Anthony Poerio
adp59@pitt.edu
University of Pittsburgh
CS1571 – Artificial Intelligence
Homework #01
Puzzlesolver – Report

# Overview

The ultimate goal of Artificial Intelligence has long been the creation of a 'generalized intelligence'—a system able to solve ***any*** abstract problem. While we are far from achieving that goal (and may never really do so, at least as commonly conceived), we are indeed able to create systems able to solve a large subset of problems, as long as we are able to model such problems in a way amenable to our system.

For this first homework in Dr. Hwa's AI class for Fall of 2016, we have been tasked with making a "puzzlesolver" system. That is, we have been tasked with creating a system that allows is able to solve three (3) different puzzle types using each of seven (7) separate AI search algorithms.

The problems modeled in this project are:

- The Water Jugs Problem
- The Path Planning Problem,
- The Burnt Pancakes problem.

The algorithms used to solve each are:

- Breadth First Search
- Depth First Search
- Iterative Deepening Depth First Search
- Unicost Search
- Greedy Search
- A* Search
- Iterative Deepening A* Search

More concretely the goal of this assignment is to determine the best search algorithm to solve each of our three problems.

This report, I will discuss the results obtained by solving each puzzle with all seven algorithms. After analyzing the resulting data, I will make a qualitative decision regarding which search algorithm is the best option for each puzzle.

To see the raw data used, you may view:

1. Transcripts of my program runs, in the **_transcripts** folder

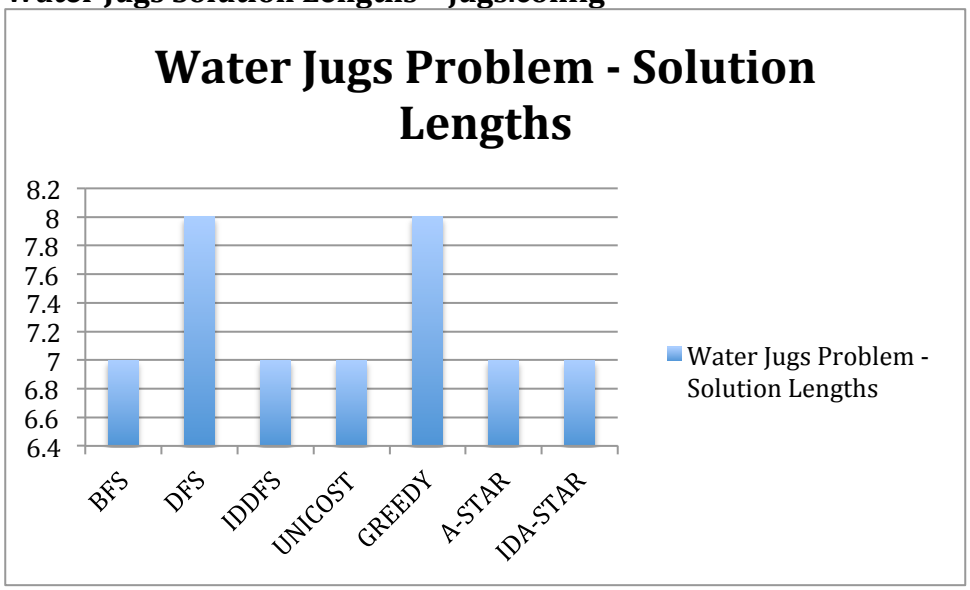
2. The collated data and graphs, in the excel file named "**Algorithm_Data.xlsx**"

# Water Jugs

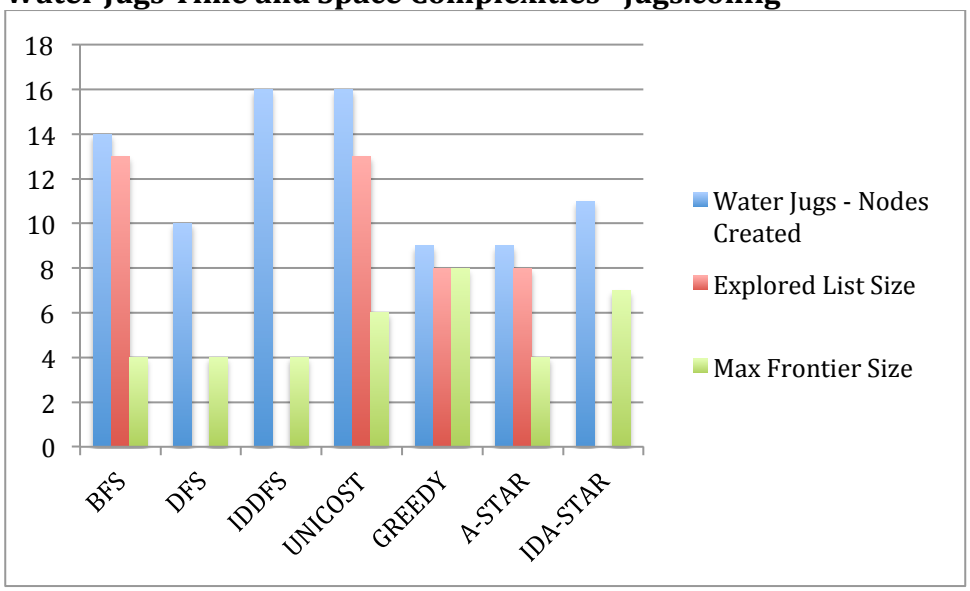
To begin, I ran every search algorithm on our first example of the Water Jugs problem, with the data found in the config file named **jugs.config**.

