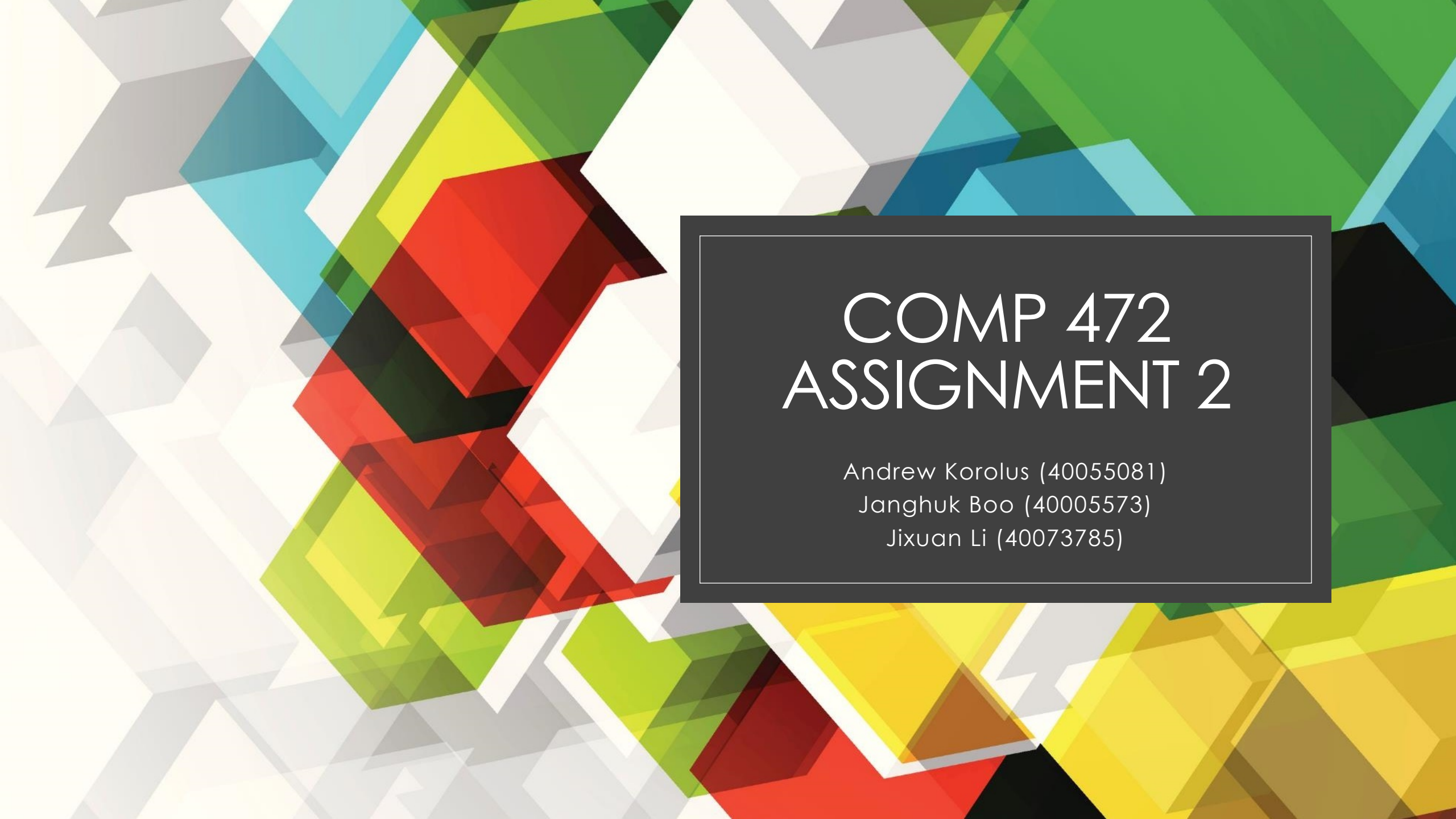


COMP 472 ASSIGNMENT 2

Andrew Korolus (40055081)
Janghuk Boo (40005573)
Jixuan Li (40073785)

