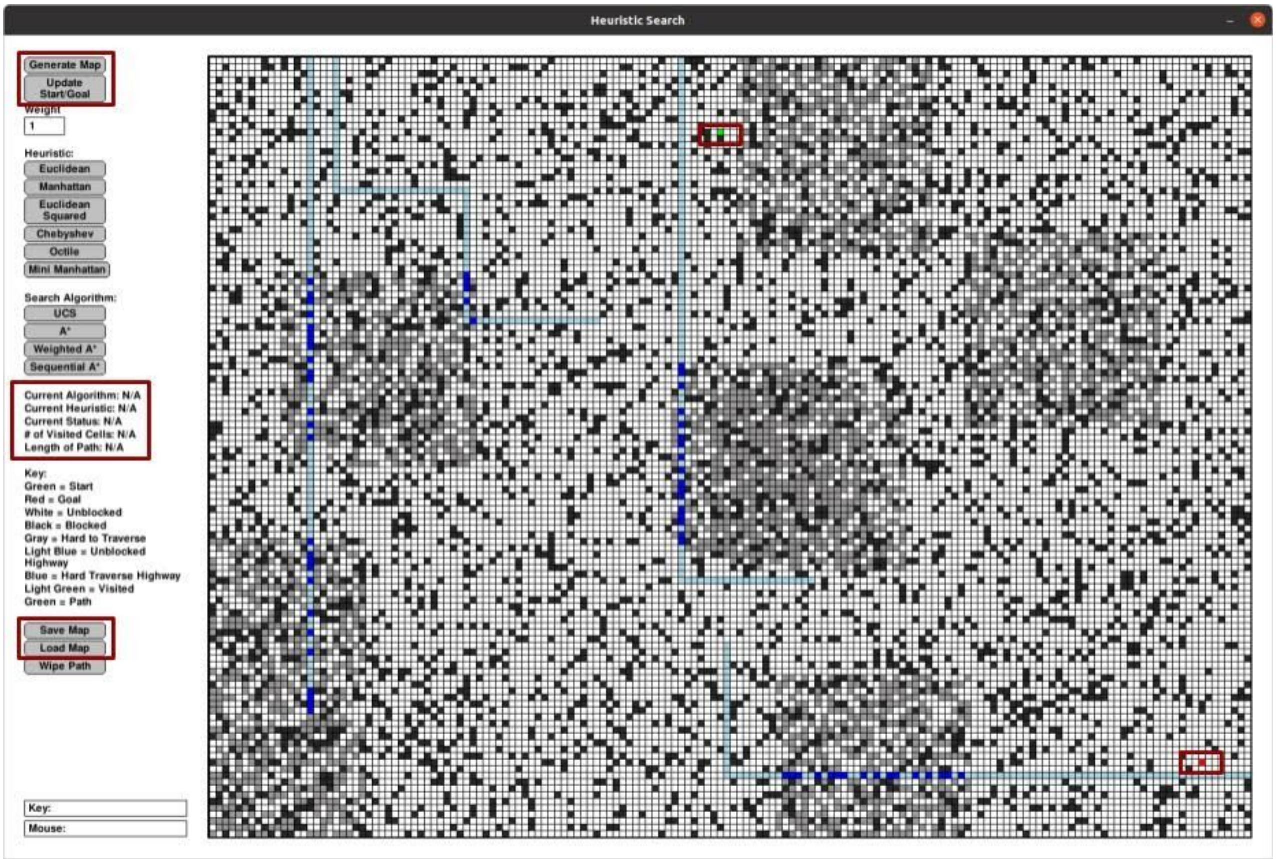
Homework 1 Report
Arthur DiLeo

1. Create an interface so as to create and visualize 50 eight-neighbor benchmark grids you are going to use for your experiments, which correspond to:
    a. 5 different maps as described above
    b. For each map, generate 10 different start-goal pairs for which the problem is solvable.

    Your software should be able to load a file representing a valid map and visualize: the start and the goal location, the different terrain types on the map (e.g., use different colors or symbols) and the path computed by an A -family algorithm. You should be able to visualize the values h, * g and *f* computed by A -family algorithms on each cell (e.g., after selecting with the mouse a * specific cell, or after using the keyboard to specify which cell's information to display). Use the images in this report from the traces of algorithms as inspiration on how to design your visualization. (10 points)



Weight: 1

Current Algorithm: N/A
Current Heuristic: N/A
Current Status: N/A
# of Visited Cells: N/A
Length of Path: N/A