The collated results are as follows.

**Water Jugs Solution Lengths – jugs.config**




**Water Jugs Time and Space Complexities– jugs.config**



**Optimality:**
From this data, we can see that 5 of the search algorithms found what is presumably the optimal solution, length of 7. (Or at least the lowest that any of my algorithms discovered.)

**Space Complexity:**
As expected, Unicost and BFS have the **largest** explored lists. The maximum frontier size was largest for Greedy. And, the DFS derivative searches had the **best** space complexity overall, because they do NOT use an explored list.

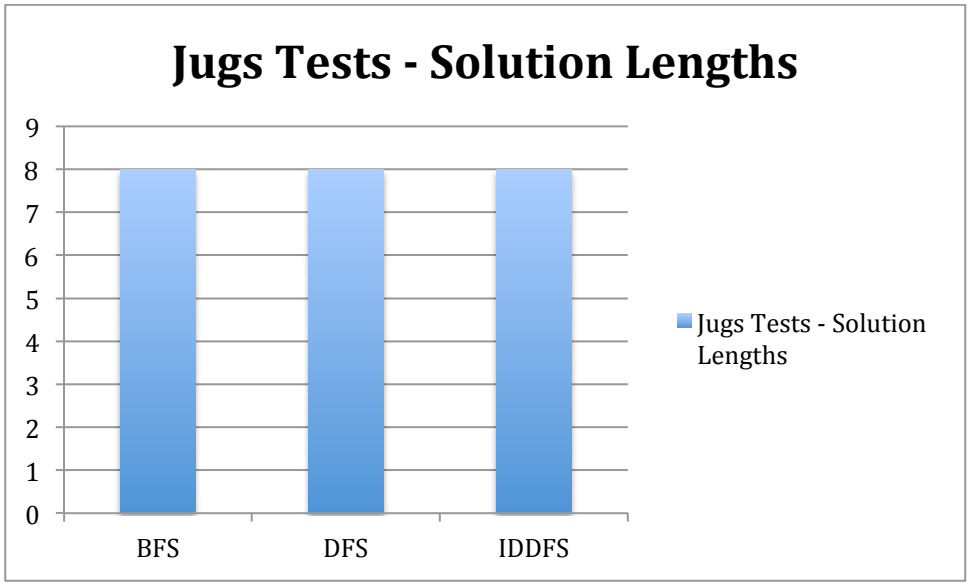
**Time Complexity:**
The best overall time complexity (judging by nodes created) was shared by the Greedy and A* searches. The worst time complexity was shared by two of the complete searches, IDFS and Unicost.

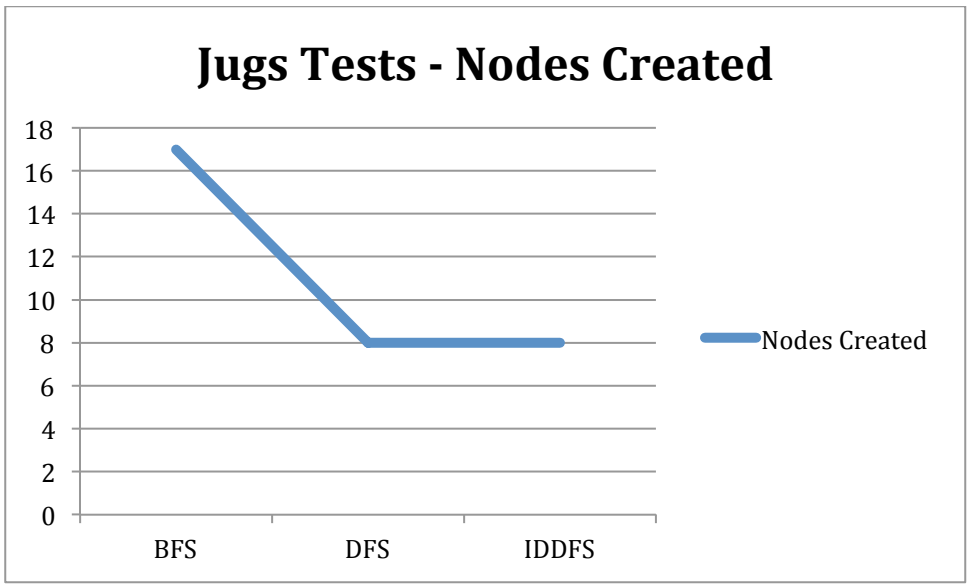
**Completeness:**
BFS, IDDFS, Unicost, A*, and ID-A* are all complete. So ideally the best search algorithm will be drawn from this list because completeness does matter in the Water Jugs problem.

