Eric Rock

CS510 - 001

HW 1

Small Towers of Hanoi:

For the Tower of Hanoi problem, given the problem definition, we posit the following specification:

Performance Measurement:

The performance measure will be specified as a cost function of 1 unit per action, intending to reward agents which provide a solution utilizing as few moves as possible.

Environment:

The environment consists of three rods, named R_1, R_2, and R_3; as well as three disks named D_1, D_2, and D_3, sorted by increasing size (such that D_1 is the smallest disk, and D_3 is the largest). The initial state is known to the agent (disks D_3, D_2, and D_1 are located on rod R_1, with D_3 on the bottom and D_1 on top). The effect of applying an action <D, R> is to remove disk D from its rod and to place it upon disk R. The environment may be represented by a 3-tuple of stack data structures, whereby the first element of the tuple will correspond to the stack representing R_1, the second will represent R_2, and the third will represent R_3. Performing a pop operation on any stack will remove the top disk from that stack, and insertion onto any stack will place the new disk at the top of the stack. Finally, disks shall be represented by the integer value of their subscript, such

that D_1 will be represented by the integer 1, and will stand for the smallest disk. Therefore, applying action $<D_1, R_2>$ upon the initial state will peek at each stack to determine the stack containing D_1, pop that stack, peek at the top element of R_2 to ensure that its value is smaller than 1 (or it is empty), and push D_1 on top of the stack representing R_2.

Actions:

An action is specified as a 2-tuple consisting of a disk D and rod R (i.e. $<D_1, R_2>$) which involves removing disk D from its current rod and placing it upon rod R. In any given state, only actions which involve placing a disk upon a rod containing either no disks or a larger disk are valid (i.e. if R_2 contains D_1, no move specifying R_2 as a destination is valid, as there are no disks smaller than D_1). Similarly, if rod R does not contain disk D, or if disk D is not the disk at the top of rod R, the move is also invalid.

Sensors:

The agent is able to perceive the number and ordering of each disk upon each rod.

Pac Man:

For our simplified version of Pac-Man, containing only a rectangular grid consisting of walls, pellets, our intrepid (and hungry) explorer Pac Man, or nothing, we may specify the problem as follows:

Performance Measurement:

The performance measure for this problem will consist of awarding 2 points for every pellet which is consumed by Pac Man, and removing 1 point for each movement where Pac Man does not eat a pellet, intending to reward agents which eat as many pellets using as few moves as possible.

Environment:

The environment consists of an N x M grid consisting of walls, pellets, empty space, or the agent (represented by Pac Man). The initial state will consist of some configuration of walls, pellets, empty space, and Pac Man such that there is at least one cell containing a pellet, one cell containing Pac Man, at least one path leading from Pac Man to any cell containing a pellet, and walls on all frontier cells (cells which are not surrounded on all four sides by other cells in the grid, such as position (0, 0)). We may represent a state by enumerating each cell type as an integer value, with empty space designated as 0, walls designated as 1, pellets designated as 2, and Pac Man designated as 3. Therefore, a given state may be represented by an N x M matrix consisting of integer values between 0 and 3, inclusive. An action, furthermore, will be represented by an enumeration designating UP as the integer 0, DOWN as the integer 1, LEFT as the integer 2, and RIGHT as the integer 3. The effect of applying an action to a given state will be to search for the coordinates of the cell containing Pac Man within the state, determine the direction of travel indicated by the action, ensure that the cell 1 unit in the direction of travel is either empty or contains a pellet,

replacing the cell which contains Pac Man with the value of 0 (empty), and replacing the cell 1 unit in the direction of travel from Pac Man's previous position with 3 (Pac Man).

Actions:

An action is specified as an enumerated value specifying UP, DOWN, LEFT, or RIGHT. In a given state, only actions which will involve moving Pac Man into a space which is either empty or contains a pellet will be valid. For example, if Pac Man is surrounded by walls above, to the right, and below, the only valid move for Pac Man in this state is to move LEFT. The effect of applying a valid action to a state is to replace the contents of the cell 1 unit in the direction specified by the action from Pac Man with Pac Man, and to clear the contents of the cell Pac Man was in previously.

Sensors:

We assume that an agent is equipped with sensors sufficient enough to perceive the entire state of the grid, including the location of all walls, all pellets, and the agent (represented by Pac Man) on the grid.

Forward Backward Search:

The problem of determining whether Eloise is, in fact, a direct descendant of Benjamin Franklin may be restated as the problem of determining whether Benjamin Franklin is a proper ancestor of Eloise in the tree of human ancestry. A solution to this problem might involve either searching, starting from the node containing Eloise within the human ancestry tree, for

Benjamin Franklin among all of the ancestors of Eloise (That is, only considering the parents of any given node for search) or searching, starting from the node containing Benjamin Franklin, for Eloise (only considering all descendants of any given node for search).

In order to determine which method is more appropriate, we must examine the characteristics of the human ancestry tree, as well as the search algorithm. If we consider searching through the ancestry of Eloise, we note that any given node has exactly two immediate parent nodes. If a breadth-first strategy is used to search through this space, it will visit at least 2^{N-1} nodes (for a total search space of 2^N nodes) where N is the length of a direct path from Eloise to Benjamin Franklin in the tree. If we consider the reverse (searching from Benjamin Franklin to Eloise) we note that the size of the search space is highly dependent upon the characteristics of the tree. Any given node may have any number of descendants, any of whom may or may not fail to procreate (pruning otherwise potential branches of the tree at random). The size of the search space will depend upon the average number of children per node from Benjamin Franklin to Eloise. If we denote the maximum branching factor as B, the search space could be as large as B^N nodes, in the worst case. Thus, for a breadth-first strategy, it would be preferable to search through the ancestry of Eloise, rather than through the descendants of Benjamin Franklin.

If, however, we consider a depth-first strategy, we consider different characteristics of the ancestry tree. For example, we note that the longest path from the node containing Eloise to any descendent leaf node is likely very small – possibly as much as 4, in the very worst case. In other words, we note that Eloise is very near to the maximum depth of the tree. If we consider the operation of depth first search, we note that each successive node visited will be

at a different depth of the tree. (I.e. we will either visit all of a node's parent nodes, or all of a node's child nodes, before considering any sibling of a node). Therefore, an unbounded depth-first search would necessarily need to be conducted from Benjamin Franklin through his descendants. In practice, we can remove this limitation if we know the depth of Eloise and Benjamin Franklin within the tree. If we know the path length N between them, then we can perform a depth-limited depth-first search from Eloise to Benjamin Franklin with a bounded depth of N, for the same effect as the unbounded search from Benjamin Franklin to Eloise.

Therefore, if the maximum path length between both nodes is known, then the optimal search strategy would be to perform a depth-limited depth first search from Eloise through her ancestry, where the search depth is bounded by the maximum path length between Eloise and Benjamin Franklin, because an upwards search would only need to consider a search space of 2^N, whereas a search from Benjamin Franklin to Eloise may need to consider up to B^N nodes in the worst case. Furthermore, a depth-first algorithm need only keep a frontier size of 2 * N nodes in memory at a time, whereas a breadth-first algorithm may require $O(2^N)$ nodes in memory at a time. If the path length between Benjamin Franklin and Eloise is not known, an iterative deepening search from Eloise to Benjamin Franklin will also suffice, at the cost of revisiting most nodes of the search space multiple times.