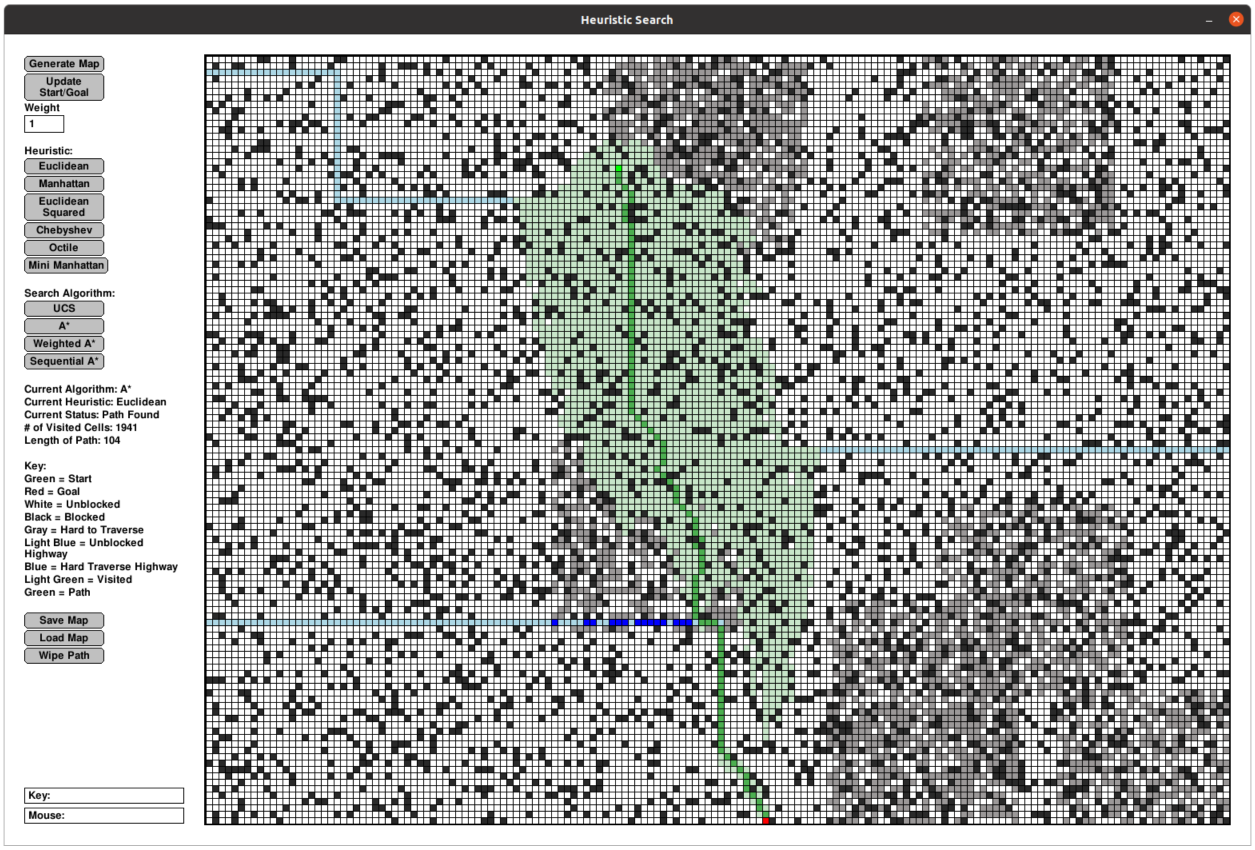
Key:
Green = Start
Red = Goal
White = Unblocked
Black = Blocked
Gray = Hard to Traverse
Light Blue = Unblocked Highway
Blue = Hard Traverse Highway
Light Green = Visited
Green = Path

Key:

Mouse:

Through the UI, you are able to see the start and goal location through a green and red cell, respectively. The different terrain is outlined on the UI through a color key combination. In the terminal, you are also able to see the live h, g, and f values as computed by A*. Furthermore, I offered an option to save and load a map where the filename is specified in terminal. All of this functionality can be visualized in the demo.

2. Implement an abstract heuristic algorithm and three instantiations of it: Uniform-cost search, A * * and Weighted A (it should be easy from the interface to define the weight w). Try to follow a modular implementation so that the code you have to write for each one of the concrete algorithms is minimized relative to the abstract implementation (e.g., take advantage of inheritance in C++ or Java, etc.) Show that you can compute correct solutions with these methods. (15 points)



Weight
1

Current Algorithm: A*
Current Heuristic: Euclidean
Current Status: Path Found
# of Visited Cells: 1941
Length of Path: 104

Key:
Green = Start
Red = Goal
White = Unblocked
Black = Blocked
Gray = Hard to Traverse
Light Blue = Unblocked Highway
Blue = Hard Traverse Highway
Light Green = Visited
Green = Path

Key:
Mouse:

As pictured through the UI above, the user is able to select which of the algorithms they wish to test against the grid. It is easy to change and review the statistics of each path finding algorithm through the menu on the side.

3. Optimize your implementation of the above algorithms. Discuss your optimizations in your report. (10 points)
In each of my implementations of the algorithm, I was able to utilize a priority queue which limits the amount of memory required to run the program. Furthermore, with the vast amount of heuristics provided, the user is able to get a sense of different types of optimizations throughout the path finding program. Using different weights in the program also exemplifies the optimizations a user can make throughout their experience. Each cell maintains a system where you can easily compare cells by position and all cells have a precalculated h value so that the

algorithm does not have to re-calculate each time. As for finding cell neighbors, I came up with an efficient way to quickly compute them in O(1) time.

4. Propose different types of heuristics for the considered problem and discuss them in your report. In particular:
    a. Propose the best admissible/consistent heuristic that you can come up with in this grid world.
    b. Propose at least four other heuristics, which can be inadmissible but still useful for this problem and justify your choices.

   Remember that good heuristics can effectively guide the exploration process of a search algorithm towards finding a good solution fast and they are computationally inexpensive to compute. (10 points)

   Manhattan - this is an admissible heuristic (never overestimates the cost of reaching a goal) because the distance computed for every path will have to be moved at least the number of spots in between itself and its correct position. Computationally inexpensive, as it only requires 2 subtractions and 1 addition.

   Mini Manhattan - manhattan distance divided by 4, computationally inexpensive to compute in addition to how Euclidean distance is computed. Minimizes the effect of the h(n) which allows for the result of f(n) to be more focused on g(n).

   Euclidean Distance - admissible heuristic, gives the shortest path between current node and goal. Computationally inexpensive, only requires 2 subtractions, the squaring on those results, and 1 addition. Minimizes the effect of the g(n) which allows for the result of f(n) to be more focused on h(n).

