CAP 5625: Introduction to Artificial Intelligence
Project-1 Part-2

Ashutosh Modi (modia), MSCS
Palak Dave (palakdave), MSCS
Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620
Email: modia@mail.usf.edu; palakdave@mail.usf.edu

1. Introduction

This report summarizes technical project to find optimal path between any two cities of available network using the A* (pronounced as “A star”) search algorithm. A* algorithm is widely used for pathfinding and graph traversal. The implementation (referred as backend here after) is further integrated with GUI to facilitate friendly and easy to use interface to user. Complete work is accomplished using MATLAB R2016a.

Section 2 of this report describes the A* algorithm, related terminologies and briefing about application specific implementation.

Section 3 describes application design and functions involved.

Section 4 describes the user interface and user inputs.

Section 5 compares performance of algorithm implementation using different heuristics.

Section 6 is about the efforts involved in the development.

2. A*

This section explains how A star algorithm works and its implementation using two heuristics

2.1. Algorithm

As mentioned in introduction, A star is widely used search algorithm for optimal pathfinding and graph traversal. Algorithm uses knowledge, known as heuristic, to avoid exhaustive search, as done in BFS and DFS. It is a variant of Dijkstra’s algorithm. While searching, the next node to explore is chosen based on its cost value. This cost is sum of cost incurred $G(n)$ to reach that node and minimum expected cost $H(n)$, heuristic, to reach the destination from that node. The node with minimum cost value is chosen from available open nodes. The performance of the algorithm depends on the goodness of heuristic. Heuristic must be chosen smartly such that it is always maximum underestimate i.e. minimum value.

2.2. Implementation

Complete project is implemented using MATLAB. The pseudo code is shown in Algorithm 1 on the following page Two different heuristics are implemented as described below. User can chose either of them from the UI.

- Straight Line Dist: If given node is not the destination node, Euclidean distance between given node and destination node is found based on their locations. As the minimum distance between any two points is Euclidean distance, this heuristic is an underestimate. (We are not considering manifolds here.)
- Fewest Links: If given node is not the destination node, at least one more link is to be traversed to reach the destination. So, the heuristic value is considered as 1 for all nodes other than destination node.

3. Application design

Figure 1 on the next page shows design overview and following are the code files involved in the A star application, with brief details.

- AStar.m: Contains mainly A* algorithm implementation.
- AStarApplication.fig: Is the GUI layout associated with AStarApplication.m

Algorithm 1 A* Pseudo code

```
– Open and closed list initialization
– Insert start node in open list
– Initialize f(n) and g(n) to max value (infinity)
– F(n) for start node is zero
– While open list is not empty
        Find node from open list with min f(n)
        If it is a destination return the traced path
        Else remove it from open list and add to closed list
        For each connections of this node
                Connection in closed list then continue
                Connection not in open list then add it
                Calculate g(n)=f(n) + h(n)
                If it is lower than what we had
                        Update parent mapping table
                else
                        continue
        End
End
```

