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

George Smith

May 21, 2021

1 Description of my branch-and-bound implementation

I implemented the branch and bound search using the two classes RamblersState and RamblersSearch which extend SearchState and Search respectively. RamblerSearch contains get methods for TerrainMap it's using and coordinates of the goal point. RamblersState has three main override methods. First, goalPredicate returns a boolean value for whether or not the current node coodinate is the goal point. Secondly, getSuccessors returns an ArrayList of searchStates which are the available nodes adjacent to the current one. Finally, sameState returns a boolean on whether a given SearchState is equal to the current one.

2 Description of my A* implementation

My A* implementation builds upon the RamblersState class from the branch and bound. The key difference is the use of an estimated remaining cost calculated in 3 different ways with different heuristics. The euclid and manhatten methods each use their own equations to return this value while the final estimate uses the height between the current and goal point. How each of these strategies affect efficiency will be explored below.

*https://git.shefcompsci.org.uk/aca19gs/com1005assignment

3 Assessing efficiency

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

Every test represents 50 searches each with a random start and goal point. These tests are using the tmc.pgm file.

Test	Average efficiency
1	0.15100991975516082
2	0.16733702167868614
3	0.1626079661399126

Table 1: Branch and Bound test results

Each test represents 1 search each with a random start and goal point. Using the diablo.pgm file. Due to the increased terrain map size and therefore increased runtime I conducted less diablo tests.

Test	Average efficiency
1	0.01116487244144082
2	0.018535560462623835
3	0.017769947182387114

Table 2: Branch and Bound test results

The data shows that the efficiency of tends to get worse as the terrain map increases in size.

3.2 Assessing the efficiency of my A* search algorithm

Every test represents 50 searches each with a random start and goal point. These tests are using the tmc.pgm file.

Test	Average efficiency
1	0.23535407803952693
2	0.22775641083717346
3	0.24974265683442354

Table 3: Euclid estimate results

Test	Average efficiency
1	0.29250853292644025
2	0.26403864987194536
3	0.2426462508738041

Table 4: Manhatten estimate results

Test	Average efficiency
1	0.24906378224492073
2	0.3079067297279835
3	0.27080396443605426

Table 5: Direct line results

The data shows that the efficiency is worst using the euclid method with the smaller tcm map.

Each test represents 1 search each with a random start and goal point Using the diablo.pgm file. Due to the increased terrain map size I conducted less diablo tests

Test	Average efficiency
1	0.01055575255304575

Table 6: Euclid estimate results

Test	Average efficiency
1	0.07496175169944763

Table 7: Manhatten estimate results

Test	Average efficiency
1	0.007165135350078344

Table 8: Direct line results

The data shows that the manhatten estimate is the most efficient using the larger diablo terrain map while the direct line method is now by far the least efficient.

3.3 Comparing the two search strategies

For the most part, the A* searches remain more efficient than the branch and bound. With the small tmc terrain map, all heuristics for the A* search are more efficient. However, when searching through the larger diablo terrain map only the manhatten heuristic is more efficient while the euclid method has similar efficiency to the branch and bound and the direct line method is much more inefficient than any of them.

4 Conclusions

In conclusion, the A* search method is the most efficient in most scenarios as long as the manhatten method is used for larger terrain maps as it scales up best, however, neither of the strategies are particularly time effective when tackling the larger diablo terrain map.