Alejandro Sherman
SID: 862062898
No partner

CS170 Eight Tile Project Report

Due Tuesday, April 27, 2021

Introduction

This eight tile solver project aims to depict the differences in performance of blind and informed search, when applied to the eight tile problem. This project also aims to expose the variance in effectiveness of differing implementations of an informed A* search, the key difference being, the information (heuristic) applied to it.

Design

My design follows the graph search algorithm given in the project instruction document. Additionally, my code utilizes a problem class with the goal state and operators built in. I also used a node class to allow for the creation of expanded nodes (children). I did not build any tree class however, and children nodes lack a pointer to their parents, so a backwards trace is not possible under my implementation.

I designed my graph search algorithm to be general such that the actual search heuristic algorithm being applied can be done elsewhere, and be chosen by the user in the main. I also attempted to have the tile game be expandable, by most references to the number of rows being general, rather than hard coded to 3. While this is the case, my design would need to be changed in many places to scale to a larger tile game - goal state, comparisons with goal state for A* algorithms, puzzle creator in the main etc.

The frontier I used (set_list) was a simple array that I treated as a set and made sure to only insert new states. (I went with a set design as Rutuja had suggested)

Hurdles

My implementation is in python.
As most students at UCR, I have been using the c++ programming language longer than any other. Even so, recently I have been doing more programming in python, and have appreciated a lot of the extra flexibility given by the language (no semicolons and dynamic var typing for instance). This being the case, there were still many growing pains during the implementation process as I developed. Pages that explained the python equivalents to c++ methods, were especially helpful, and a few are linked below with my sources.

Once I was in the swing of things, however, I feel that python proved to be a good choice for the implementation I was going for. Discovery of deepcopy from the copy library was especially helpful for the saving of children nodes.

Three search algorithms used

As mentioned before, this project aims to show the searching ability of informed A* searches. In particular, the formula follows f(n) = g(n) + h(n) to find the cheapest node to take. Here g(n) just equals the path or depth, as under this design every edge has a uniform weight of 1. What varies is the value of h(n).

For uniform cost search (blind) h(n) is hardcoded to 0.
For misplaced tile heuristic A* search, h(n) is equal to the number of misplaced tiles from the current puzzle state, against the goal state.
For euclidean distance heuristic A* search, h(n) is equal to the sum of the exact distances of misplaced tiles from the current puzzle state, against the goal state.
(formula: $d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$ - from wikipedia, link below)

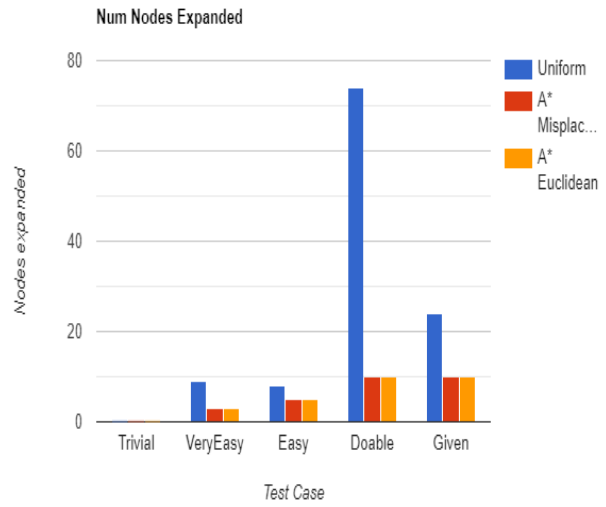
Performance on test cases

Below are various puzzle test cases, and their performance given each search algorithm in terms of total nodes expanded and max queue size.

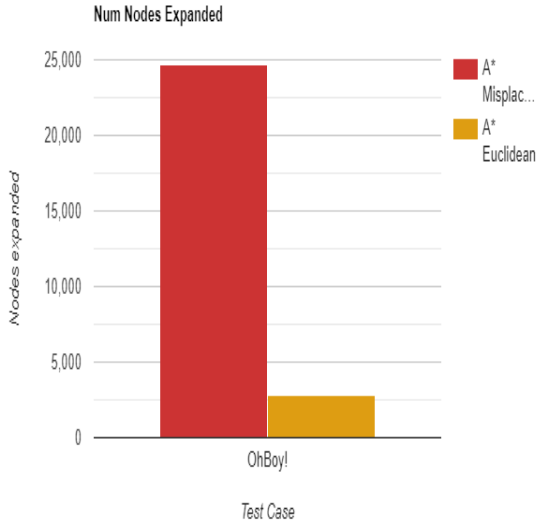
Test cases:

Trivial	Very Easy	Easy
1 2 3	1 2 3	1 2 0
4 5 6	4 5 6	4 5 3
7 8 0	7 0 8	7 8 6

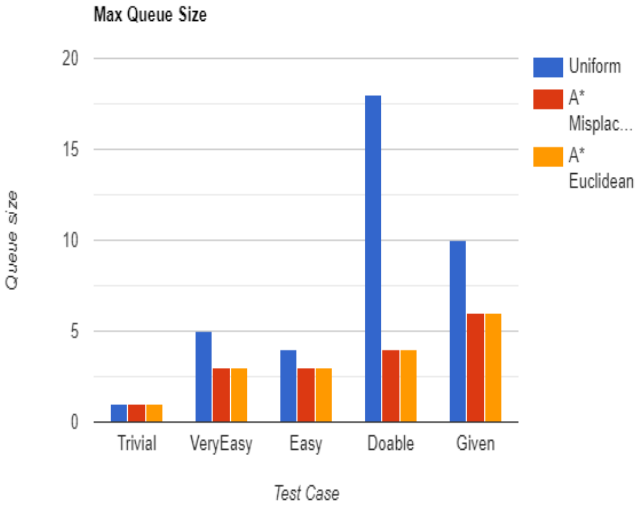
Doable	Oh Boy!	Given (Demo)
0 1 2	8 7 1	1 0 3
4 5 3	6 0 2	4 2 6
7 8 6	5 4 3	7 5 8



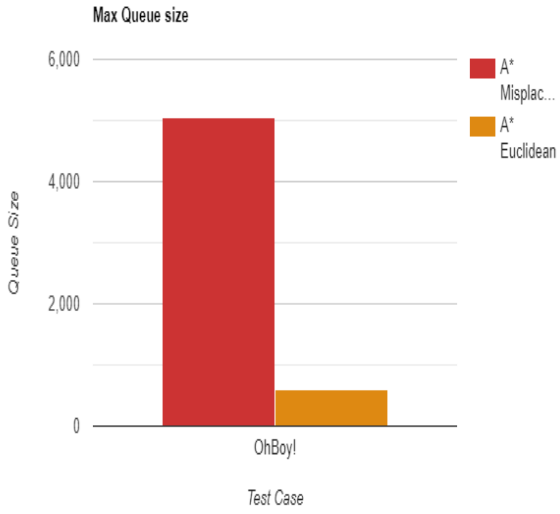
The first image here shows the A* searches being about equal for the first five cases, while the uniform cost search algorithm worked for really simple cases, but struggled more on the harder cases.



In this second chart, I placed the Oh Boy! test case on it's own as, it's values were in a league of its own. Furthermore, uniform cost search was excluded here as it's execution took too long to compare. Here we can see that the euclidean search is far more efficient in time on the long scale than misplaced tile.



In this image depicting queue size, we again see the A* algorithms being similar for these five cases, as uniform cost can only keep up for very simple cases.



Again, the euclidean search heuristic looks to save more space overall than the misplaced tile heuristic.

Final Takeaways

Given my results, I can safely say that a blind search, such as uniform cost will perform far less efficiently in terms of time and space used to reach to the solution, when compared to an informed search such as A*.

Additionally, while A* searches are most definitely better than blind searches, differing heuristics for A* are likely only equivalent for smaller and simpler search cases. For longer searches, it is clear that A* with euclidean heuristic outclasses A* with misplaced tile heuristic in terms of time and space complexity.

In this way, it can be inferred that not all forms of an informed search are necessarily going to perform similarly, and there will be room for optimization.

Resource Links

General:
https://docs.python.org/3/contents.html - pythons documentation
Lecture slides and discussions on blind and heuristic search
A* search doc under course material additional reading

Formulas and understanding of search algorithm help:
https://medium.com/@kunal_gohrani/different-types-of-distance-metrics-used-in-machine-learning-e9928c5e26c7
https://en.wikipedia.org/wiki/Euclidean_distance

Code convention:
https://www.askpython.com/python/string/extract-digits-from-python-string
https://stackoverflow.com/questions/4289331/how-to-extract-numbers-from-a-string-in-python
https://pybind11.readthedocs.io/en/stable/advanced/pycpp/index.html
https://stackoverflow.com/questions/7604966/maximum-and-minimum-values-for-ints
https://www.w3schools.com/python/ref_func_max.asp

Deepcopy:
https://stackoverflow.com/questions/2612802/list-changes-unexpectedly-after-assignment-why-is-this-and-how-to-prevent-it/2612815#2612815

A* Euclidean Distance Heuristic Search for given matrix: ([[1, 0, 3], [4, 2, 6], [7, 5, 8]])

Welcome to Alejandro Sherman's 8-puzzle solver.
Type '1' to use a default puzzle, or '2' to enter your own puzzle.

2
Enter your puzzle, use a zero to represent the blank and press enter when done with each step.
Enter the first row, using a space between numbers: 1 0 3
Enter the second row, using a space between numbers: 4 2 6
Enter the third row, using a space between numbers: 7 5 8
Choice of algorithms to use:
1. Uniform Cost Search
2. A* with Misplaced Tile Heuristic
3: A* with Euclidean Distance Heuristic

3
Initiating A* with Euclidean Distance Heuristic on
[1, 0, 3]
[4, 2, 6]
[7, 5, 8]

The best state to expand with a g(n) = 0 and h(n) = 3.0 is...
[1, 0, 3]
[4, 2, 6]
[7, 5, 8]
Expanding this node...

The best state to expand with a g(n) = 1 and h(n) = 2.0 is...
[1, 2, 3]
[4, 0, 6]
[7, 5, 8]
Expanding this node...

The best state to expand with a g(n) = 2 and h(n) = 1.0 is...
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]
Expanding this node...

Solution found!

Number of nodes expanded: 10
Max queue size: 6