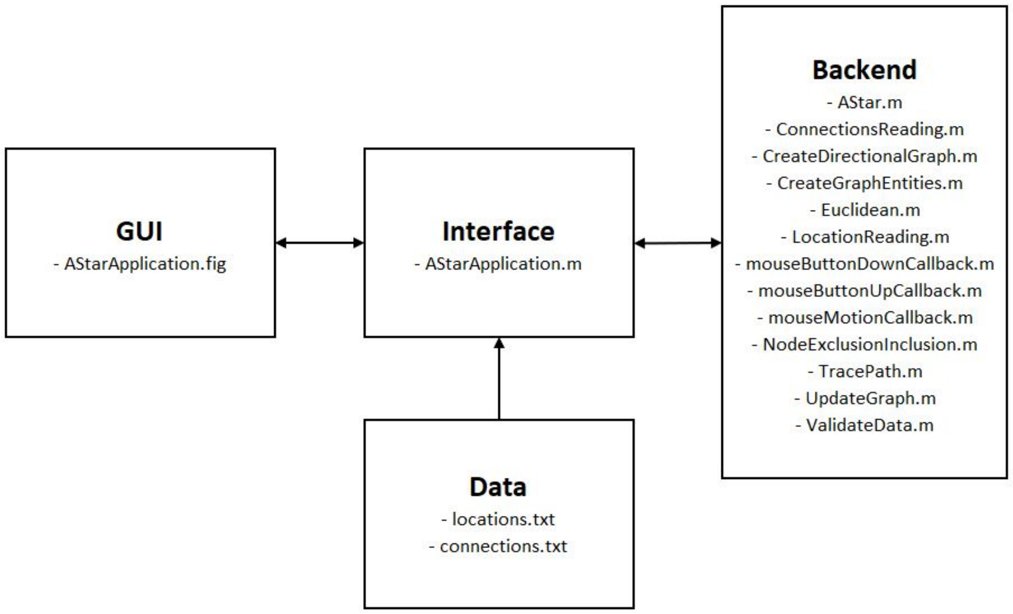


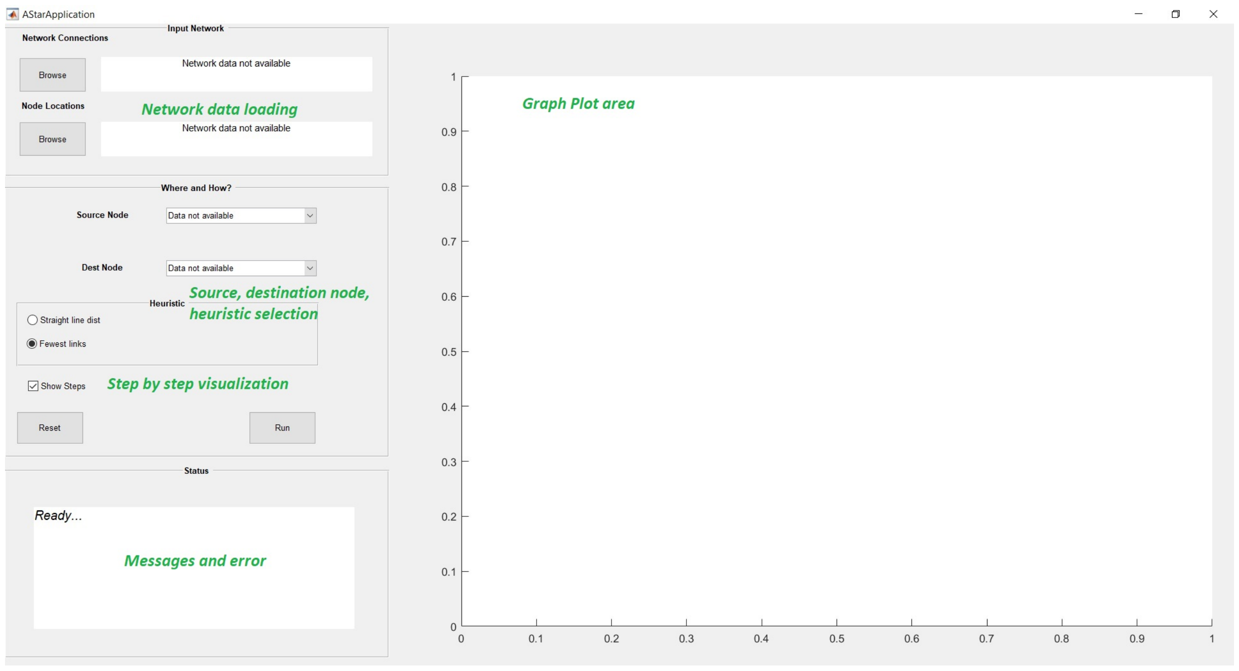
Figure 1. System design

- AStarApplication.m: Is the GUI handler and acts as an interface to backend application
- connections.txt: Sample connection file
- ConnectionsReading.m: Reads connections text file and stores information in structure.
- CreateDirectionalGraph.m: Creates directional graph and plots it on axes plot
- CreateGraphEntities.m: Creates a data structure which holds information about the network
- Euclidean.m: Straight distance heuristic
- LocationReading.m: Reads locations text file and stores information in structure.
- locations.txt: Sample locations file
- mouseButtonDownCallback.m: Mouse event handler for button press event on graph
- mouseButtonUpCallback.m: Mouse event handler for button release event on graph
- mouseMotionCallback.m: Mouse event handler for mouse movement event on graph