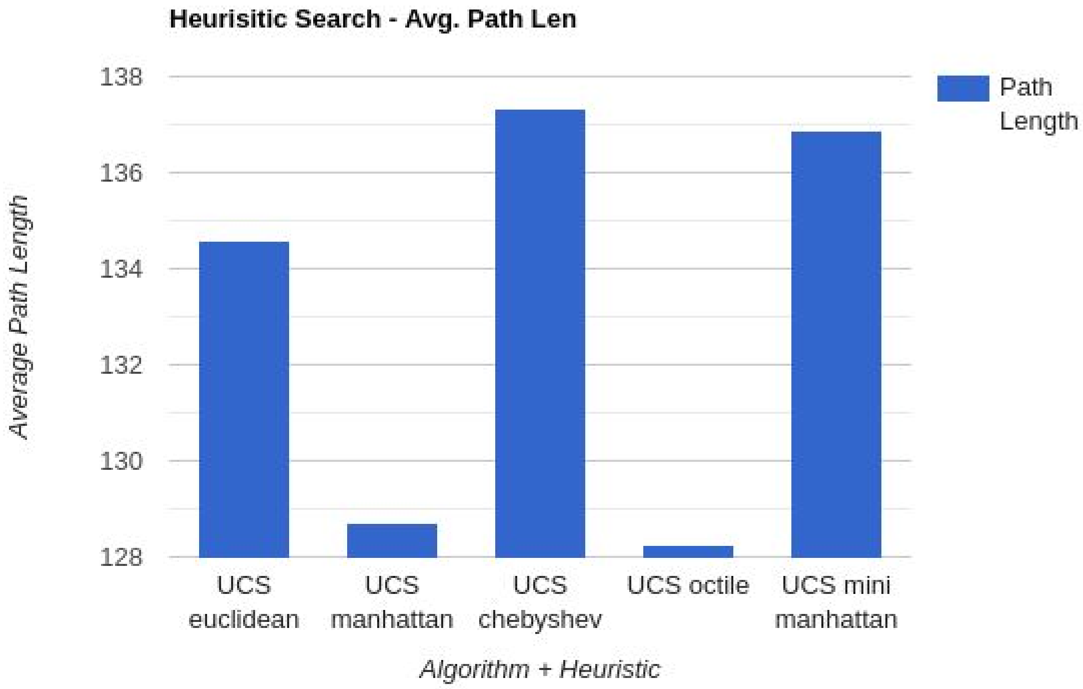
   Euclidean Distance Squared - self explanatory. Computationally inexpensive, it only requires one more square. Good heuristic as it exaggerates the difference between two paths that may be taken from the goal to the destination.
   Mini Euclidean - euclidean distance divided by 4, computationally inexpensive to compute in addition to how Euclidean distance is computed. Minimizes the effect of the h(n) which allows for the result of f(n) to be more focused on g(n).

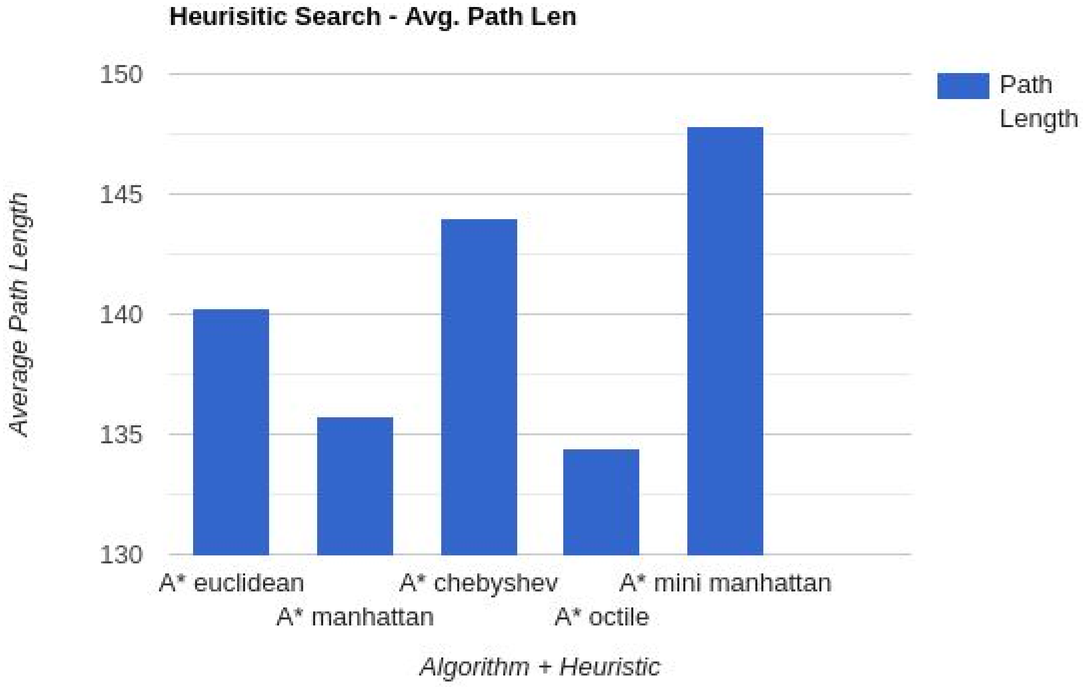
   Chebyshev - allows for all 8 adjacent cells to be reached with a cost of one (manhattan distance does not allow for diagonal moves, only vertical and horizontal, resulting with a diagonal move to cost 2 in Manhattan distance). Computationally, requires 1 multiplication, 5 subtractions, and the calculation of one minimum.

