CS 440: Artificial Intelligence
Project 1
Heuristic Search using Information from Many Heuristics

Allen Tung
Julian Liu
February 5, 2017

a) 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:

5 different maps as described above

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)

Our visualization was designed in Visual Studio, using the Windows Presentation Foundation (WPF). The backend was written in C#, managing the algorithms and map generation. Specific application behavior is defined in the XAML codebehind files, and data is stored in a viewmodel, somewhat following the Model - View - ViewModel behavior as dictated by Microsoft. Observable data on the front end is represented as properties in the Viewmodel, and are bound together. When the viewmodel is updated, the observable data on the front end updates as well. The map is represented by 120*160 buttons, with different colors and borders to distinguish different kinds of cells. Rivers are denoted by an aqua border, black nodes are impassable, white nodes are freely traversable, and gray nodes are difficult to traverse.

There is only one window for the user interface, consisting of a side panel to display the relevant A* algorithm h, f, and g values, and to give the user the options to select or generate maps, calculate new runtimes and a new path, and change the specifications of each algorithm. Start-goal pairs are also available as a dropdown box for the user to select, upon which, a new path will be calculated. The path will be highlighted, with the start and goal pairs marked on the map.

Each button on the grid will trigger an event to update the sidebar with the specific information pertatining to the path. On the trigger, the button will only pass the coordinate of the tile, which will be used to query the backend for data. The backend will calculate the path, and pass the information back to the front end, which will update accordingly. When a button on the map is pressed, the F, G, and H information in the UI will be loaded and displayed. Upon changing to a different map or startgoal pair, the screen will be reloaded with new information, and the A* path and information will be recomputed.

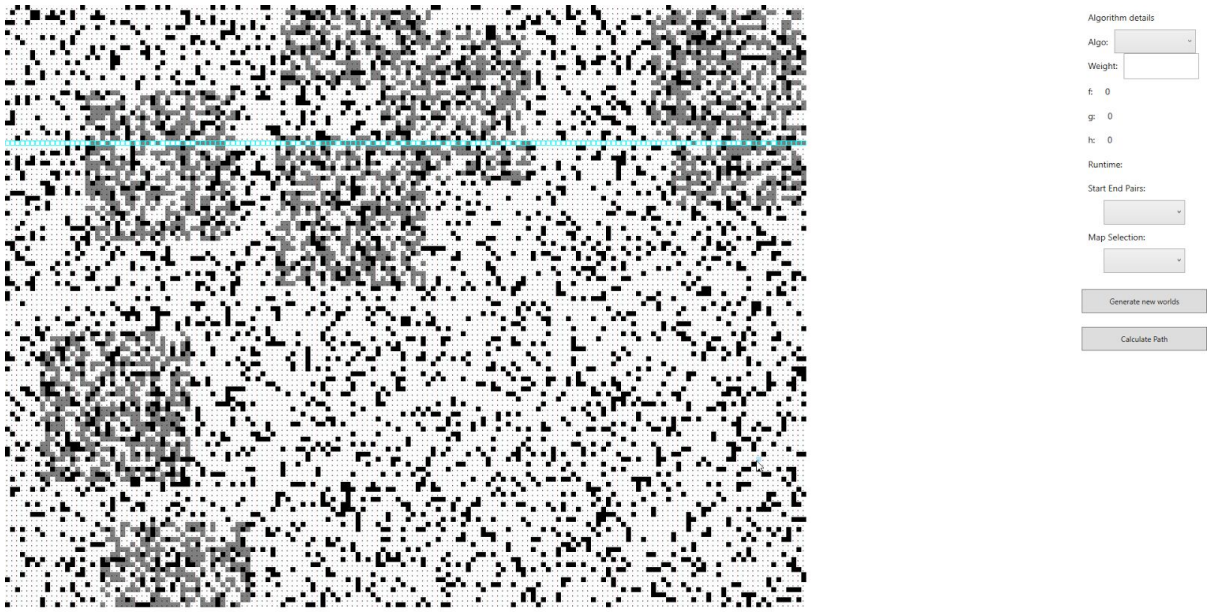

Figure 1. Basic UI and map example.