Work Distribution

- Andrew:
  - Uniform Cost algorithm
  - All non-algorithm specific code (i.e. main.py, analytics.py, output_writer.py, random_input_generator.py, x_puzzle_solver.py, algorithms/helper.py (except for the heuristic functions), algorithms/node.py, scaling_up.py)
  - Demo Slides
- Janghuk:
  - A* algorithm
  - Contributed to the development of the heuristic functions
- Jixuan:
  - Greedy-Best-First-Search algorithm
  - Contributed to the development of the heuristic functions

Github link: https://github.com/AndrewK-7/Comp472-Assignment2.git

Heuristic Functions

- Heuristic 1:
  - For the first heuristic, the "Hamming Distance" calculation was used
  - The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different
  - We adjusted the calculation by multiplying the resulting value by two when the state is not continuous (i.e. not in the form of 1[,2,3,4,5,6,7,0] for example)
    - The reasoning behind this is to make the rather simple heuristic slightly more accurate to the actual cost, while keeping it admissible.
- Heuristic 2:
  - For the second heuristic, the "Manhattan Distance" calculation was used
  - Manhattan distance is usually preferred over the more common Hamming Distance when there is high dimensionality in the data.
- Heuristic 0:
  - The "default" heuristic was implemented as described in the assignment outline. This heuristic was not used in the 50-puzzle analysis.

Uniform Cost

Out of 50 randomly generated puzzles, there are the results that were found for this algorithm:

Total Solution Length	Total # of No Solutions	Total Search Length	Total Cost of All Moves	Total Execution Time
15	48	69,160	17	2909.44 s

Avg Solution Length	Avg No Solutions	Avg Search Length	Avg Cost of each Move	Average Execution Time
7.5	96%	34,580	1.13	58.19 s

Greedy-Best-First-Search

Out of 50 randomly generated puzzles, there are the results that were found for this algorithm using $\mathbf{h_1}$:

Total Solution Length	Total # of No Solutions	Total Search Length	Total Cost of All Moves	Total Execution Time
913	0	4,065	1,475	49.48 s

Avg Solution Length	Avg No Solutions	Avg Search Length	Avg Cost of each Move	Average Execution Time
18.26	0%	81.3	1.61	0.99 s

Greedy-Best-First-Search

Out of 50 randomly generated puzzles, there are the results that were found for this algorithm using $\mathbf{h_2}$:

Total Solution Length	Total # of No Solutions	Total Search Length	Total Cost of All Moves	Total Execution Time
1,007	0	5,109	1,326	69.99 s

Avg Solution Length	Avg No Solutions	Avg Search Length	Avg Cost of each Move	Average Execution Time
20.14	0%	102.18	1.31	1.40 s

A*

Out of 50 randomly generated puzzles, there are the results that were found for this algorithm using $\mathbf{h_1}$:

Total Solution Length	Total # of No Solutions	Total Search Length	Total Cost of All Moves	Total Execution Time
544	5	17,398	691	663.28 s

Avg Solution Length	Avg No Solutions	Avg Search Length	Avg Cost of each Move	Average Execution Time
12.09	10%	386.62	1.29	13.27 s