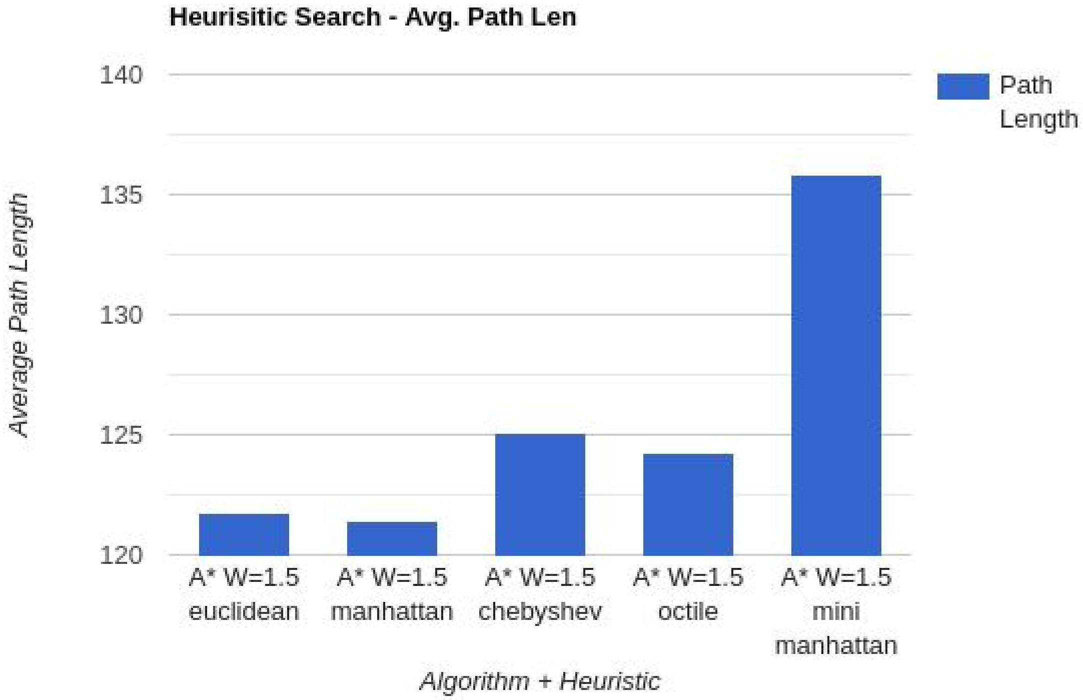
   Octile - Similar to Chebyshev distance, with the modification that diagonal distance will have a cost of sqrt(2) rather than 1. Computationally is very similar to Chebyshev as well.

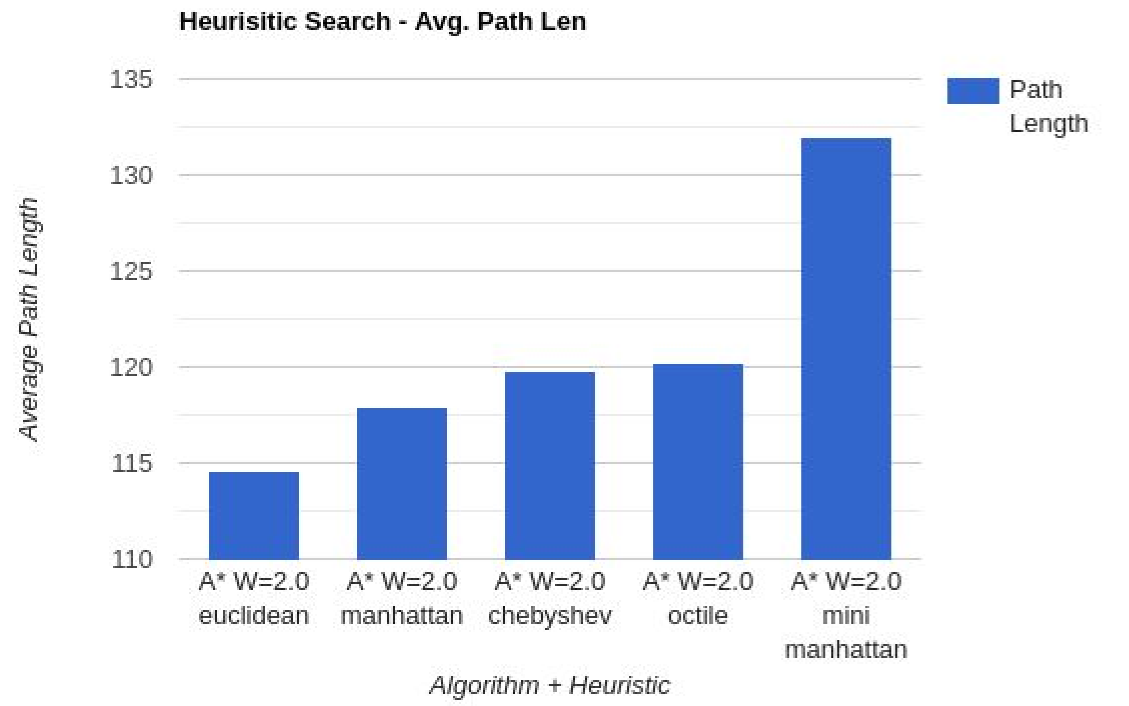
5. Perform an experimental evaluation on the 50 benchmarks using the three algorithms that you have implemented for the 5 different heuristics that you have considered. For Weighted A you * should try at least two w values, e.g., 1.25 and 2 (feel free to experiment). Compare the various solutions in terms of average run-time, average

