Concordia University

October 15th 2018

COMP 472 Mini-Project 1

Jacob Gagné - Benjamin Barault

40003704 - 40003661

We certify that all the work submitted for this course will comply with these requirements and with additional requirements stated in the course outline.

Heuristics

Heuristic #1, euclidean distance:

The euclidean distance is the shortest path from point A to point B. It can be calculated given point A as $(a_1, a_2, a_3, ..., a_n)$, where a_1 through a_n are where the point lies in a given dimension, 1 being the first dimension, 2 being the second dimension, n being the nth dimension. We are then given a point B which has the same properties as point A but can hold different values. We can then use the following formula to calculate the euclidean distance, where we denote the euclidean distance as the following function $E(P_A, P_B)$, where P_A is point A, and P_B is point B:

$$E(P_A, P_B) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + ... + (b_n - a_n)^2}$$

Given the case of the n-puzzle we are dealing with the second dimension so the above can be simplified to the following formula:

$$E(P_A, P_B) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}$$

Where point A is the current tile which we are evaluating and point B is where the current tiles value should lie in the puzzle. In particular the points first dimension would relate to the row in which it currently lies and where it should lie, and the second dimension would relate to the column in which it currently lies and where it should lie. We simply apply this formula for every non-zero tile in our n-puzzle state.

Given the fact that the euclidean distance is the shortest path from one point to another, it easily translates to the same thing given the n-puzzle problem. That is the n-puzzle problem can be viewed as a cartesian graph, and the same properties of the euclidean distance hold. It can easily be seen that this heuristic is admissible because the moves available to the heuristic is either moving 1 in either the north, west, south, or east direction or by traversing on a diagonal which is a distance of $\sqrt{2}$ (for a single tile move), while the euclidean distance is not limited to these moves, it can take even smaller paths, while the heuristic would be forced to make discrete moves (leading to a longer path).

Heuristic #2, modified manhattan distance:

The modified manhattan distance implemented is the distance between where a tile is and where it should be considering the move possibilities. This includes diagonal movements. If a tile is two rows above and one column in front of where it should be the cost will be two. The following formula explains how the total cost is derived.

$$\delta(i, j) = | i - j |$$

Delta represents the absolute difference between two values. In the formula below this is used to compare distance in one dimension.

$$E(P_A, P_B) = \left|\delta(x_A, x_B) - \delta(y_A, y_B)\right| + \lambda$$

$$where\ \lambda = Min\ (\ \delta(x_A, x_B),\ \delta(y_A, y_B)\ )$$

Where point A is the current tile which we are evaluating and point B is where the current tiles value should lie in the puzzle. x_A, x_B represent the coordinates of P_A and P_B in the first dimension and y_A, y_B represent the second dimension. The difference in rows and columns is subtracted and added to the minimum difference providing the minimum possible moves to make for tile at P_A to get to P_B. This heuristic is admissible since it assures that the amount of estimated moves is always smaller or equal than the actual amount of moves.

Difficulties

One difficulty which we ran into was that while programming the GameStatePriorityQueue object, it was difficult to debug our implementation because to properly print out the order of the queue we would have to use the pop function, but since we are working on a reference to an object this actually affected the object which we were using to perform the state space search. To overcome this problem we first wrote unit tests for the GameStatePriorityQueue which helped us realize that the queue was working just fine, and later on we created a print function which creates a copy of the priority queue and then works on that copy instead of the original.

Experiments

EBF = Effective Branching Factor
(we use an approximate: $EBF = N^{\frac{1}{d}}$ $where\ N\ =\ NNV,\ and\ d\ =\ depth\ of\ the\ solution$
NNV = Number of Nodes Visited

The following results are our heuristics #1 and #2 being run on 5 different boards of 3x4 size, using best first search and A* search. An analysis on the results shall follow.