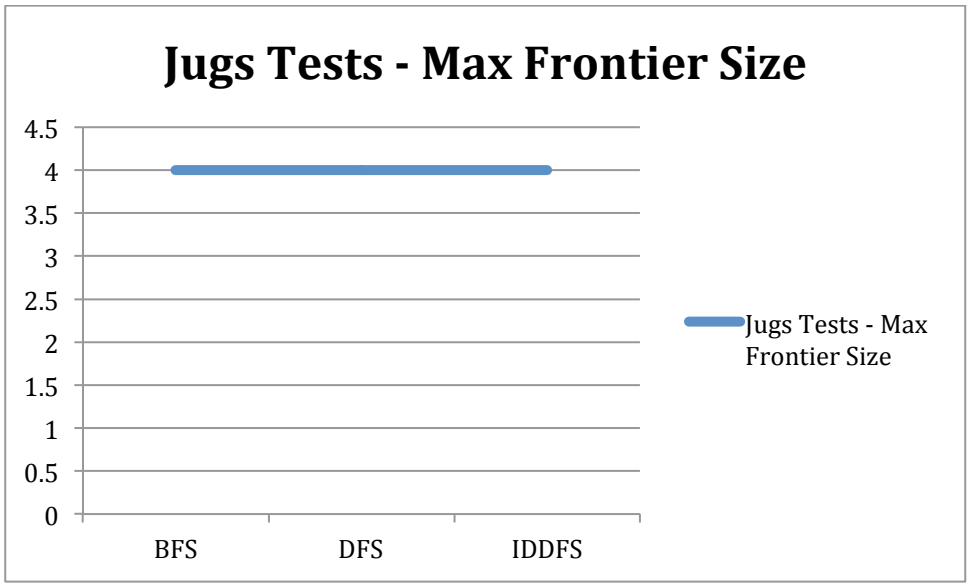
--

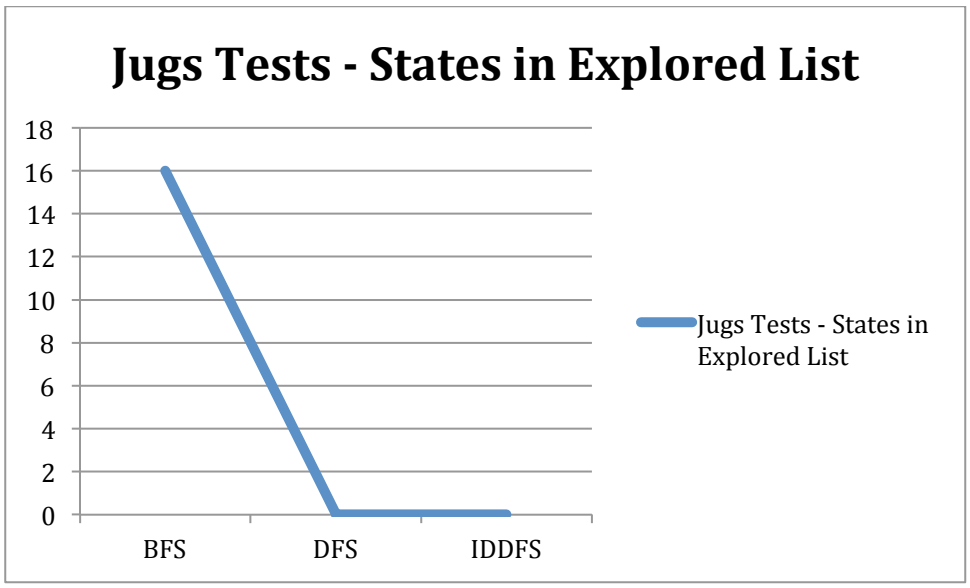
**Part II Tests**
Next, I further tested the Water Jugs problem by running the **test_jugs.config** file on the BFS, DFS, and IDDFS search algorithms, as specified in Part II of the assignment.