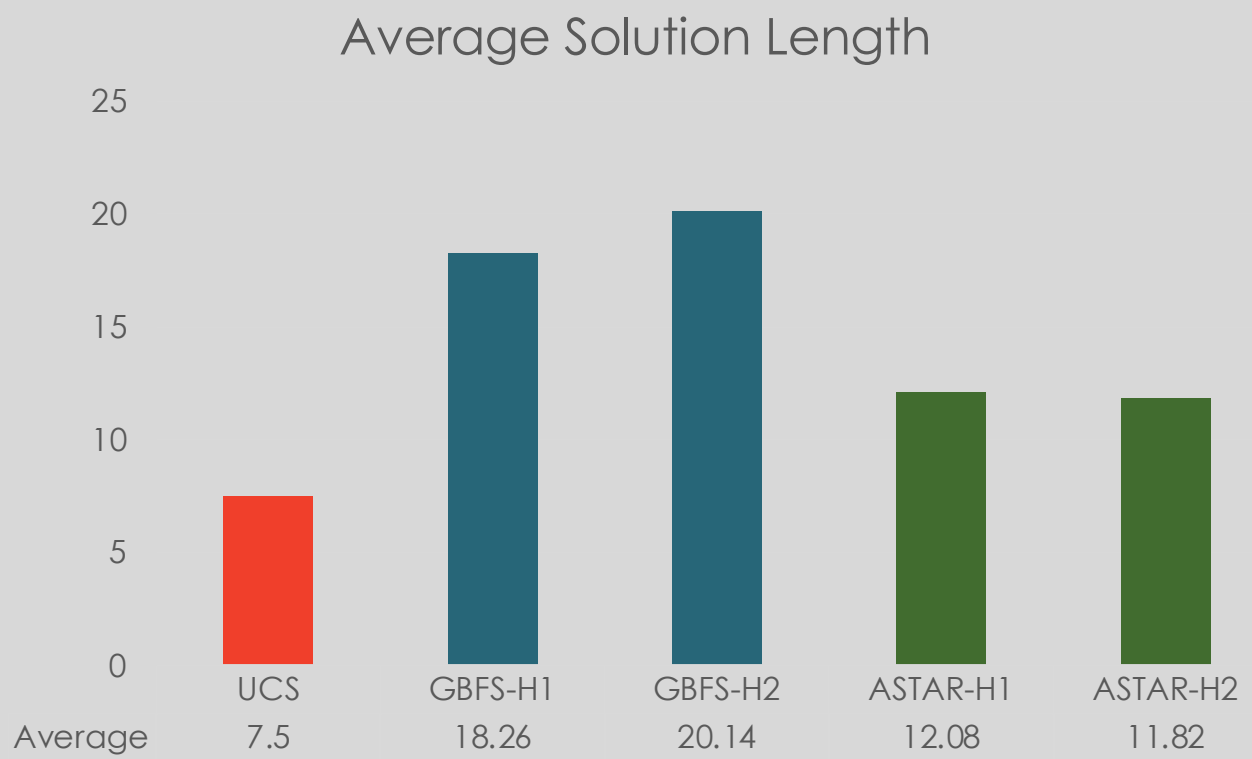
A*

Out of 50 randomly generated puzzles, there are the results that were found for this algorithm using $\mathbf{h_2}$:

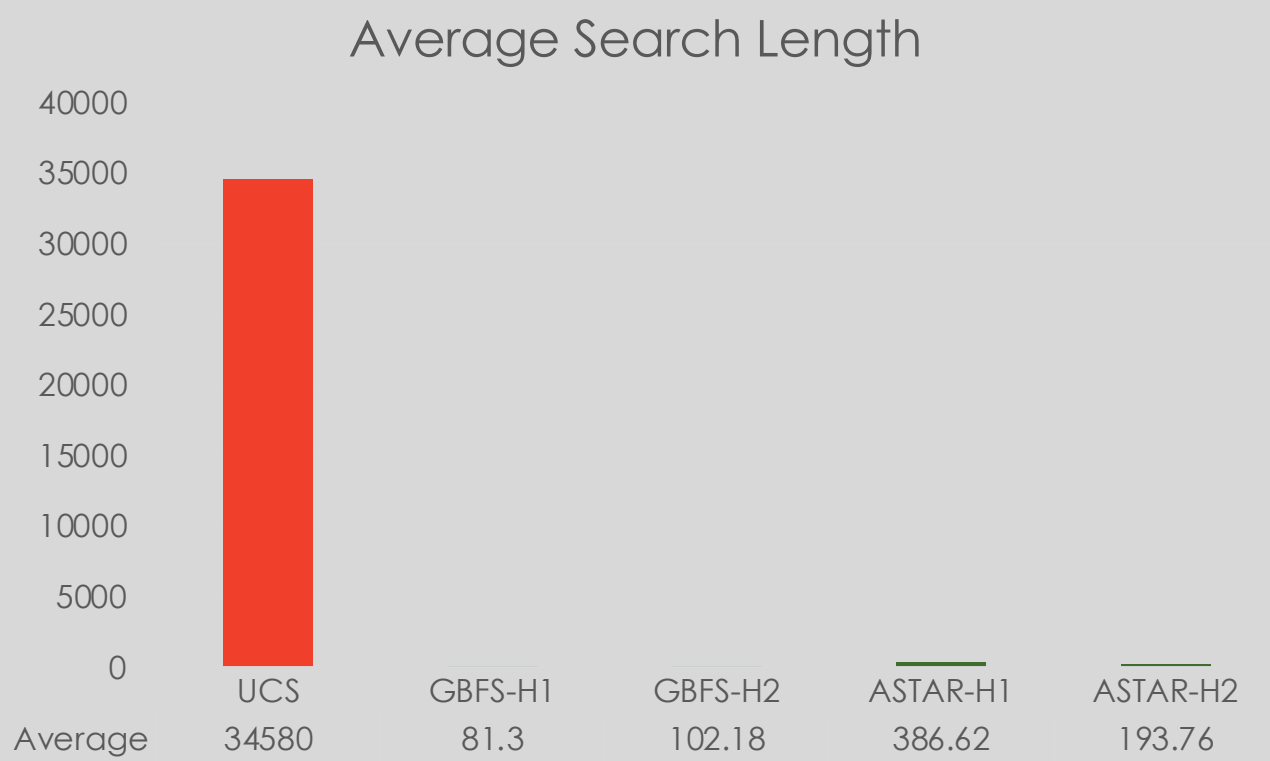
Total Solution Length	Total # of No Solutions	Total Search Length	Total Cost of All Moves	Total Execution Time
591	0	9,688	681	266.86 s