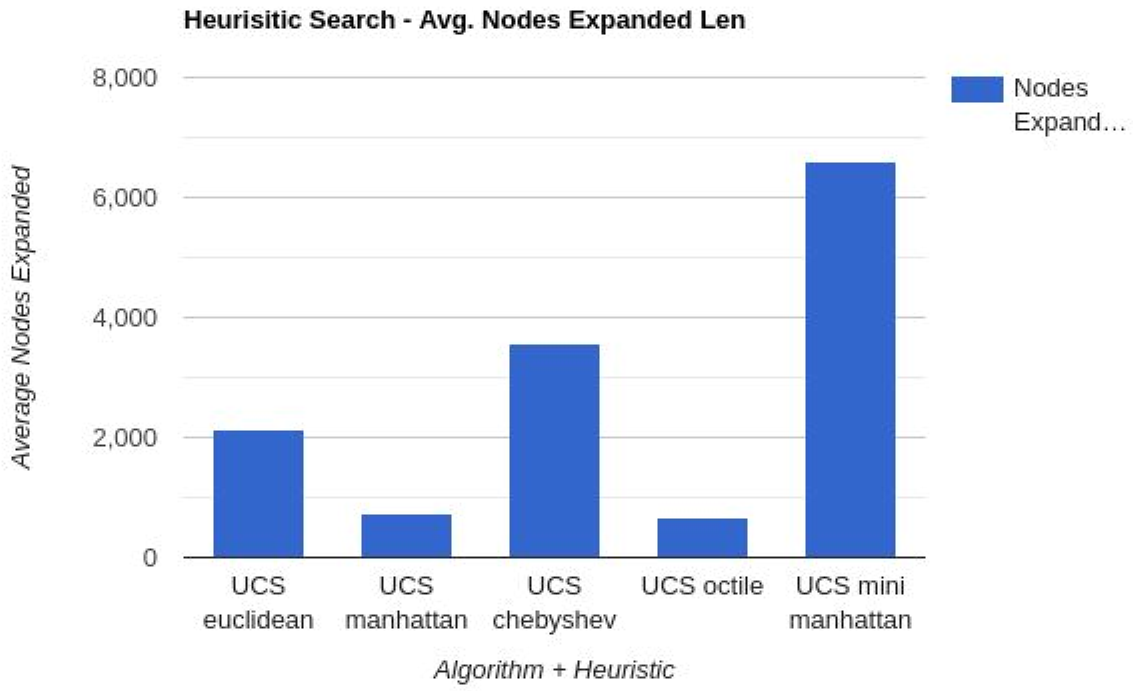
resulting path lengths as a function of the optimum length, average number of nodes expanded and memory requirements (average over all 50 benchmarks). In your report, provide your experimental results. (15 points)