The results were as follows:




Jugs Tests - Nodes Created
18
16
14
12
10
8
6
4
2
0
BFS
DFS
IDDFS
Nodes Created


Jugs Tests - Max Frontier Size
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
BFS
DFS
IDDFS
Jugs Tests - Max Frontier Size




**Decision**

Weighing these results in tandem with those of the first test**, I believe that the best algorithm for Water-Jugs is IDDFS**.

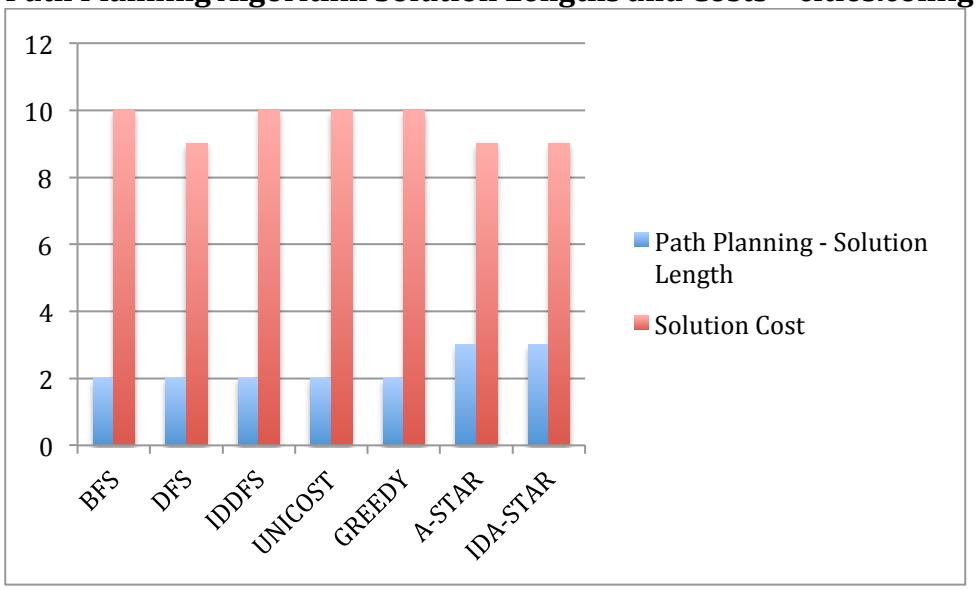
This is because: 1) It has lower space complexity since it does not use an explored list. 2) It is complete, so we are guaranteed to find an answer if one exists. 3) The not always, It has the possibility that fewer nodes will be created, and hence the time complexity will be better overall. 4) The algorithm is optimal, since it runs similar to a BFS. So we'll also get the optimal solution, not just 'any' solution.

# Path-Planning

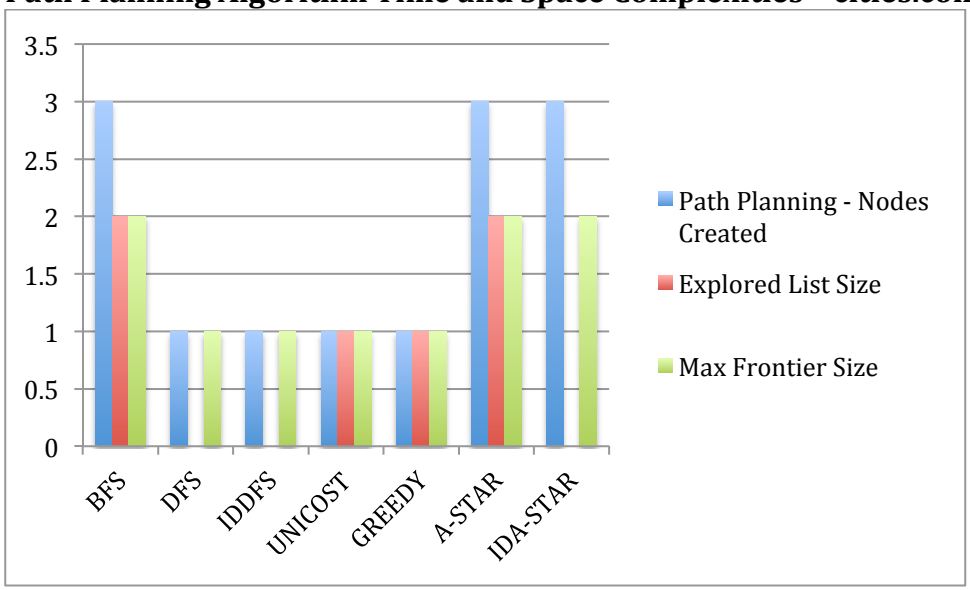
Next, I ran every search algorithm on our first example of the Path Finding problem, with the data found in the config file named **cities.config**.

The collated results are as follows.