Avg Solution Length	Avg No Solutions	Avg Search Length	Avg Cost of each Move	Average Execution Time
11.82	0%	193.76	1.17	5.34 s

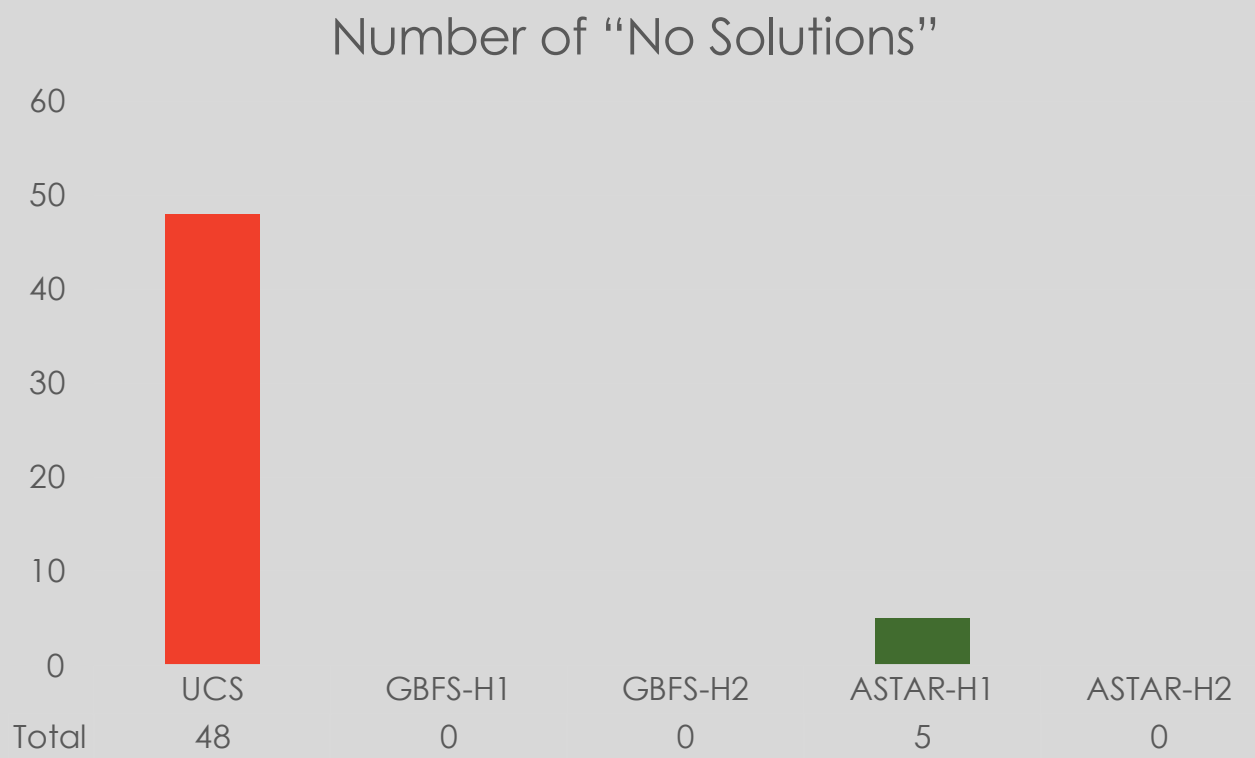
Comparing the Different Algorithms



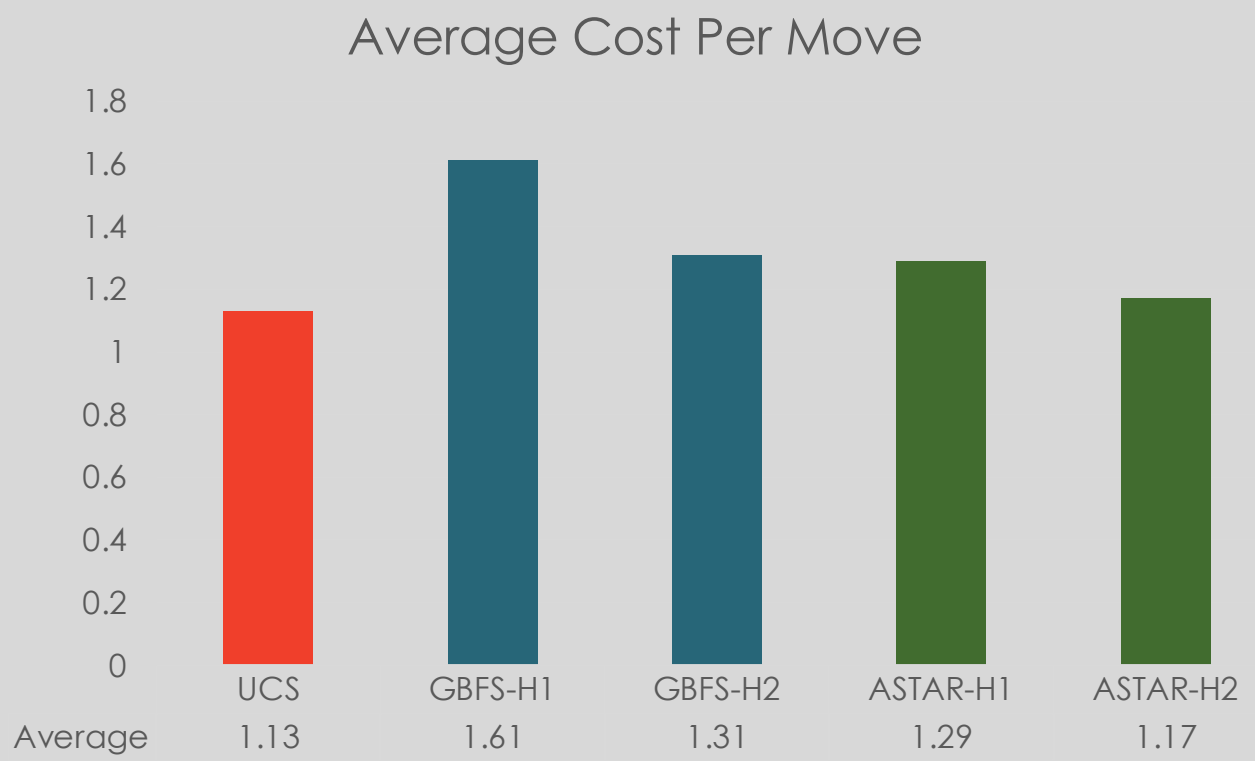
Comparing the Different Algorithms



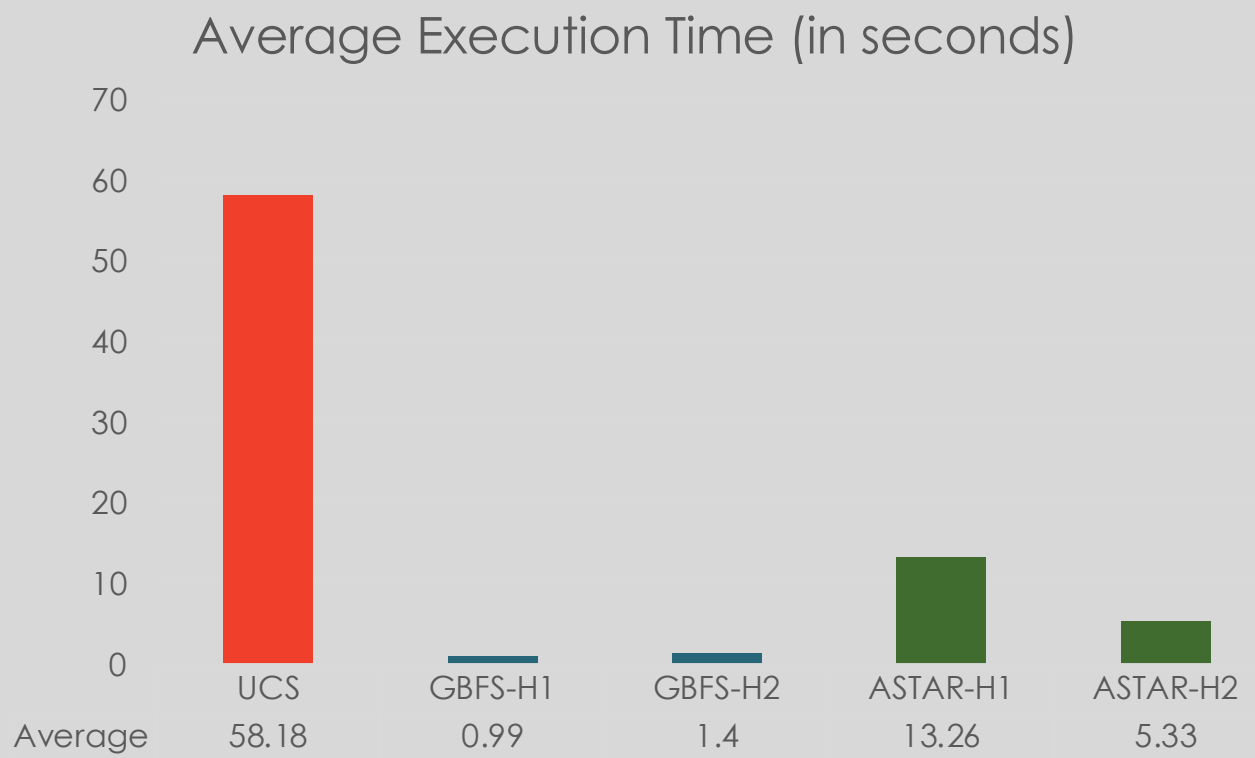
Comparing the Different Algorithms



Comparing the Different Algorithms



Comparing the Different Algorithms



Optimality of the Solution Paths

Uniform Cost	GBFS-H1	GBFS-H2	ASTAR-H1	ASTAR-H2
- Fairly short (not much data to reference, therefore inaccurate) - Low average cost (again, inaccurate due to small sample size) - Based on the puzzles that were actually solved, solution paths are optimal.	