Experimental report for the 2021 COM1005 Assignment: The Rambler's Problem*

John Craik

May 21, 2021

1 Description of my branch-and-bound implementation

To implement branch-and-bound, I created two classes that would extend existing classes that had been provided. These where as follows;

- The class 'RamblersSearch', which extended the provided class 'Search', was used to define the map and goal co-ordinate.
- The class 'RamblersState', which extended the provided class 'SearchState', was used to calculate the successor states and their corresponding costs.

The 'getSuccessors' function in the class 'RamblersState' was used to get the possible successor states from the current state. The maps used in the search are pgm maps, so each value in the pgm is connected to the values left, right, top and bottom to the value (if such values exist to the left, right, top and bottom). This gave up to 4 possible successors to each state.

The cost of moving from one state to another was dependent on the height of the current state and the height of the successor state. If the successor state had a height that was lower or equal to the current states height, the cost would simply be 1. However, if the successor state has a height that was greater than the height of the current states height, the cost was defined by this equation:

- **Cost = 1 + (successorHeight(y,x) - currentHeight(y,x)).**

*https://github.com/johncraik/com1005Assignment.git

The heights were obtained from a 2D array representing the pgm map, and then compared with eachother to calculate the cost, adding each possible successor state into a list that would be returned by this function.

2 Description of my A* implementation

Still to implement.

3 Assessing efficiency

3.1 Assessing the efficiency of my branch-and-bound search algorithm

Table 1 shows 30 randomly chosen start and goal states and their corresponding efficiency, from a 100x100 pgm map. The data below shows the mean average, highest and lowest:

- Average Efficiency: 0.03197
- Hieghst Efficiency: 0.1818
- Lowest Efficiency: 0.00843

3.2 Assessing the efficiency of my A* search algorithm

Still to implement.

3.3 Comparing the two search strategies

Still to implement.

4 Conclusions

Still to implement.

Start State (y,x)	Goal State (y,x)	Efficiency (4sf)
(90,70)	(75,38)	0.02169
(8,32)	(93,7)	0.02222
(21,68)	(29,80)	0.01599
(17,88)	(61,2)	0.02311
(86,49)	(65,41)	0.01444
(3,87)	(92,98)	0.01804
(79,80)	(24,79)	0.01455
(25,20)	(64,77)	0.01479
(69,70)	(48,46)	0.01545
(36,65)	(42,91)	0.02798
(52,39)	(4,27)	0.01227
(2,47)	(18,22)	0.02340
(14,80)	(87,76)	0.01926
(9,52)	(19,39)	0.01199
(52,1)	(20,34)	0.01360
(60,0)	(97,84)	0.01492
(56,60)	(1,50)	0.00843
(54,88)	(87,83)	0.01060
(56,4)	(61,33)	0.07559
(82,55)	(86,57)	0.09465
(91,96)	(50,5)	0.02513
(83,34)	(94,88)	0.02283
(41,92)	(36,40)	0.05949
(62,76)	(24,59)	0.00923
(54,75)	(19,0)	0.01596
(34,88)	(70,75)	0.01059
(27,98)	(68,15)	0.01801
(76,7)	(79,35)	0.1096
(44,60)	(43,54)	0.1818
(76,87)	(87,5)	0.03351

Table 1: Effieciencies of 30 random start and goal co-ordinates on a 100x100 pgm map.