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

Benjamin Consterdine

May 21, 2021

1 Description of my branch-and-bound implementation

I implemented the branch-and-bound search by writing 3 classes to work alongside the provided ones. These new classes were:

- RamblersSearch
- RamblersState
- RunRamblersBB

RamblersSearch is a fairly basic class which extends the provided Search class. It's job is to set up the search environment with a map and goal coordinates.

RamblersState makes up the majority of the implementation. It defines a given state on the map, including the coordinates, and the cost of getting to that location (localCost). It has a method for finding the possible successors to a given location (getSuccessors), which calls on a helper method (costToSuccessor) to find the localCost for that move. As well as this, it has a method to check if the current state is the goal (goalPredicate) and a method for checking if two states are the same (sameState).

Finally, RunRamblersBB is a small class that consists only of a main method, which defines the terrain map file to use, the starting coordinates and the goal coordinates. It then makes a RamblersSearch object and RamblersState (initState) using that information. It then runs the search using the runSearch method provided and prints the details of the search

*https://github.com/BennCon/1005assignment.git

2 Description of my A* implementation

As with the branch-and-bound search, there were three classes that needed to be implemented. RamblersSearch was identical to the branch-and-bound implementation, and RunRamblersBB was replaced with RunRamblersAstart, which was mostly the same but now specified A* search as the search strategy. RamblersState had some additions, as for A* there is an estimated remaining cost factored in. To add this to the class, I needed a helper method (estRemCost) that could calculate an estimate from a state to the goal, based on different heuristics.

The first heuristic I chose was the 'Manhattan distance' - the difference in x + the difference in y ($dx + dy$). In this set-up which doesn't allow for diagonal movement, this effectively provides the lowest possible distance to the goal. As distance and cost are closely related, this is a logical heuristic. It keeps the search admissible as it is guaranteed not to be an overestimate of the cost.

Next I implemented the euclidean distance as an estimate of the remaining cost. This is similar logic to the Manhattan distance, but is simply the shortest straight line distance between the points. I will discuss this heuristic's effectiveness for our setup in a later section.

Finally, I added two heuristics which were adaptations of these accounting for height differences. One estimated the cost to be the difference in height + the Manhattan distance, $(dz) + (dx + dy)$, as each move costs height difference + 1. The other heuristic was effectively the 3D Euclidean distance, $\sqrt{(dx)^2 + (dy)^2 + (dz)^2}$.

For both of these heuristics, I decided that if the height difference was negative (i.e. the goal was lower than the current state) the estimated cost should just be the Manhattan Distance and Euclidean Distance, respectively. This is because it would not make sense to give a negative estimated cost; every move costs at least 1.

3 Assessing efficiency

In order to assess the efficiency of the different algorithms and heuristics, I ran searches between random coordinates on two different terrain maps; one fairly small (16x16), and another much larger one (252x249). For the smaller map, I was able to run 10,000 searches for branch-and-bound, and a further 10,000 for each A* heuristic. Due to run-time constraints, I was limited to running 50 searches for each on the larger map, but this is still

enough to get a fairly reliable average reading.

Included alongside this report are two CSV files containing all the details of each test ran.

For the purpose of this report, efficiency refers to the ratio of nodes explored to the number of nodes on the solution path. This means that a search with an efficiency of 1 only explored nodes that ended up in the solution.

Figure 1 shows a visualisation of the larger of the two maps. Brighter areas represent higher terrain. This was made in the free software IrfanView.

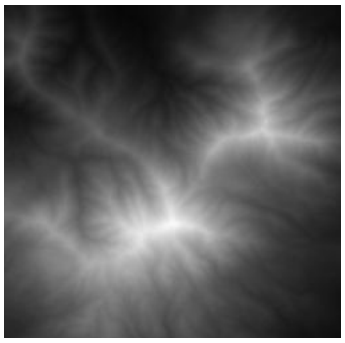

Figure 1: Visualisation of the larger 'Diablo' map.

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

On the smaller map, I took an average efficiency of 0.15541 (rounded for brevity) over the 10,000 readings. These tests ran very quickly and efficiency is fairly high - making it clear that for smaller set-ups like 16x16 map, branch-and-bound is an effective enough algorithm.

