CSE 5525 Artificial Intelligence II

Homework #1: A* Search and Minimax

Wei Xu, Ohio State University

Your Name: ____________________ OSU Username: ______________

1. (**3 points**) Assume we run $\alpha - \beta$ pruning expanding successors from left to right on a game with tree as shown in Figure 1 (a). Then we have that:

   (a) (*true* or *false*) For some choice of pay-off values, no pruning will be achieved (shown in Figure 1 (a)).

   (b) (*true* or *false*) For some choice of pay-off values, the pruning shown in Figure 1 (b) will be achieved.

   (c) (*true* or *false*) For some choice of pay-off values, the pruning shown in Figure 1 (c) will be achieved.

   (d) (*true* or *false*) For some choice of pay-off values, the pruning shown in Figure 1 (d) will be achieved.

   (e) (*true* or *false*) For some choice of pay-off values, the pruning shown in Figure 1 (e) will be achieved.

   (f) (*true* or *false*) For some choice of pay-off values, the pruning shown in Figure 1 (f) will be achieved.

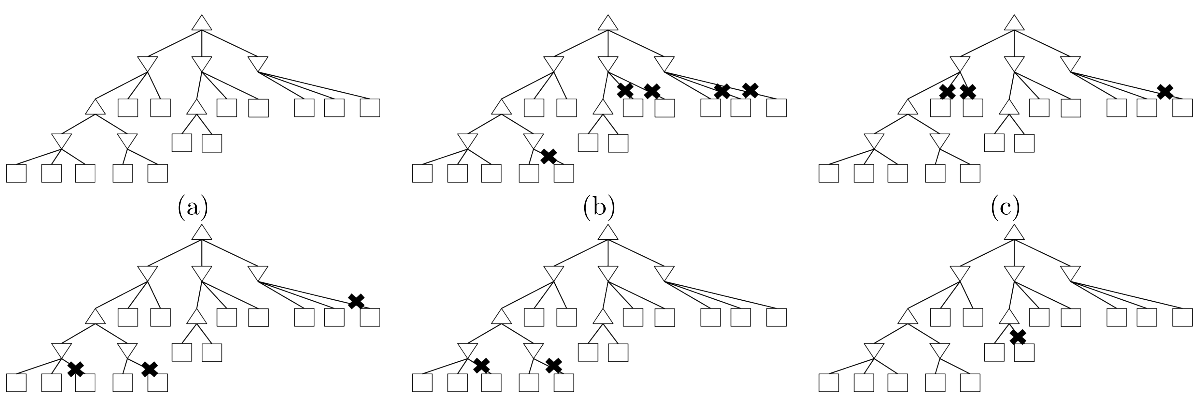

Figure 1: Game trees.

2. The following implementation of graph search may be incorrect. Circle all the problems with the code.

```
function GRAPH-SEARCH(problem, fringe)
    closed ← an empty set,
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop
        if fringe is empty then
            return failure
        end if
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem,STATE[node]) then
            return node
        end if
        ADD STATE[node] TO closed
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end loop
end function
```

This algorithm is equivalent to tree search, rather than graph search.

In graph search, nodes added to the "closed" list should not be expanded again.

(a) Nodes may be expanded twice.

(b) The algorithm is no longer complete.

(c) The algorithm could return an incorrect solution.

(d) None of the above.

3. (**2 points**) The following implementation of A* graph search may be incorrect. You may assume that the algorithm is being run with a consistent heuristic. Circle all the problems with the code.

```
function A*-SEARCH(problem, fringe)
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop
        if fringe is empty then
            return failure
        end if
        node ← REMOVE-FRONT(fringe)
        if STATE[node] IS NOT IN closed then
            ADD STATE[node] TO closed
            for successor IN GETSUCCESSORS(problem, STATE[node]) do
                fringe ← INSERT(MAKE-NODE(successor), fringe)
                if GOAL-TEST(problem,successor) then
                    return successor
                end if
            end for
        end if
    end loop
end function
```

When should A* terminate?

Should we stop when we enqueue a goal?

No. only stop when we dequeue a goal.

Otherwise, like the stated algorithm here, it expands fewer nodes to find a goal.

(a) Nodes may be expanded twice.

(b) The algorithm is no longer complete.

(c) The algorithm does not always find the optimal solution.

(d) None of the above.

(a) Recall that, a search algorithm is **complete**, if whenever there is at least one solution, the algorithm is guaranteed to find it withing a finite amount of time. A search algorithm is **optimal** if when it finds a solution, it is guaranteed to be the best one (e.g. the least cost).

The word "incorrect" means "not optimal" or "suboptimal" here. The given incorrect graph search algorithm never checked whether the node is in *closed* (i.e. explored before), thus it is effectively doing tree search. Tree search could not return an "suboptimal solution" – when tree search returns any solution it will be the best optimal solution; however, it could possibly (e.g. depth-first tree search) not return any solution at all if stuck in infinite loops. Comparing to tree search, graph search will not only eliminate redundant paths but also avoid infinite loops.

(b) The correct implementation of generic graph search and A* graph search looks as follows. By "generic", it means that for depth-first (a stack - last in first out), breadth-first (a queue - last in last out), uniform cost (a priority queue), and A* tree search (a priority queue; also need heuristics), the only difference is what you use to implement the fringe; A* search in addition considers heuristics.

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

function A*-GRAPH-SEARCH(problem, fringe, Heuristic)
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop
        if fringe is empty then
            return failure
        end if
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem,STATE[node]) then
            return node
        end if
        if STATE[node] IS NOT IN closed then
            ADD STATE[node] TO closed
            for successor IN GETSUCCESSORS(problem, STATE[node]) do
                h ← Heuristic(successor, problem)
                fringe ← INSERT(MAKE-NODE(successor, h), fringe)
            end for
        end if
    end loop
end function
```