- Fairly long - Highest cost - Worst resulting optimality of solution paths (excluding UCS) - H1 produced slightly less optimal solution paths due to it being less informed - GFBS does not use g(n) so the costs will be higher than A*	- Longest length - Average cost - Second worst optimality of solution paths (excluding UCS) - H2 produced slightly more optimal solution paths due to it being more informed - GFBS does not use g(n) so the costs will be higher than A*	- Fairly short - Average cost - Second best optimality of solution paths (excluding UCS) - H1 produced slightly less optimal solution paths due to it being less informed - A* uses g(n) which is why it had lower costs over GBFS	- Shortest length - Lowest cost - Best resulting optimality of solution paths (excluding UCS) - H2 produced slightly more optimal solution paths due to it being more informed - A* uses g(n) which is why it had lower costs over GBFS

Our “Best” Performing Algorithm

We decided that our “best” performing algorithm is A* using H_2

- Had the best average solution path length and cost (most optimal solution paths were found)
- Was also the quick to execute (even compared to GBFS)
- Resulted in ALL puzzles being solved (i.e. no “no solutions” or timeouts)
- Very reasonable search paths, especially compared to A* using H_1

Scaling Up – Tests with A* using H_2

- Ten tests were performed, with a larger puzzle size each time, starting with a 2x2 puzzle
- A 60-minute timeout value was set
- Execution times increased exponentially
- Maximum solvable size seems to be ~4x3 puzzles
- The time it takes to solve puzzles increases exponentially