Project 1 Report

**Project Setup**

* Python version: 2.7
* Developed using: PyCharm for Mac

**Sources**

* I spoke with Riley Marzka and Nathan Harris about this assignment. We discussed clarifying questions about project requirements, such as what heuristics are required, and which search algorithms require a visited list.

**Bugs**

* There are no (known) bugs in my project.

**Notes**

* All test outputs mentioned in the Report Specification are located in the *report/* directory at the root of the project. There is a zip file for each puzzle containing output text files for the puzzles below.
* All configuration files should be added to the *resources/* directory at the root of the project.
* My implementation of the cities game constructs the full graph of cities prior to search. This is because edges are pre-defined in the configuration file and are not defined by a set of actions. I could hold the data in another data structure and create new states dynamically as I do in the jugs and tiles puzzles, but this would require a lot of unnecessary work with little time/space improvement.

**Cities**

Action Expansion Order

1. *The order by which a city’s edges appear in the configuration file*

Heuristic Functions (command line argument in parentheses)

1. *Manhattan Distance (md)*: vertical + horizontal distance from cities (x1, y1) to (x2, y2)
2. *Euclidean Distance (eu)*: distance along diagonal from cities (x1, y1) to (x2, y2)

Analysis

The Greedy algorithm boasted the best space complexity in solving the Cities puzzle. When using the Euclidean Distance and Manhattan Distance heuristics, both functions resulted in a maximum frontier size of 30 and visited size of 12. These were the best recorded results, aside from DFS which always has a visited size of 0. A\* had a similar space complexity but was slightly worse. With a larger map, A\* may outperform Greedy.

While the Greedy algorithm boasted the best space complexity, all puzzles shared the same time complexity with 25 nodes. This is because the full graph is created at initialization, rather than dynamically. I discussed my reasoning for this above in *Notes*.

For the Cities problem, Both A\* and the Greedy algorithm are optimal and complete, meaning they always find the solution if one exists, and the solution found will be (one of) the best. Unicost will also be optimal and complete in this problem.

In contrast, BFS and DFS will struggle to find the optimal solution because they don’t consider edge costs. BFS will always find a solution if one exists, meaning its complete.

**Jugs**

Action Expansion Order

1. *Moves that require filling a jug, sorted by amount to fill from largest to smallest*
2. *Moves that require emptying a jug, sorted by amount to empty from largest to smallest*
3. *Moves that require transferring water between jugs, sorted by the amount being transferred from largest to smallest*

Heuristic Functions (command line argument in parentheses)

1. *Distance from Goal (d)*: the sum of the differences between each jug’s capacity and its respective goal state’s capacity
   1. Example: h([0, 3] w/ goal state of [1, 4]) = (1-0) + (4-3) = 2
2. *Misplaced Jugs (mj)*: number of jugs that are not currently in their respective goal states

Analysis

The Greedy and A\* algorithms both showed the best space complexity in solving the Jugs puzzle. The algorithms’ results were identical because edge cost is always 0 for A\*. When using the Distance and Misplaced Jugs heuristics, both functions resulted in a maximum frontier size of 3 and visited size of 14. These were again the best recorded results, aside from DFS which always has a visited size of 0.

The Misplaced Jug heuristic would only be admissible if 1 capacity could be modified at a time. This is because the function increments the heuristic value for each jug not in its respective goal state; however, two jugs could be brought into their respective goal states in one action, meaning the heuristic was an overestimate. Under similar logic but instead measuring distance as gallons of water from the goal state, the Distance heuristic is also not admissible.

The Greedy and A\* algorithms also boasted the best time complexity with 18 nodes created. This doesn’t surprise me considering the other algorithms do not use estimates of how much distance is remaining. BFS and Unicost, which act like the same algorithm when no edge costs exist, were close behind at 20 nodes created.

For the Jugs problem, A\*, Greedy, Unicost, and BFS are optimal and complete. Unicost and BFS will also be optimal and complete in this problem, but DFS will not.

**Tiles**

Action Expansion Order

1. *Move blank tile Right*
2. *Move blank tile Up*
3. *Move blank tile Left*
4. *Move blank tile Down*

Heuristic Functions (command line argument in parentheses)

1. *Manhattan Distance (md)*: vertical + horizontal distance between two tiles
2. *Misplaced Tiles (mt)*: number of tiles that are not currently in their respective goal states
3. *Euclidean Distance (eu)*: distance along Pythagorean diagonal between two tiles

Analysis

The Greedy and A\* algorithms boasted the best space complexity in solving the Tiles puzzle. When using the Euclidean Distance and Manhattan Distance heuristics, both functions resulted in a maximum frontier size of (roughly) 40 and visited size of (roughly) 30. Surprisingly, the Misplaced Tile heuristic did worse than the other two functions, with a maximum frontier size of 2,143 and a maximum visited size of 1,755.

The Greedy and A\* algorithms also boasted the best time complexity with (roughly) 70 nodes created. DFS, BFS, and Unicost all solved the initial tile configuration provided, but take significantly more time (BFS/Unicost was 500,000+ iterations of the search).

For the Jugs problem, A\*, Greedy, Unicost, and BFS are optimal and complete, although Unicost and BFS will be extremely slow. DFS will be optimal and complete as well since there are a limited number of actions to take from each state.