b) 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). 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. (10 points)

We choose to represent the map as a 2D array of structures, which will each contain an integer to describe the properties of the cell. Since we are aware of the array size, dynamic allocation will not be an issue. Expanding nodes would involve memory accesses of array locations in a block around the desired cell, which could be inefficient in terms of memory accesses, depending on the architecture, which could be a source of optimization in the future. We keep a priority queue for potential nodes to be expanded, and dequeue as necessary. The cost of traversal for each path is stored in a dictionary, where we list the nodes we traverse, and the values of f, g, and h for each.

For the heuristic, we choose to use the euclidian distance from start to goal, while assuming that every node that is traversed is a highway. In this way, the heuristic is guaranteed to be admissable.

c) Optimize your implementation of the above algorithms. Discuss your optimizations in your report. (5 points)

By using a dictionary to store the fringe and closed set for our A* implementation, we give up space efficiency in favor of time efficiency. Each lookup is only one step away, as opposed to a linked list, or something similar.

d) Propose different types of heuristics for the considered problem and discuss them in your report. In particular:

- Propose the best admissible/consistent heuristic that you can come up with in this grid world.
- Propose at least four other heuristics, which can be inadmissible 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)

We propose first Euclidian distance, the straight line distance from the current point to the goal, as the most naiive heuristic. The shortest possible path will always be greater to or equal to the straight line distance, and thus, is generally a consistent heuristic. However, there is always a computationally intensive square root necessary to compute the distance, and in a grid world where only 8 directions of movement are possible, there must be a better choice than Euclidian distance. Further, we do not consider highways or impassable terrain or difficult to traverse terrain, all of which would affect the real cost of travel.

We propose next a heuristic that attempts to remove the computational burden of euclidian distance, using the ‘squared distance’, such that we do not need to perform a square root. However, we expect this to perform very poorly, being that the heuristic will be in a different scale than the traversed cost, which it will be added to. Cutting corners for performance should end up worsening the performance.

Discarding the idea of euclidian distances, we attempt to use the Manhattan distance, the distance between any point and the goal while travelling only in right angled axes. This heuristic is also proven to be admissible on 4-way connective grids, however, we must consider that we have 8-way connectivity. While this should prove to be a better heuristic than euclidian, and of course better than squared euclidian, we can do better.

As this is the case, we propose next, the diagonal distance, also known as the Octile distance, in which we consider the diagonal path as well in our heuristic. In this case, we assume the diagonal cost is the square root of 2, as discussed in the assignment itself. This seems to follow the actual graph costs most closely, and thus we choose this to be the best admissible heuristic we can come up with. The assumptions made here are that each tile is freely traversable. However, this is potentially not the case- there could even be a highway connecting a start goal node pair directly, in a straight line. In this case, if we are to make this admissible, we must scale the heuristic, and assume that we travel solely on highways when moving in any one of the cardinal directions. This way, there is no possible more efficient route to the goal. As a result, we will preferentially expand nodes in the horizontal and vertical directions, reducing our efficiency, but the resulting heuristic should prove to be admissible.

Another heuristic is similar to the Octile distance, called the Chebyshev distance, which considers all paths as equivalent in cost, whether they are diagonal or not. We do not expect

this to perform as well as Octile distance, being that it is less close to the actual conditions of the map.

e) 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 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. (10 points)

Unfortunately, under the time constraints, experimental evaluation was impossible.

f) 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)

Different heuristic functions affect the speed at which the algorithms approach the answer, and they affect the correctness of the result. For A* searches, a heuristic must be admissible if it is a tree search, and consistent if it is a graph search. An admissible heuristic never overestimates the cost to reach the goal. If the heuristic is consistent, it is also admissible; also, the heuristic must not overestimate the distance from a neighbor to the goal summed with the cost to reach the neighbor. If A* is given a consistent heuristic, it will find the optimally efficient path to the goal, revisiting no nodes, and expanding the least necessary nodes