**Path Planning Algorithm Solution Lengths and Costs – cities.config**




**Path Planning Algorithm Time and Space Complexities – cities.config**




**Optimality:**

Interestingly, we have 2 competing ideas of optimality for path-finding problem. 1) Number of nodes traversed; and 2) Total cost of those nodes. Unicost, BFS, and the

complete search algorithms that do not account for weights (with a heuristic) performed the best with respect to number of nodes, while those algorithms that used a heuristic (informed search) outperformed with regards to total lengths here. Because cost is more important to the path planning problem, I would suggest that we want to use an informed search for this problem – A* and ID-A* performed the best from the informed search group.

**Space Complexity:**
Here, BFS and A* have the **largest** explored lists. The maximum frontier size was largest for BFS, A* and ID-A*. As expected, DFS derivative searches had the **best** space complexity overall, because they do NOT use an explored list.

**Time Complexity:**
The best overall time complexity (judging by nodes created) was shared by DFS, IDDFS, Unicost, and Greedy. The worst time complexity was shared by three of the complete searches, BFS, A*, and ID-A*.

**Completeness:**
BFS, IDDFS, Unicost, A*, and ID-A* are all complete. So ideally the best search algorithm will be drawn from this list because completeness does matter in the Path-finding problem.

**Water Jugs Heuristic**
For the Water Jugs problem I chose to calculate the "Total Distance" from the solution state.

For example, in state (2,4), if we are searching for solution state (0,2), the heuristic would be: abs( [ (0-2) + (2-4) ]) = 4

Similar to Euclidean Distance, the goal of this heuristic is to identify how many much water needs to be moved, total, to reach the goal state. This heuristic is **admissible** because it does not overestimate the cost of reaching the goal, if we are using the total volume of water as costs. Moreover, it is **consistent**, because it is exactly equal to the 'distance' between neighboring nodes and the goal state.