Map Size	Avg. Efficiency
Small	0.15541
Large	0.00669

Table 1: Performance of branch-and-bound for different map sizes

Of course, for increasing distances the efficiency does drop, as shown in Figure 2 below.

On The larger map, branch-and-bound was far less effective, with an average efficiency of 0.00669. For these more complex (and more realistic)

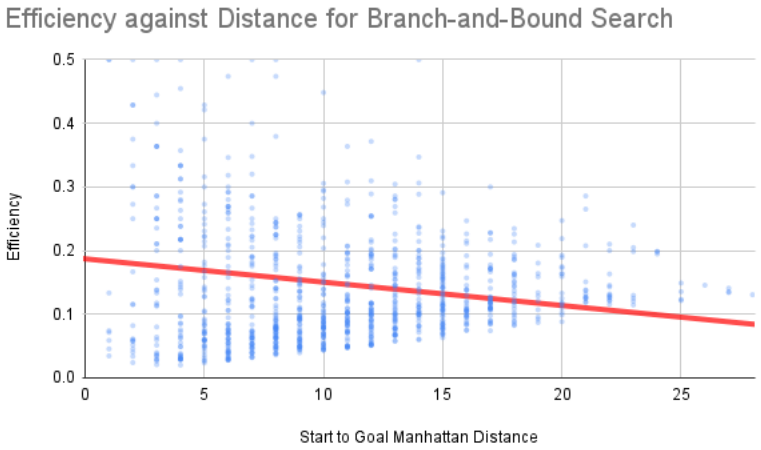

Figure 2: Graph showing how efficiency decreases as the distance between the start and goal increases for a branch-and-bound search.

applications, it becomes clear that a different algorithm may be necessary.

3.2 Assessing the efficiency of my A* search algorithm

I ran the same tests for each of the different heuristics and took average efficiencies. All of the heuristics performed well, but as seen in Table 2 below, heuristics based on Euclidean distance were significantly less efficient. This is to be expected, as diagonal movement is not possible in our set-up, so straight line shortest distance isn't an especially relevant measurement.

Heuristic	Avg. Efficiency
Manhattan Distance	0.02627
Euclidean Distance	0.01712
Height Difference	0.04014
3D Euclidean Distance	0.01857

Table 2: Performance of different A* heuristics measured on the large map

On the smaller map, performance was very strong too, with the height heuristic getting an average efficiency of 0.37744, and the weakest heuristic, the Manhattan Distance, having an average of 0.21590. I was surprised to see that here the Euclidean heuristics performed far better than they did on the large map. This is presumably due to the difference between Manhattan

Distance and Euclidean Distance being far smaller for the shorter distance searches.

3.3 Comparing the two search strategies

It is clear that the A* search performs significantly better than branch-and-bound in general. Figure 3 shows a comparison of the searches for different map sizes. For both maps it is clear that A* search with a good heuristic can make a huge difference to efficiency over an uninformed strategy like branch-and-bound. However, it is on the larger map where the difference is far more prominent, with the A* height search performing 6x better than branch-and-bound. With the difference being far less noticeable on the smaller map, it is clear that sometimes writing a more advanced search wouldn't be worth it - especially if it was a situation where heuristics are hard to form.

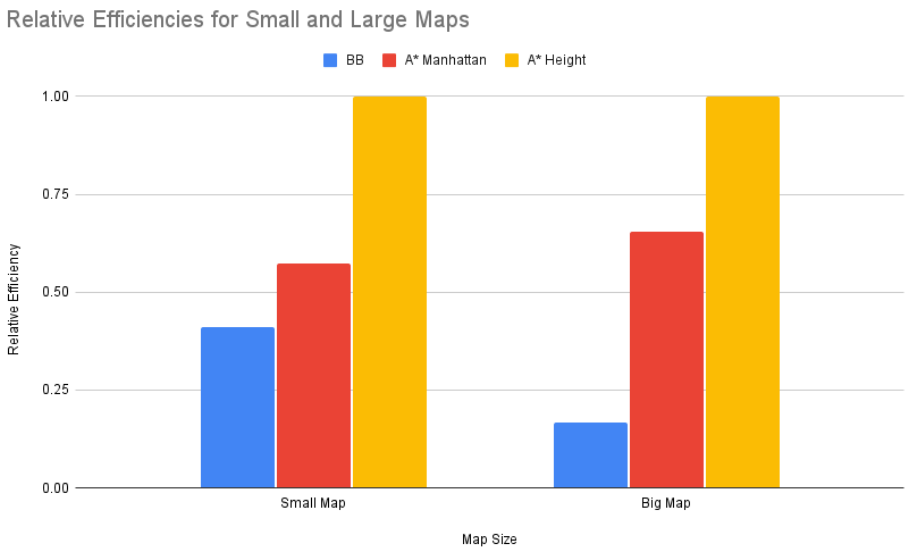

Figure 3: Relative efficiencies (best efficiency for each search was set to 1) for different searches on different map sizes.

