California Polytechnic State University, Pomona
CS420 Section 1
Artificial Intelligence

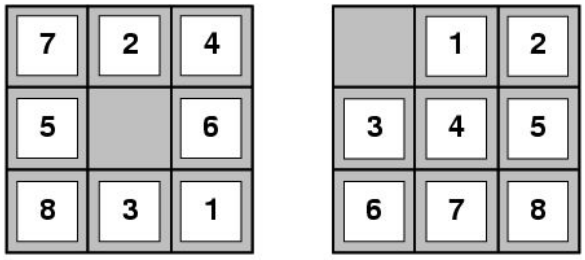
Jacob Lepere
ID: 012839151

Project 1: 8-Puzzle
02.09.2018

Introduction

A* is a well-documented informed search algorithm to find the path from an initial state to a goal state. The challenge within A* is the ability to find the 'best' heuristic function, where the 'best' heuristic solves the problem faster that other heuristics. Although these functions are problem dependent, there are characteristics of heuristic functions that should be adhered to. The first is admissibility, which states that the heuristic function should never overestimate the true cost from the current state to the goal state. The second is consistency, which follows the triangular inequality concept where $h(n) \leq c(n,n') + h(n')$, where $h(n)$ is the heuristic function applied to node n, $c(n,n')$ is the cost from n to n', and n' is a successor of n, for all n and successors n'.

The classical problem at hand is that of the 8-Puzzle, which consists of a 3x3 board of tiles containing a permutation of 0 to 8, where 0 represents the blank tile. This blank tile can move up, down, left or right, if possible, to swap places with an adjacent tile. The goal state of the puzzle is the configuration 0 1 2 3 4 5 6 7 8, reading from right to left, top to bottom. Given any initial state or permutation, the objective is to find a sequence of actions to be applied to the initial state to get to the goal state in a minimum number of steps.



Approach

The A* algorithm, implemented in Java, was used to solve the 8-Puzzle problem with two heuristic function. The first heuristic function counts the number of misplaced tiles while the second sums the distances of the tiles from their goal positions, also known as the Manhattan distance. Both functions adhere to both admissibility and consistency, although the proof will be omitted. These constraints, however, imply that A* with either heuristic function will find an optimal solution. Therefore, efficiency analysis can be performed to compare how quickly both algorithms find the optimal solution given the same input. To do this, 100 sample inputs for depths $2 \leq i \leq 20$, for even i, will be applied to both algorithms and the average number of nodes generated will be calculated for each depth. Although not definitive, admissible heuristic functions with a greater output tend to find a solution faster than admissible heuristic functions with a smaller output. Therefore, I expect the Manhattan distance heuristic to find a solution faster than the sum of misplaced tiles heuristic.

Comparison

Following as a chart depicting the average number of generated nodes per depth for both heuristic functions.

h1	Number of Misplaced Tiles
h2	Manhattan Distance

		Nodes Generated		Time (ms)	
Depth	Trials	Average of A*(h1)	Average of A*(h2)	Average of T(h1)	Average of T(h2)
2	100	5	5	0.079	0.086
4	100	9	9	0.046	0.081
6	100	15	13	0.078	0.049
8	100	30	20	0.086	0.048
10	100	66	32	0.154	0.101
12	100	156	56	0.427	0.096
14	100	371	108	1.326	0.186
16	100	860	206	7.091	0.575
18	100	1991	369	29.258	1.283
20	100	4861	599	183.300	2.968

As evident in the chart, the number of nodes generated using the Manhattan distance heuristic is less than or equal to the number of nodes generated using the number of misplaced tiles heuristic, for each depth. It is also worth noting, although obvious, that the time taken to execute A* with the first heuristic grows much faster than with the second heuristic. This is in alignment with the prior assumption regarding the correlation between heuristic output and efficiency. It is important to note that the method to calculate the number of nodes generated was to sum the size of the frontier and the size of the explored sets, after the goal was found. This would accurately calculate the total number of nodes generated if no replacement was done. This was not the approach taken. If a node was generated that was not in the explored set but was in the frontier set with a smaller evaluation function ($f(n) = g(n) + h(n)$, where $f(n)$ is the evaluation function, $g(n)$ is the path cost and $h(n)$ is the heuristic function), then all nodes with a greater than or equal to evaluation function result with the same state would be removed from the frontier prior to the insertion of the newly generated node. Although this has no effect on the solution, it results in less nodes in the frontier as well as less repeated exploration of nodes (efficiency).

Conclusion

The results of the experiment are aligned with prior assumptions as well as expected documented results. Therefore, the benefits gained from the experiment weren't the results found but rather the experience from implementing the informed search algorithm. Implementation was designed with multiple interfaces for reusability to be applied to additional problems, creating a foundation for uninformed and informed search algorithms. Also, it was beneficial to see efficiency first hand for different heuristic functions, as well as implementation approaches. For example, it was found that removing redundant nodes from the frontier with the same state but greater evaluation function results drastically improved efficiency for greater depth solutions.

In the future, it would be appealing to implement a problem with unknown solution (at least to the programmer). This would result in a higher satisfaction level after implementation is complete. For the 8-Puzzle problem, it would be interesting to analyze number of nodes generated and the runtime with a graph for the different approaches and formulate an equation for estimates with higher depth solution paths. As we can see from the chart, it may become unrealistic to test 100 puzzles with depths of 28 or so with the number of misplaced tiles heuristic. This could also be interesting to do with more heuristics, such as half of the Manhattan distance.