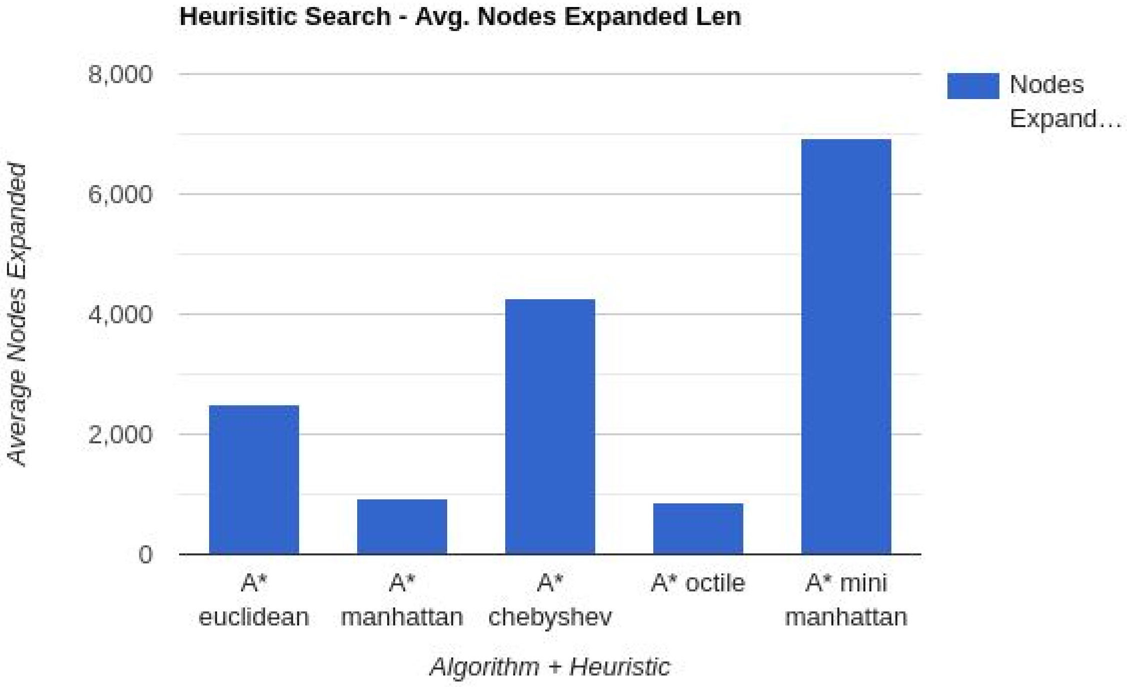


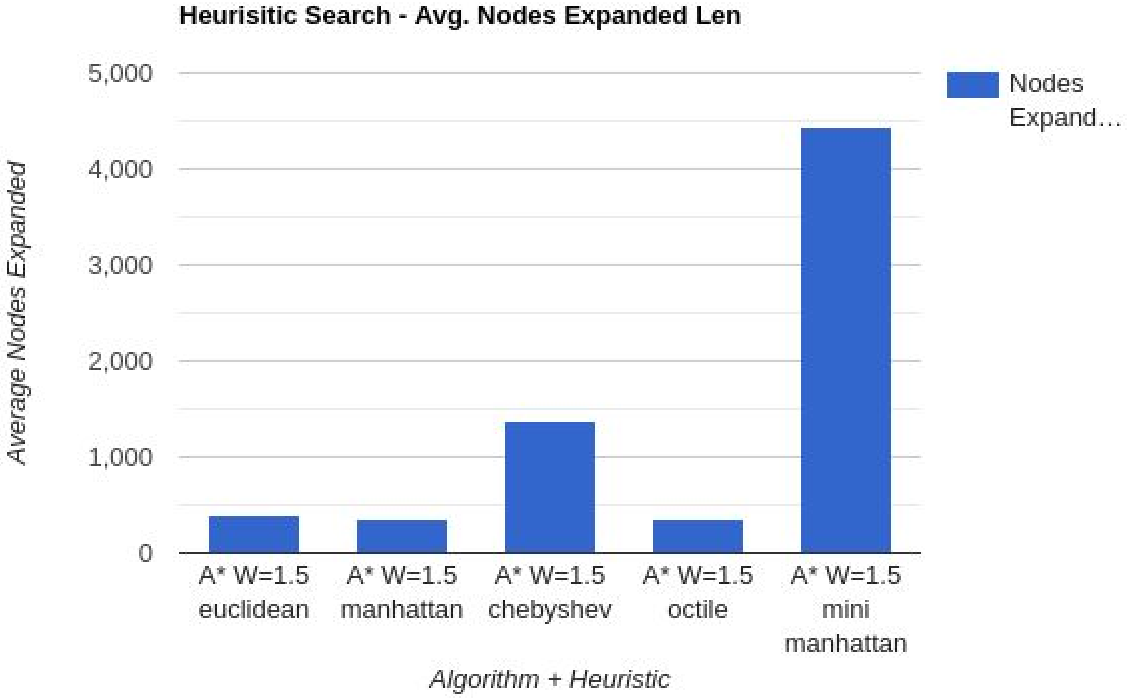

Heurisitic Search - Avg. Path Len
Average Path Length
150
145
140
135
130
A* euclidean
A* manhattan
A* chebyshev
A* octile
A* mini manhattan
Algorithm + Heuristic
Path Length

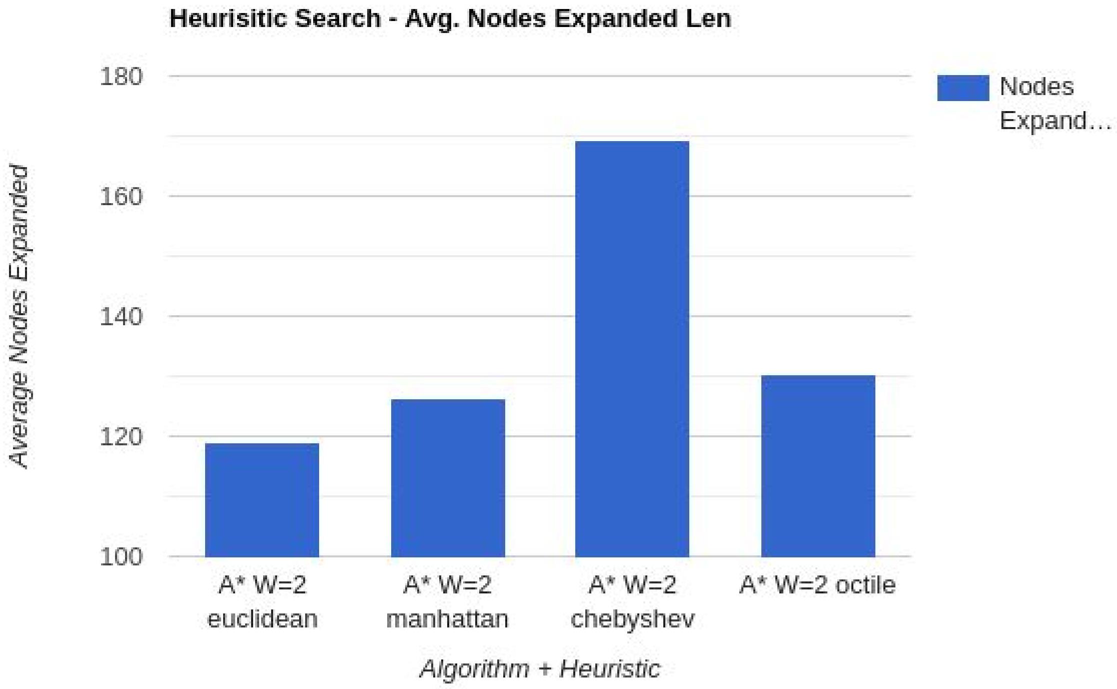

Heurisitic Search - Avg. Path Len
Average Path Length
140
135
130
125
120
A* W=1.5 euclidean
A* W=1.5 manhattan
A* W=1.5 chebyshev
A* W=1.5 octile
A* W=1.5 mini manhattan
Algorithm + Heuristic
Path Length

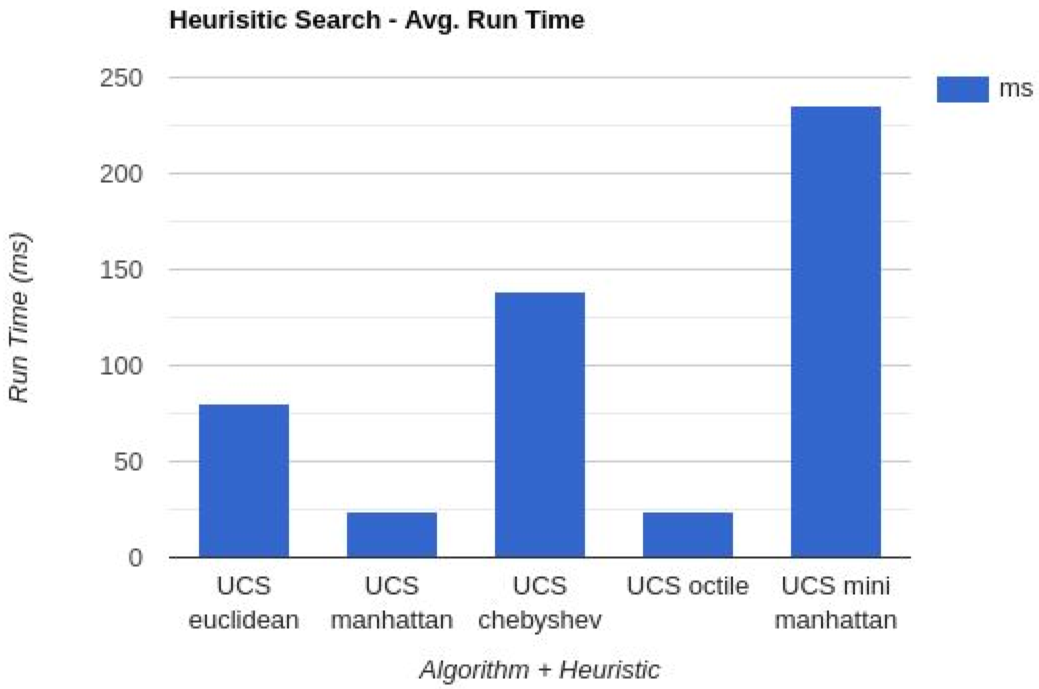

Heurisitic Search - Avg. Path Len
Path Length
Average Path Length
135
130
125
120
115
110
A* W=2.0 euclidean
A* W=2.0 manhattan
A* W=2.0 chebyshev
A* W=2.0 octile
A* W=2.0 mini manhattan
Algorithm + Heuristic