Whilst comparing results for the different algorithms, I decided to see how they perform for increasing distances. Figure 4 shows this relationship, which seem quite surprising. It is clear that the A* searches, especially with the height heuristic, perform far better than branch-and-bound for most distances. However, at the longest distances tested for this report

the efficiencies begin to converge, with the trend lines showing branch-and-bound performing marginally better than the A* heuristics. Although I'm doubtful that branch-and-bound would actually perform better (I think lack of data for large distances makes my graph somewhat unreliable), it is clear that the benefits of A* may become limited for more complex problems. Further testing for more scenarios/problem sizes would be required to assess this more fully.

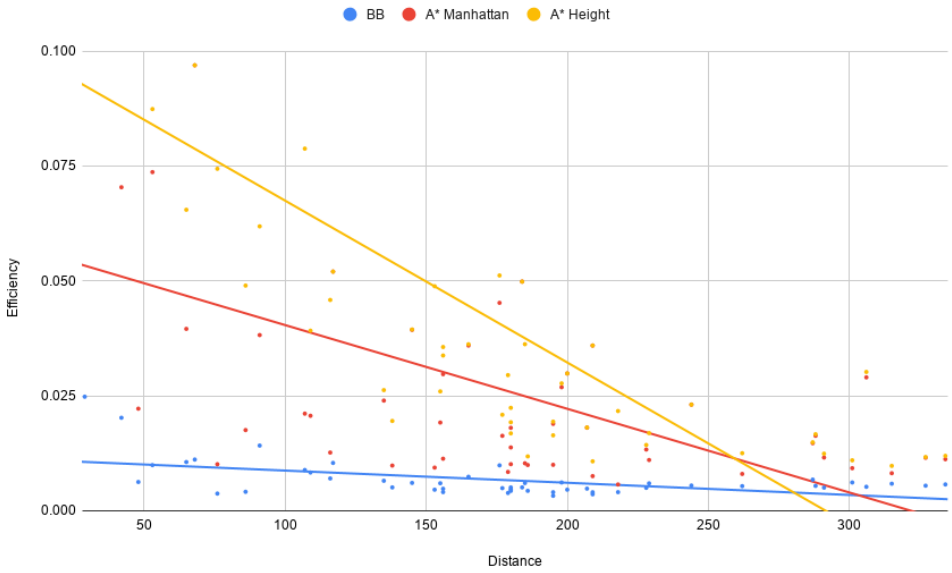

Figure 4: Efficiencies of the algorithms as distance increases on the larger map. Distance here refers to the Manhattan Distance between the start and goal.

4 Conclusions

It is clear that in general, A* searches perform significantly better than uninformed strategies such as branch-and-bound. Of course, this is entirely dependent on the heuristic used for the A* search - the reasonably poor performance of the Euclidean distance exemplifies the importance of a logical heuristic. The strong performance of A* does not make uninformed strategies obsolete however; in practise the branch-and-bound was a perfectly adequate strategy for the small map. The decreasing performance of

A* with more complex searches (i.e. longer distances) makes it apparent that a yet more advanced strategy may be needed for some problems.

To conclude, I think the evidence I have gathered shows that neither of these algorithms are perfect, and a decision should be made about which is best for the individual problem. It has to be considered whether or not an advanced informed search strategy is necessary (i.e. for small search spaces), and moreover whether or not it would be worthwhile to come up with a valid heuristic for that situation. A further consideration is that efficiency is not the be all and end all; you may have an A* algorithm with excellent efficiency but terrible run time, perhaps due to the estimated remaining cost being computationally expensive to calculate.

In more complex problems, an A* search with a good heuristic is clearly an excellent solution with high efficiency, and should absolutely be used.