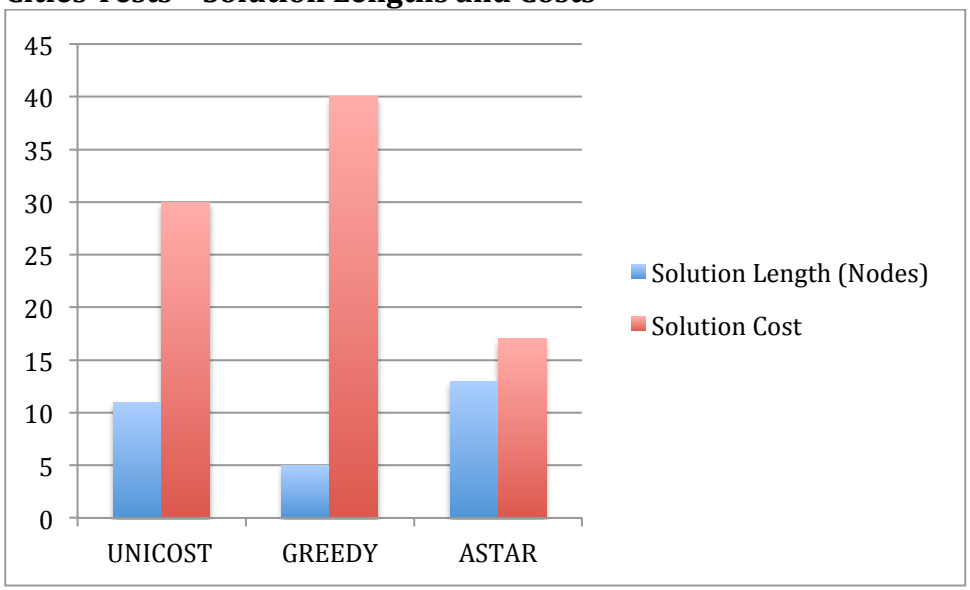
--

**Part II Tests**

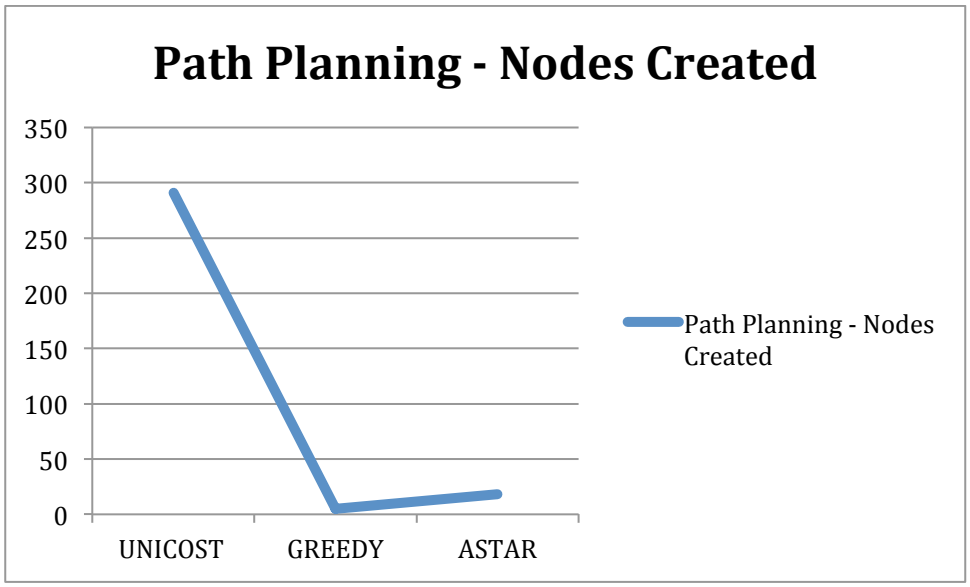
Next, I further tested the Water Jugs problem by running the **test_cities.config** file on the Unicost, Greedy and A* search algorithms, as specified in Part II of the assignment.

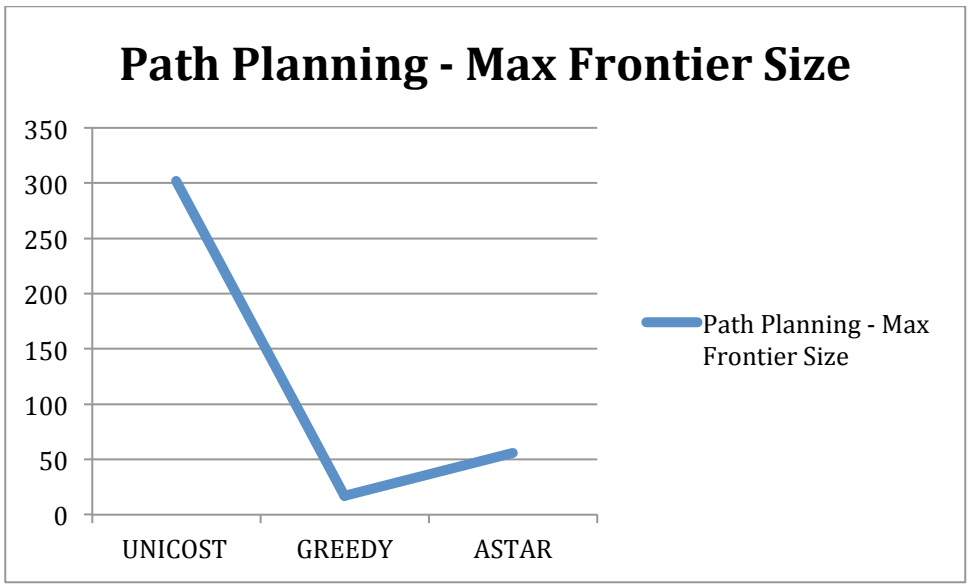
The results were as follows:

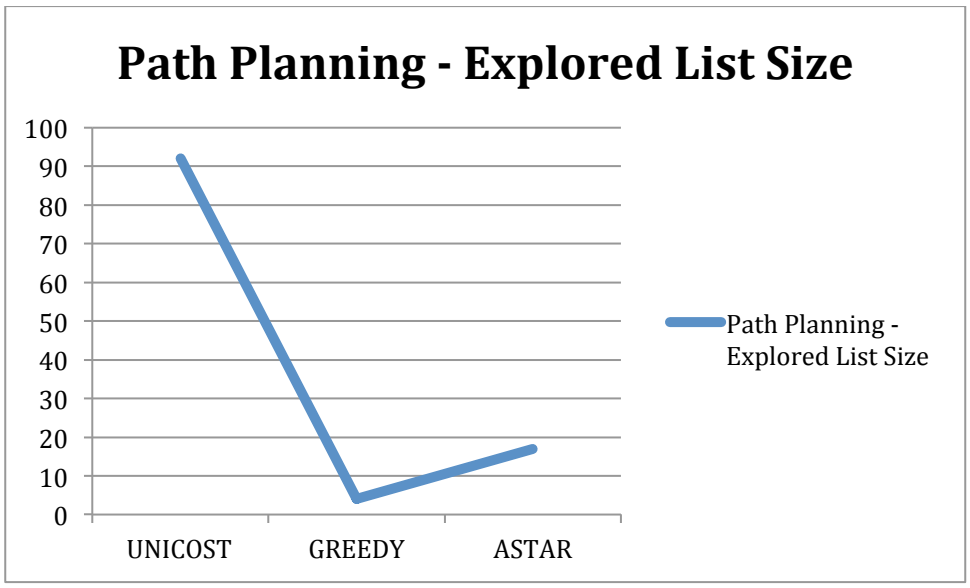
**Cities Tests – Solution Lengths and Costs**




Again, we have a discrepancy between best result with respect to the number of nodes (Greedy), and the best result with respect to total cost (A*).






