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

Jedrzej Golebiewski

May 16, 2021

1 Description of my branch-and-bound implementation

The classes *RamblersState* and *RamblersSearch* extend the more general search classes given in the starting code. *RamblersState* has two domain specific variables (coordinates and height) as well as all the domain specific methods necessary for the search to be executed properly.

Methods in *RamblersState*:

- *goalPredicate* returns *true* iff the state coordinates match the goal coordinates.
- *getSuccessors* returns all the possible successor states that are allowed. Local cost of successors is calculated according to the formula given in the problem specification.
- *sameState* returns *true* iff the states being compared have the same coordinates.

RamblersSearch has three domain specific variables (goal coordinates, map and estimation method for A* heuristic calculation).

2 Description of my A* implementation

The A* and branch-and-bound implementations use the same *RamblersState* and *RamblersSearch* classes, however there is a specific method in *RamblersState* for calculating the estimated remaining cost called *calculateEstimatedRemainingCost*.

There are four options for global remaining cost evaluation to choose from:

- Euclidean distance
  $cost = \sqrt{(x_0 - x)^2 + (y_0 - y)^2}$
- Manhattan distance
  $cost = |x_0 - x| + |y_0 - y|$
- Height difference
  $cost = h_0 - h$ *(can be negative)*

[*]https://github.com/Jed-g/com1005-assignment

- "Super advanced"
  $cost = |x_0 - x| + |y_0 - y| + ELU(h_0 - h)$ *(can be negative)*

All values will always be (under)estimates.

3 Assessing efficiency

6 efficiency tests were run in total (3 on each map) and for each test different initial and goal coordinates were chosen. For each of the tests searches were executed using the branch-and-bound and A* search techniques. A* searches were carried out 4 times for each test, each time with a different approximation method.

Results:

Test details	Branch-and-bound	A* (Euclidean distance)	A* (Height difference)	A* (Manhattan distance)	A* ("Super advanced")
tmc.pgm, (0,0) => (10,10)	13.24%	13.85%	15%	13.92%	25.71%
tmc.pgm, (0,15) => (15,0)	13.96%	15.20%	13.90%	15.5%	23.13%
tmc.pgm, (7,3) => (7,14)	16.67%	21.92%	7.27%	24.24%	24.24%
diablo.pgm, (0,0) => (245,245)	0.78%	0.98%	0.78%	1.37%	1.35%
diablo.pgm, (100,200) => (200,50)	0.43%	0.54%	0.45%	0.7%	1.1%
diablo.pgm, (175,125) => (90,200)	0.31%	0.58%	0.51%	0.76%	1.78%

Table 1: Efficiency of searches conducted with branch-and-bound and A*

4 Conclusions

All algorithms had a relatively good efficiency on the smaller map and experienced a significant drop on the bigger map due to the larger amount of paths possible between the start and goal nodes (combinatorial explosion).

Branch-and-bound on average performed the worst out of all of the search techniques due to basing the selection of the next node to expand solely on its current global cost.

The A* searches that utilized the *Euclidean distance*, *Height difference* and *Manhattan distance* heuristic for the estimated global remaining cost performed quite similarly on all of the tests except one.

The A* searches that used the *"Super advanced"* heuristic performed on average the best. This heuristic is equal to the sum of the height difference mapped to the Exponential Linear Unit (ELU) function and the Manhattan distance. The superior performance of this heuristic can be attributed to the fact that it uses two different variables (height difference + distance) for the calculation instead of just one (height difference or distance). If one variable isn't the best indicator of the rough remaining cost in a given situation, the other one will to some degree "compensate" for it.
Some ideas for the improvement of this heuristic:

- Using the Rectified Linear Unit (ReLU) for the mapping of the height difference instead of the Exponential Linear Unit (ELU). Going downhill should not be encouraged, especially earlier on, because there are no benefits in doing so. The cost for going down is not lower but exactly the same as going across flat terrain. After being encouraged by the heuristic to go downhill, the path will, more often than not, have to go uphill again decreasing the probability of it being the lowest cost solution.

- Normalizing/scaling the height difference and distance values in some way so that one doesn't "overpower" the other.