- NodeExclusionInclusion.m: Handles inclusion and exclusion of nodes
- TracePath.m: Gives final optimal path by using node-parent mapping structure
- UpdateGraph.m: This creates directional graph, highlights excluded cities and sets event handlers
- ValidateData.m: To validate if start and destination nodes are valid.

4. GUI

Steps to run the application and getting the path for different heuristics are briefed in the readme. This section shows actions done on GUI at each execution step.



Heuristic: Straight line dist [✓ Fewest links]
☑ Show Steps

Figure 2. Application GUI

4.1. Steps to execute

Figure 2 shows the A* application user interface. Basic steps to get the path for desired network is as follows:
- Load connections data file using first browse button from “Input Network” panel
- Load locations data file using second browse button from “Input Network” panel
- Select source and destination nodes from the drop-down menu from “Where and How” panel
- Select the desired heuristic using radio buttons in heuristic panel
- Enable check box if step by step execution of the algorithm needs to be visualized
- Click run button to trace path
- Reset button can be used to traced path before running algorithm for new source or destination node

4.2. Graph description

Figure 3 on the next page shows how graph plot looks once path is traversed step by step. Magenta colored lines are the path traversed while searching for destination node. Green path is the optimal path found for chosen heuristic. Source node is colored green whereas destination node is colored red. Excluded nodes are in black star shape and black dotted links represents its connections.

As shown in figure 3 on the following page A1 is source node, D3 is destination node and D2 is excluded node.

On top of the graph cursor coordinates from graph plot are displayed. This will help in moving the node t new position.

Every connection, displayed in blue line, shows Euclidean distance between connection. As there can be one way path, each connection has arrow showing direction.