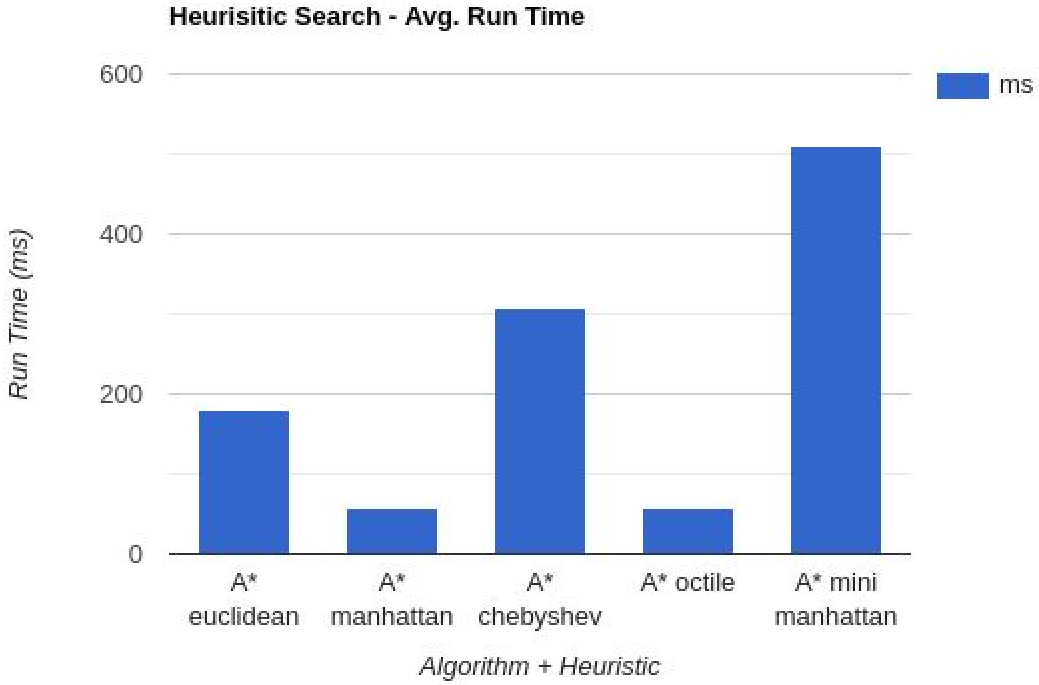

Heurisitic Search - Avg. Nodes Expanded Len
Nodes Expand…
Average Nodes Expanded
8,000
6,000
4,000
2,000
0
UCS euclidean
UCS manhattan
UCS chebyshev
UCS octile
UCS mini manhattan
Algorithm + Heuristic


Heurisitic Search - Avg. Nodes Expanded Len
Average Nodes Expanded
8,000
6,000
4,000
2,000
0
A* euclidean
A* manhattan
A* chebyshev
A* octile
A* mini manhattan
Algorithm + Heuristic
Nodes Expand…

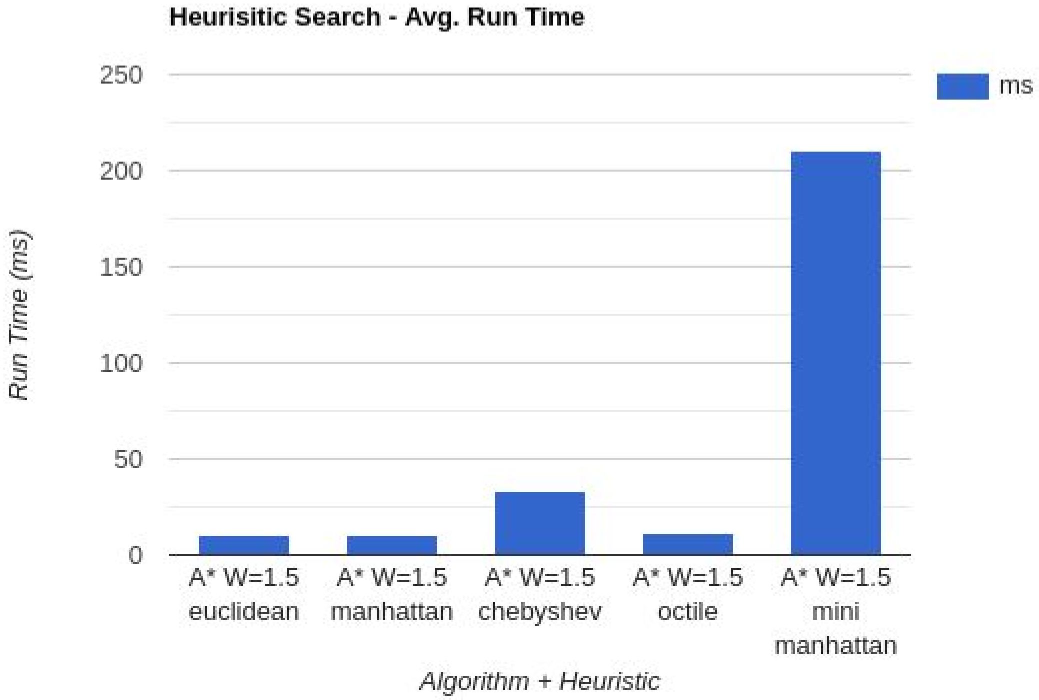

Heurisitic Search - Avg. Nodes Expanded Len
Average Nodes Expanded
5,000
4,000
3,000
2,000
1,000
0
A* W=1.5 euclidean
A* W=1.5 manhattan
A* W=1.5 chebyshev
A* W=1.5 octile
A* W=1.5 mini manhattan
Algorithm + Heuristic
Nodes Expand…


