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.

Once this had been implemented, I created another class 'RunRamblersBB'. This class was used to test my code that I had implemented and produce the statistics used later on in the document for branch-and-bound.

2 Description of my A* implementation

Implementing the A* algorithm was fairly simple as most of the work had already been done when implementing branch and bound (however, the code provided in 'search4' was used, not 'search3'). The only difference is the calculation of the cost, as A* uses both ramblers cost and an estimated remaining cost, which is based on the distance the current state is from the goal state.

Calculating the estimated remaining cost was achieved by calculating the difference between the two states, using the equation below:

- *Remaining Cost = Abs(currentY - successorY) + Abs(currentX - successorX) + Abs(currentHeight - successorHeight)*

Once this had been implemented, I created another class 'RunRamblersAstar'. This class was used to test my code that I had implemented and produce the statistics used later on in the document for A*.

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: 3.197%
- Hieghst Efficiency: 18.18%
- Lowest Efficiency: 0.843%
- Range: 17.337%

3.2 Assessing the efficiency of my A* search algorithm

Table 2 shows the efficencies of the same 30 states when using the A* algortithm The data below shows the mean average, highest and lowest:

- Average Efficiency: 7.348%
- Hieghst Efficiency: 43.75%
- Lowest Efficiency: 1.848%
- Range: 41.902%

3.3 Comparing the two search strategies

Based on the 30 searches, A* algorithm clearly has the better efficiency compared to branch-and-bound. Branch-and-bound averages around 3.2%, whereas A* averages around 7.3% - that is an increase of 4.1%.

However, branch-and-bound has a smaller range of values for efficiency (only 17.3%) compared with A*'s range of 41.9%. This implies that branch-and-bound produces more consistant search results than A*.

4 Conclusions

To conclude, A* is more efficient than branch-and-bound; however, in the pgm map used brand-and-bound produced more consistant results. This could be down to the start and goal states set at the begining by a random number generator, or the fact that the map was only 100px by 100px. This was to be expected because A* uses an extra heuristic (estimated remaining cost) which, in most cases, improves the efficiency of the search significantly (by 4.1% on average).

If I had more time to do more tests, I would use a much larger map and produce more start and goal states. Increasing the map size by at least 500% and calculating the efficiency for at least 100 different start and goal states, may provide a more accurate picture of which search is more efficient and more consistant.

Start State (y,x)	Goal State (y,x)	Efficiency (%, 4sf)
(90,70)	(75,38)	2.169
(8,32)	(93,7)	2.222
(21,68)	(29,80)	1.599
(17,88)	(61,2)	2.311
(86,49)	(65,41)	1.444
(3,87)	(92,98)	1.804
(79,80)	(24,79)	1.455
(25,20)	(64,77)	1.479
(69,70)	(48,46)	1.545
(36,65)	(42,91)	2.798
(52,39)	(4,27)	1.227
(2,47)	(18,22)	2.340
(14,80)	(87,76)	1.926
(9,52)	(19,39)	1.199
(52,1)	(20,34)	1.360
(60,0)	(97,84)	1.492
(56,60)	(1,50)	0.843
(54,88)	(87,83)	1.060
(56,4)	(61,33)	7.559
(82,55)	(86,57)	9.465
(91,96)	(50,5)	2.513
(83,34)	(94,88)	2.283
(41,92)	(36,40)	5.949
(62,76)	(24,59)	0.923
(54,75)	(19,0)	15.96
(34,88)	(70,75)	1.059
(27,98)	(68,15)	1.801
(76,7)	(79,35)	10.96
(44,60)	(43,54)	18.18
(76,87)	(87,5)	3.351

Table 1: The effieciencies of 30 random start and goal states on a 100x100 pgm map, using branch-and-bound.

Start State (y,x)	Goal State (y,x)	Efficiency (%, 4sf)
(90,70)	(75,38)	6.409
(8,32)	(93,7)	2.834
(21,68)	(29,80)	16.8
(17,88)	(61,2)	2.039
(86,49)	(65,41)	7.633
(3,87)	(92,98)	2.260
(79,80)	(24,79)	16.91
(25,20)	(64,77)	2.066
(69,70)	(48,46)	10.55
(36,65)	(42,91)	9.621
(52,39)	(4,27)	4.111
(2,47)	(18,22)	3.097
(14,80)	(87,76)	1.848
(9,52)	(19,39)	1.848
(52,1)	(20,34)	4.885
(60,0)	(97,84)	2.029
(56,60)	(1,50)	15.14
(54,88)	(87,83)	3.380
(56,4)	(61,33)	3.380
(82,55)	(86,57)	43.75
(91,96)	(50,5)	1.910
(83,34)	(94,88)	3.411
(41,92)	(36,40)	3.411
(62,76)	(24,59)	12.90
(54,75)	(19,0)	2.027
(34,88)	(70,75)	2.027
(27,98)	(68,15)	1.918
(76,7)	(79,35)	46.04
(44,60)	(43,54)	20.51
(76,87)	(87,5)	7.127

Table 2: The effieciencies of the same 30 start and goal states used in branch-and-bound, however searching using the A* algorithm.