problem

0.0/2.0 points (graded)

Assume you run EARLY-GOAL-CHECKING-GRAPH-SEARCH with the Uniform Cost node expansion strategy, 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.

The solutions for this question are in the next part of this question.

Submit

Answers are displayed within the problem

problem

0.0/2.0 points (graded)
Assume you run EARLY-GOAL-CHECKING-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.

For both UCS and A* with EARLY-GOAL-CHECKING-GRAPH-SEARCH, the first two choices are correct. The EXPAND function can be called at most once for each state because, like in 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. Completeness is also guaranteed, meaning that if there exists a solution to the search problem, the algorithm will find it. Adding the goal-check condition early does not change the completeness of the algorithm.

However, optimality is not guaranteed. The bug is that EARLY-GOAL-CHECKING-GRAPH-SEARCH checks if the *child-node* satisfies the goal test at the time of insertion onto the fringe, rather than at the time the node is popped from the fringe. As a consequence of this bug, a suboptimal path can be returned.

For the special case of breadth-first graph search this modification will not affect its property of finding a plan that reaches a goal state that requires a minimal number of actions while at the same time giving a minor improvement in running time for breadth-first graph search. (Think about why this is the case!)

Keep in mind, however, that for consistency across DFS, BFS, UCS, and A* in this class we use the generally applicable version of graph search presented in lecture. For the projects you should use the version presented in lecture (reproduced at the top of this page).

Submit

Answers are displayed within the problem