Heurisitic Search - Avg. Nodes Expanded Len
Nodes Expand…
Average Nodes Expanded
180
160
140
120
100
A* W=2 euclidean
A* W=2 manhattan
A* W=2 chebyshev
A* W=2 octile
Algorithm + Heuristic

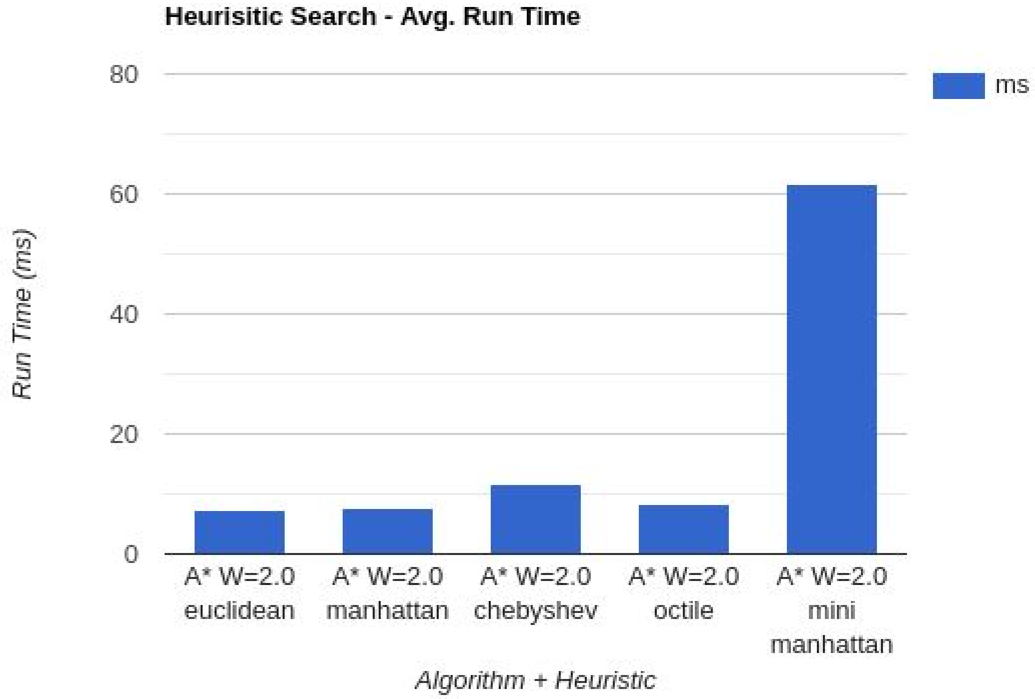

Heurisitic Search - Avg. Run Time
ms
Run Time (ms)
250
200
150
100
50
0
UCS euclidean
UCS manhattan
UCS chebyshev
UCS octile
UCS mini manhattan
Algorithm + Heuristic



Heurisitic Search - Avg. Run Time



Heurisitic Search - Avg. Run Time



6. Explain your results and discuss in detail your observations regarding the relative performance of the different methods. What impact do you perceive that different heuristic functions have on the behavior of the algorithms and why? What is the relative performance of the different algorithms and why? (10 points)
   As visualized with using different heuristics, it will inherently vary the run time, path generation and visited cells. As for run time, it simply comes down to the computations of the heuristic and the A* algorithm. Weighted A* is by far is the fastest algorithm. Coming in second, would be UCS followed by A*. This comes as no surprise since UCS has less computations than A*.
7. In Weighted-A search, the g value of any state s expanded by the algorithm is at most * w1- suboptimal. The same is true for the anchor search of Algorithm 2, i.e., for any state s for which it is true that Key(s, 0) ≤ Key(u, 0) ∀u OPEN0, it holds that g ∈ 0(s) ≤ w1 c (s), where c (s) * * * is the optimum cost to state s.

   Given this property, show that the anchor key of any state s, when it is the minimum anchor key in an iteration of Algorithm 2, is also bounded by w1 times the cost of the optimal path to the goal. In other words, show that for any state s for which it is true that Key(s, 0) ≤ Key(u, 0) ∀u ∈ OPEN0, it holds that Key(s, 0) ≤ w1 g (s * * goal).

   [Hint: Think about a state si in the OPEN0 queue, which is located along a least cost path from sstart to sgoal. Does such a state always exist in the queue? If yes, try to show first that a similar property holds for such a state is given the properties of a Weighted-A anchor search. Consider * then what this implies for the key of the state s that is the minimum.]

The g value used in the A* algorithm for any state, s, to be expanded in achor search is bounded at w1.

Then, prove that when the Sequential Heuristic A terminates in the * i th search, that gi(sgoal) ≤ w1 *w2 c (s * * goal). In other words, prove that the solution cost obtained by the algorithm is bounded by a w1 *w2 sub-optimality factor. [Hint: Consider all cases that the algorithm can terminate] (10 points)

As per the writeup in the assignment, if the algorithm were to terminate at line 32:

g0(sgoal) <= w1*g*(sgoal) ⇒ <= w1*w2*g*(sgoal) ⇒ thus, w2 >= 1

If we were to terminate on line 24:

gi(sgoal) <= w2*OPEN.MinKey() ⇒ <= w1*w2*g*(sgoal)

Thus, the solution cost would be bounded by w1*w2/