CS 534 Homework Assignment#2

-Sailesh Rajagopalan

1. (3.13)

   We understand from the graph separation property which states that "every path from the initial state to an unexplored state has to pass through a state in the frontier." Every path from the initial state to an unexplored state will include a node in the frontier. So it will have to go through. Beginning of the search the frontier has the initial state so for an agent to be able to go to a new state it must first go through the frontier. This property will be true for graph search since as we complete the iterations and ho through nodes in search of the goal state the frontier node will always exist before being able to go to an unexplored node or state.
   The graph search according to its property every path from an initial state to the unexplored state has a node in the frontier, we can violate the algorithm, when we move the nodes without getting all the possible successors, it will be possible to only be able to generate certain nodes in order to get from a to b path which would violate the graph separation property.

2. (3.16)
   a. Initial State: a piece that is selected arbitrarily
      Succer Function: We can access it by a multiple step approach, we can do so by connecting pieces to any remaining piece. The curved pieces can be added "in either orientation" for forks too; it can be the same. So ideally we will add depending upon the kind of hole it will add a piece from what's left. It is better to also disallow overlapping configurations.
      Goal Test: All the pieces are used in the connected track and the usage of every piece there is.
      Step Cost: It will be the usage of every piece.
   b. Since in this scenario we have a big state space we should do depth first search to solve so that we can achieve the results as a search of depth. Comparing it to iterative search or breadth first or has unnecessary steps included with it.
   c. We require an equal number of pegs and holes to be able to do it, the solution has no open pegs or holes for that reason. IF we remove a fork it will not follow the property of it. Considering a fork creates two tracks which will make the forks the only possible way to rejoin them, removing a fork will not work in that. So we need that to have an ending to the track.
   d. There are 3 open pegs and would be if there is an open peg with fork added to the initial ones. The space created due to it will lead us to 56 total choices. Which is 168 for each 3 open pegs. Since there will be pieces that used twice it will have $168^{32}/(12! \,.\, 16! \,.\, 2! \,.\, 2!)$

3. (3.26)
   a. The branching factor b in this state space is 4
   b. There will be 4k distinct states at the depth k

c. The maximum number of nodded expanded by breadth first tree search is 4^{x+y}
d. The maximum number of nodes expanded by breadth first graph search is $2(x + y)(x + y + 1)$
e. Yes it is the distance between two points which is by the absolute difference between the points in the coordinate system.
f. The heuristic is perfect so it is $x + y$ the number of nodes are expanded by A* graph search using h
g. Yes, If we remove some links the path length will increase making the heuristic underestimate.
h. No, if we add links then the decrease in the path lengths and heuristic won't be an underestimate.

4. (4.3)

Separate file

5. (4.5)

function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan, or failure
OR-SEARCH(problem.INITIAL-STATE, problem, [ ])

function OR-SEARCH(state, problem, path) returns a conditional plan, or failure
if problem.GOAL-TEST(state) then return the empty plan
if state has previously been solved then return RECALL-SUCCESS(state)
if state has previously failed for a subset of path then return failure
if state is on path then
RECORD-FAILURE(state, path)
return failure
for each action in problem.ACTIONS(state) do
plan ← AND-SEARCH(RESULTS(state, action), problem, [state | path])
if plan ≠ failure then
RECORD-SUCCESS(state, [action | plan])
return [action | plan]
return failure

function AND-SEARCH(states, problem, path) returns a conditional plan, or failure
for each s_i in states do
$plan_i \leftarrow$ OR-SEARCH(s_i , problem, path)
if plani = failure then return failure
return [if s_1 then $plan_1$ else if s_2 then $plan_2$ else . . . if s_{n-1} then$plan_{n-1}$ else $plan_n$]

The and-or-graph-search every state will have it which can both be a solved state or a failed state. The or-search if it finds the answer it will be able to come back to it again and it will return the solution back. If the or-search fails then to the solution it will be based on the path. The failure is recorded into path and it is the subset of its value which will return us a failure.

To avoid repetition of the sub solutions we can label and record them and we can come back and return for them.