**Decision**

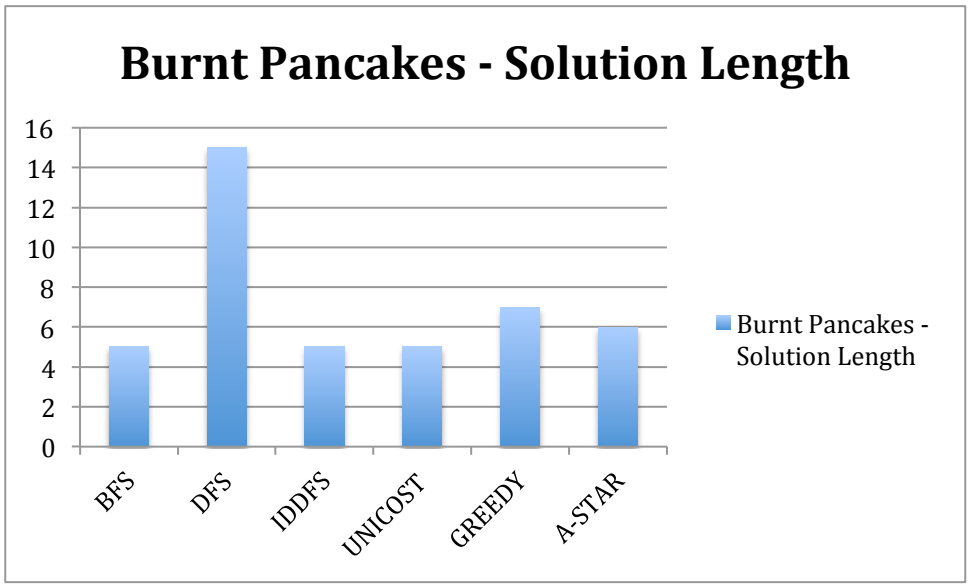
Weighing these results in tandem with those of the first test**, I believe that the best algorithm for Path-Planning puzzle is A***.

This is because: 1) A* is optimal, and we will find the best path with respect to cost, as long as we have an effective heuristic defined. 2) A* is complete, so we will find a solution if one does exist. 3) It achieves the best balance between space and time complexities, given that it will always find the best answer. However, because it relies on having a solid heuristic defined, we need to be careful, and Unicost would be a good choice we are in a position where a good heuristic is not attainable.

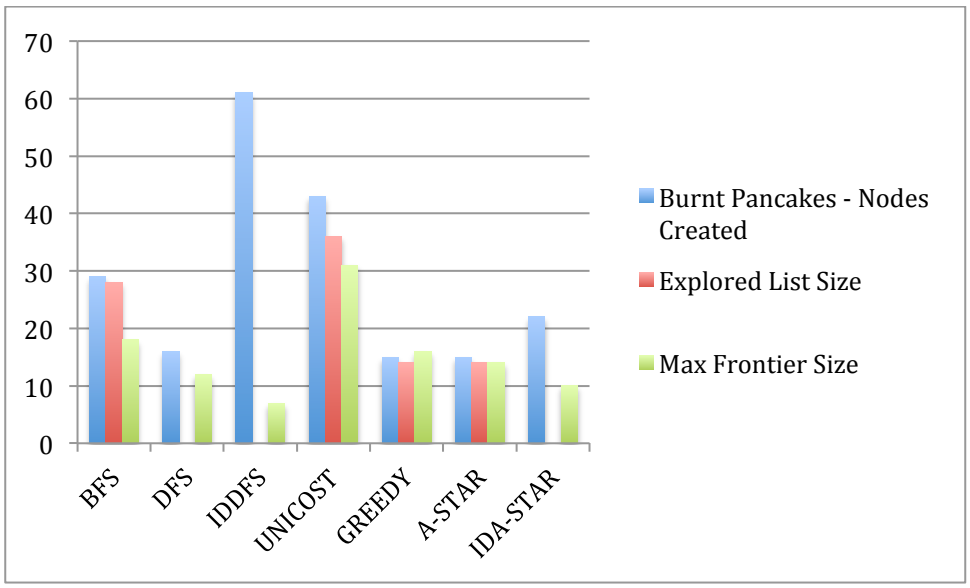
# Burnt Pancakes

Next, I ran every search algorithm on our first example of the Path Finding problem, with the data found in the config file named **small_pancakes.config**. This is a config file I created to ensure that the algorithms all run as expected for the burnt pancakes problem.

The collated results are as follows.