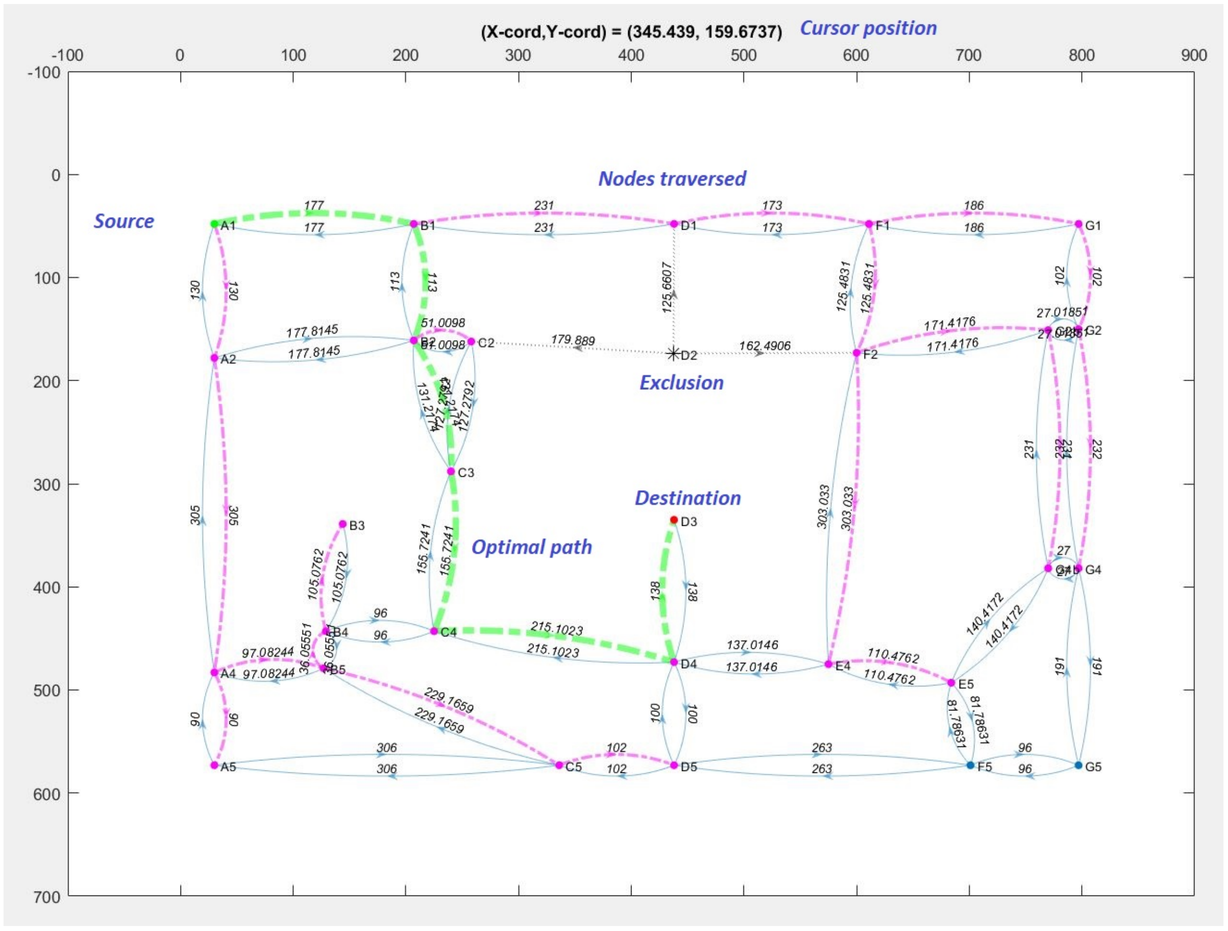


Figure 3. Traversed Path

4.3. Moving a node

Any node on the graph plot can be moved to another location using mouse click and drag operation. Using left click button select the interested node, drag it to new desired location and release the button. This will update the new location of the node in network data in backend. After clicking "Run" button will give optimal path as per new coordinates. Multiple nodes can be moved to new locations and there is no restriction on how many nodes and where on the axes plot.

4.4. Node exclusion/inclusion

Any node on the graph plot can be excluded in A* algorithm calculations by clicking it with mouse right button. This will turn the node in black star shape and removes paths to connected nodes. But it shows the connections in black dotted lines.

Clicking the excluded node with mouse right button will include it again in network.

4.5. Reset

Reset button will reset the plot to original state. By clicking reset button will delete all user configuration of moved nodes and excluded nodes.

5. Performance and result

To check for the performance of the algorithm, network from the sample files is used. Time taken to find the path for different heuristic can be compared. Table 1 on the next page shows time in seconds to get path from source node to destination for straight line distance and fewest links. Network used is same as shown in figure 3.

Heuristic	Source-Destination			
	A1-D3	A1-G5	C4-G1	C4-D3
Straight Line Dist	0.685	0.842	0.724	0.167
Fewest Links	0.653	0.751	0.598	0.218

TABLE 1. COMPARISON RESULTS

Tasks	Hrs
Literature survey	4
Backend development	16
GUI development and integration	20
Testing and results	2
Documentation	2

TABLE 2. TIME EFFORTS

It can be observed that there is no good or bad heuristic. Even though both heuristics use underestimate value to start with, performance depends on the layout and start-destination node location. With these few tests and for small network it is difficult to compare the heuristics and to find a good measure.[1]

6. Tasks involved

Following are the main tasks associated with completion of this project. (A - Ashutosh, P - Palak)
- Environment and language (A,P)
- A star implementation (A,P)
- GUI layout (P)
- GUI backend interface (A,P)
- Event handling (A)
- Performance and Result (A,P)
- Documentation (A,P)
- Functional testing (A)

Time efforts spent on the project can be classified as shown in table 2.

1. Matlab tic-toc functionality has been used to get execution time of function AStar().