**CS 404 HW1**

1. There are 3 kinds of state in this given problem:

So the first state is equals to prism when it is just sitting on one square. There might be 4 different successors while in this state.

This state has only 1 x and y coordinate and I repsesented these coordinates as [[x,y]] in array form. It’s successor states are shown also shown in the left

Second state is when prism has 2 x and y coordinates and stands on 2 squares vertically.

This state has 2 x and y coordinate and I repsesented these coordinates as [[x1,y1],[x2,y2]] such that y1 and y2 is equal to each other. It’s successor states are shown also shown in the left

Third state is when prism has 2 x and y coordinates and stands on 2 squares horizontally.

This state has 2 x and y coordinate and I repsesented these coordinates as [[x1,y1],[x2,y2]] such that x1 and x2 is equal to each other. It’s successor states are shown also shown in the left

Goal Test: Find the smallest sequence of moves which will lead the prism to one of the goal states.

Successor State Function: Depending one what the state is it should first check if that state’s successor is out of bounds. If it is not out of bounds then it should check whether that successor is standing on X. Finally if it is not standing on X in our matrix then we should return that state (coordinates) in order to add it to the queue.

Initial State: Our problem starts in the [[2,3],[3,3]] state.

Step cost: For step cost I considered each move as 1 cost. When calculating the heuristic cost I calculated the distance by measuring that point’s distance from the goal point.

1. **Admissibility of the heuristic function**

Given that condition for being admissible is h(n) < h\*(n) where h(n) is the value of the heuristic function and h\*(n) is the cost of the optimal path from n to a goal node, my heuristic function is admissible such that it uses a method that gives the average distance of the certain node to given goal states.

Method:

In this example h\*(n) is the optimal road for this problem (sum of the yellow arrows)

On the other hand our heuristic function is direct distance from given node to goal node

(black arrow). Which means that my heuristic function is admissible for this problem

1. Based on the maps that I came up with, here is the comparison of 2 algorithm:

(Time) UCS A\*

Example 1 0.00116 s 0.00050 s

Example 2 0.00034 s 0.00391 s

Example 3 0.00082 s 0.00671 s

(Memory) UCS A\*

Example 1 197 64

Example 2 48 28

Example 3 123 57

In terms of memory consumption, A\* is looks lesser number of nodes because A\* only checks the closest ones to the goal. On the other hand, UCS checks all possibility so it checks more number of nodes if we compare it with A\*.

In terms of time consumption, A\* spends more time since the algorithm also checks for the heuristic and calculates it for each node. On the other hand UCS algorithm does not have much to calculate since it is not dealing with the heuristic score in each node.

Finally, I believe that results were not exciting since A\* checks for heuristic score for each node that it visits.