**Burnt Pancakes Puzzle - Time and Space Complexities – cities.config**



**Optimality:**
From this data, we can see that 3 of the search algorithms found what is presumably the optimal solution, length of 5 flips. These are 3 of the complete algorithms: BFS, IDDFS, and Unicost.

**Space Complexity:**
The **best** algorithm with respect to space complexity is the DFS-group, DFS, IDDFS, and ID-A*. Unicost and BFS performed the **worst**.

**Time Complexity:**
The best overall algorithms for time complexity were Greedy and A*. The worst were IDDFS, Unicost, and ID-A*.

**Completeness:**
BFS, IDDFS, Unicost, A*, and ID-A* are all complete. So ideally the best search algorithm will be drawn from this list because completeness does matter in the Water Jugs problem.

**Burnt Pancakes Heuristic**
For the burnt pancakes heuristic, I chose to find the total number of pancakes in a given state that are either: A) Burnt Side Up; or B) Not of adjacent size to the pancake below it.

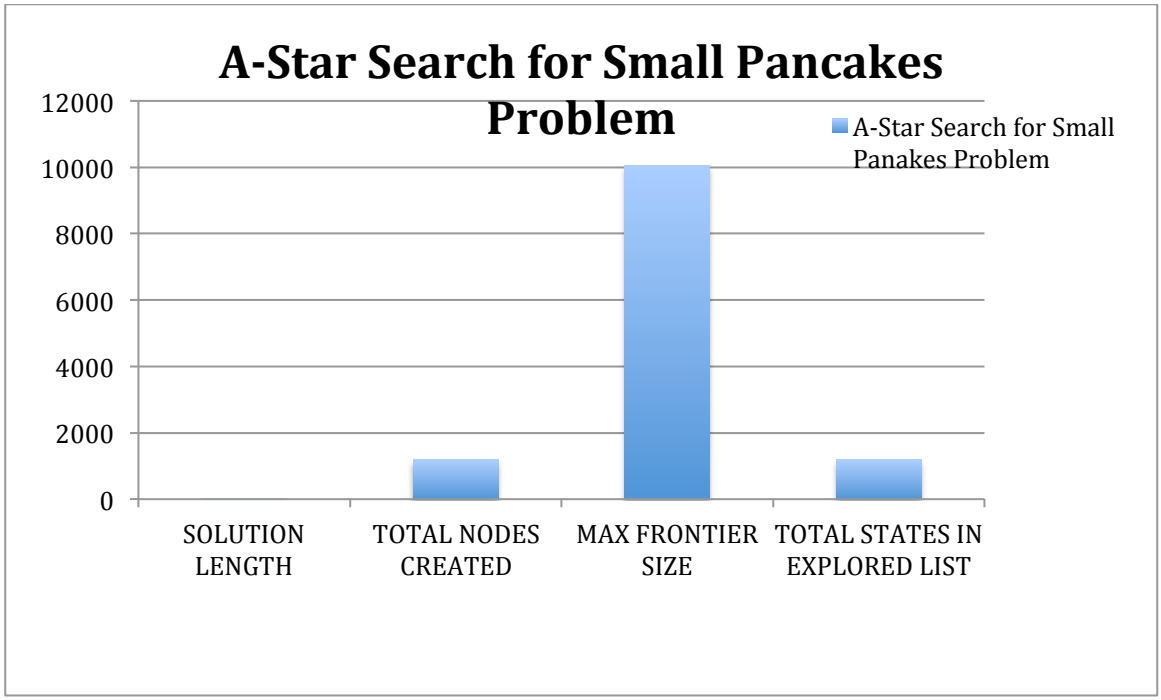
This is commonly called the "Gap Heuristic", and I discovered it while doing research on the problem, in this academic paper:
https://www.aaai.org/ocs/index.php/SOCS/SOCS11/paper/viewFile/4013/4360

--

**Part II Tests**
Next, I further tested the Water Jugs problem by running the **test_pancakes1.config** and **test_pancakes2.config** files, for the IDDFS, A* and ID-A* search algorithms, as specified in Part II of the assignment.

Unfortunately, because these problems are so large, I was only able to get one of them to complete. A* to for test_pancakes1.config. **The others ran for over 30 minutes.**

The results of that run are as follows:




**Decision**
Weighing these results in tandem with those of the first test**, I believe that the best algorithm for Path-Planning puzzle is A***.

This is because: 1) A* is optimal, and we will find the best path with respect to cost, as long as we have an effective heuristic defined. 2) A* is complete, so we will find a solution if one does exist. 3) It achieves the best balance between space and time complexities, given that it will always find the best answer.

Moreover, A* is the ONLY algorithm that completed in reasonable time I believe the best algorithm for the Burnt Pancake problem is A*.