Heuristic #1 experiments:

Input	EBF	NNV	Final Cost	Time Taken
[9, 1, 7, 5, 10, 8, 6, 3, 0, 2, 11, 4] BFS	1.14899	37	26	684.214μs
[9, 1, 7, 5, 10, 8, 6, 3, 0, 2, 11, 4] A*	1.22229	276	28	9.204634ms
[11, 6, 4, 1, 3, 9, 8, 10, 2, 5, 0, 7] BFS	1.10450	194	53	2.015502ms
[11, 6, 4, 1, 3, 9, 8, 10, 2, 5, 0, 7] A*	1.16695	88	29	1.179576ms
[7, 4, 1, 9, 0, 5, 3, 11, 8, 6, 10, 2] BFS	1.12327	187	45	2.857958ms
[7, 4, 1, 9, 0, 5, 3, 11, 8, 6, 10, 2] A*	1.21464	612	33	15.669033ms
[8, 2, 3, 1, 5, 11, 9, 7, 6, 10, 0, 4] BFS	1.15539	323	40	3.47854ms
[8, 2, 3, 1, 5, 11, 9, 7, 6, 10, 0, 4] A*	1.35821	2865	26	36.928323ms
[8, 4, 2, 5, 0, 11, 9, 7, 1, 3, 6, 10] BFS	1.12185	395	52	2.594647ms
[8, 4, 2, 5, 0, 11, 9, 7, 1, 3, 6, 10] A*	1.17269	225	34	1.954013ms

Heuristic #2 experiments:

Input	EBF	NNV	Final Cost	Time Taken
[9, 1, 7, 5, 10, 8, 6, 3, 0, 2, 11, 4] BFS	1.16903	58	26	1.021384ms
[9, 1, 7, 5, 10, 8, 6, 3, 0, 2, 11, 4] A*	1.22507	87	22	1.57976ms

[11, 6, 4, 1, 3, 9, 8, 10, 2, 5, 0, 7] BFS	1.18643	51	23	888.207µs
[11, 6, 4, 1, 3, 9, 8, 10, 2, 5, 0, 7] A*	1.26444	138	21	2.233015ms
[7, 4, 1, 9, 0, 5, 3, 11, 8, 6, 10, 2] BFS	1.14397	325	43	11.110853ms
[7, 4, 1, 9, 0, 5, 3, 11, 8, 6, 10, 2] A*	1.25896	1260	31	29.109885ms
[8, 2, 3, 1, 5, 11, 9, 7, 6, 10, 0, 4] BFS	1.09405	774	74	14.114146ms
[8, 2, 3, 1, 5, 11, 9, 7, 6, 10, 0, 4] A*	1.31829	759	24	14.309288ms
[8, 4, 2, 5, 0, 11, 9, 7, 1, 3, 6, 10] BFS	1.13919	84	34	1.319513ms
[8, 4, 2, 5, 0, 11, 9, 7, 1, 3, 6, 10] A*	1.15984	115	32	3.440117ms

Analysis

The first thing that is noticeable when looking at the results is the fact that A* search tends to find a shorter path than best-first search does comparing A* to best-first for both individual heuristics. This makes sense because best-first search does not take into consideration the cost of the path that it is currently on and so it will more easily go down paths which may not be optimal. This also explains the fact that the number of visited nodes of A* tends to be greater than the number of visited nodes of best-first search.

The second thing that is noticeable is that our second heuristic tends to find shorter paths on the same input for both best-first search and A* search. This can be explained by the fact that our second heuristic is more informed than our first, that is that $h_1 \leq h_2 \leq h^*$ and this is easily proven because both h_1 and h_2 are both admissible as shown previously in this document, and h_2 always takes a path longer than h_1 , this makes sense because they are both traversing a graph but h_1 is the euclidean distance (shortest distance) so it must always be less than h_2 . The fact that our second heuristic is more informed than our first can also explain the fact that it needs to search less nodes on average to find the solution path, this can be thought as our second heuristic making better guesses so it converges more quickly.

Finally we can make an observation about the EBF values. We can make the observation that it is not an important factor in the case of our experiments, and that is because the final depth of one heuristic might be different from the final depth of another. This is the case because we are doing a Graph Search, and therefore to have the benefit of not re-considering revisited states we need our heuristics to be both admissible and consistent. In which case our first heuristic is probably not monotonic because our second heuristic was capable of finding shorter paths to the goal state. Had it been the case that both of our heuristics were admissible and monotonic, or if we were to use a Tree Search algorithm instead (reconsidering already visited states if they have a cheaper cost) then we would probably be able to make use of the EBF column.

References

[1] http://ai.stanford.edu/~latombe/cs121/2011/slides/D-heuristic-search.pdf

[2] http://ozark.hendrix.edu/~ferrer/courses/335/f11/lectures/effective-branching.html

[3] https://golang.org/pkg/container/heap/