CS638 Machine Intelligence
Assignment 2
Route Finding using the A* Algorithm

Dilip Antony Joseph (CS00294)
Rohit Ramesh Saboo (CS00327)

December 10, 2002

1 Problem Definition

Finding the shortest route between two places is a problem of great practical utility. Searching for the path to be taken can be speeded up by the use of heuristics. This assignment deals with implementing the A* algorithm and using it to find the shortest routes between two places in a city. A graphical frontend helps the user define the city map. The various steps of the algorithm are also animated through the GUI.

2 A* Algorithm

2.1 Description

The A* algorithm is used to find the optimal path from a starting node to a goal node. The algorithm maintains two lists - *Open List* and *Closed List*. The Open list consists of all nodes seen by the algorithm, but not yet expanded. All the nodes which have been expanded already are put in the Closed List. Starting from the start node, A* chooses the cheapest node in the Open list to be expanded next. The cost of a node is determined on the basis of its *f value*. The *f value* of a node is defined as follows:

$$f = g + h \text{ , where}$$

g is the length of the path from the start to the node
h is the heuristic estimate of the distance of the node to a goal node

2.2 Algorithm Steps

Initialize OPEN to contain the Start Node.
Initialize the Start Node : g = 0, h = heuristic(Start Node), f = h.
Set CLOSED to the empty list
Until a goal node is found, repeat the following steps
If OPEN is empty report FAILURE
else
Pick the node from OPEN with the lowest value of f. Let this node be BESTNODE
BESTNODE is removed from OPEN and added to CLOSED
If BESTNODE is a goal node Report the solution found and exit
else
Generate the successors of BESTNODE
For each successor SUCC of BESTNODE do the following steps
Set BESTNODE as the parent of SUCC
Compute g : g(SUCC) = g(BESTNODE) + cost of link
If SUCC is already in the OPEN list do
Let the node corresponding to SUCC in the
OPEN list be called OLDNODE.Add OLDNODE
to the list of BESTNODE's successors
If it is cheaper to reach OLDNODE through
BESTNODE than through the earlier path
i.e. if (g(SUCC) ¡ g(OLDNODE)) then
Update the g value of OLDNODE :
g(OLDNODE) = g(SUCC), Recalculate the f value
Set BESTNODE as the parent of OLDNODE
else if SUCC is already in the CLOSED list do
Let the node corresponding to SUCC in the CLOSED
list be called OLDNODE. Add OLDNODE to the list
of BESTNODE's successors
If it is cheaper to reach OLDNODE through
BESTNODE than through the earlier path
i.e. if (g(SUCC) ¡ g(OLDNODE)) then

Update the g value of OLDNODE :
g(OLDNODE) = g(SUCC)
Recalculate the f value
Set BESTNODE as the parent of OLDNODE
Propogate the improved f value to the
successors of OLDNODE. This is done by
doing a Depth First Traversal from OLD NODE
else if SUCC is not in OPEN list nor in the CLOSED list then
Add SUCC to the list of BESTNODE's successors
Add SUCC to the OPEN list
Calculate f value of SUCC

3 Heuristic Used

The A* search is guided by a heuristic function. The straight line distance between the current node and goal node was used as the heuristic estimate for the current node. This heuristic never overestimates the distance to the goal. Therefore it is an admissible heursitic and thus the A* algorithm guarantees an optimal solution. It was also observed that the heuristic behaved like a monotonic heuristic - propogation into the CLOSED list never occurred during the A* search.

4 Features

The following are the features of CityConstructor:

- Contruction of maps
- Support for curved roads and dead ends
- Animation of the A* algorithm
- Two modes of animation - play mode and step by step
- Status bar messages indicating the progress of the Algorithm
- Nodes in open list are coloured in various shades of the same colour, with the cheapest having the brightest colour

5 Implementation Details

This assignment was implemented in Java SDK ver 1.40. The Model-View-Controller Pattern was followed in the design. The MVC paradigm calls for separation of the model (the actual city map), the view (how the map is displayed) and the controller(handling the user input to insert nodes, start search etc.)

The following is a brief description of the important classes used in the program :

5.1 CityConstructor.class

The CityConstructor is the starting point of the application. It loads the various components and provides for the menus and tool bars.

5.2 MapModel.class

The MapModel stores the complete data about the map. It contains a list of all the nodes and links. It provides functions for saving and loading maps from the disk. Other functions provided by this class include - retrieval of a node based on its coordinates, adding nodes/links , removing nodes/links. The MapModel collects all the information about the city in a single place, thus providing a uniform and consistent interface to the other classes.

5.3 MapView.class

The MapView class is used for displaying the map. It uses the MapModel to retrieve information about the nodes and links. The sole job of this class is to paint the map on screen.

5.4 MapEditor.class

The MapEditor is the controller class for the CityConstructor. It responds to all user events and launches the necessary actions.

5.5 Node.class

The Node class encapsulates a city or place on the map. It contains all details about the city including its name, location coordinates and the links to other Nodes. It maintains status information for use by the AStar algorithm - for example, whether the city is currently in the OPEN or CLOSED lists. The MapView colours the Node according to its current status. Nodes in the OPEN list are coloured in shades of yellow, with the cheapest node having the brightest colour. The CLOSED nodes are depicted in RED, while the unexplored nodes are white.

5.6 Link.class

Nodes are connected to other Nodes through the Link class. A Link connects two nodes, one at each end point. The link may be a straight line or it may be curved. Each link is associated with a link cost, which is the actual length of the link. This link cost is used by the AStar algorithm while calculating the g values. The MapView colours a Link according to its current status - unexpanded, expanded, parent link, path to goal etc.

5.7 AStar.class

The AStar class implements the A* algorithm. A new instance of this class is instantiated for each search. It is passed the start and goal nodes by the controller class. This class displays the progress of the search using different colours. The animation can be automatic or manual (on mouse clicks). The class maintains the Open and Closed lists of the A* algorithm. The nodes in the open list are coloured in shades of yellow , with the cheapest node having the brightest colour. A node in the closed list is coloured red.

6 ScreenShot

Figure 1 shows a screenshot of the City Constructor.

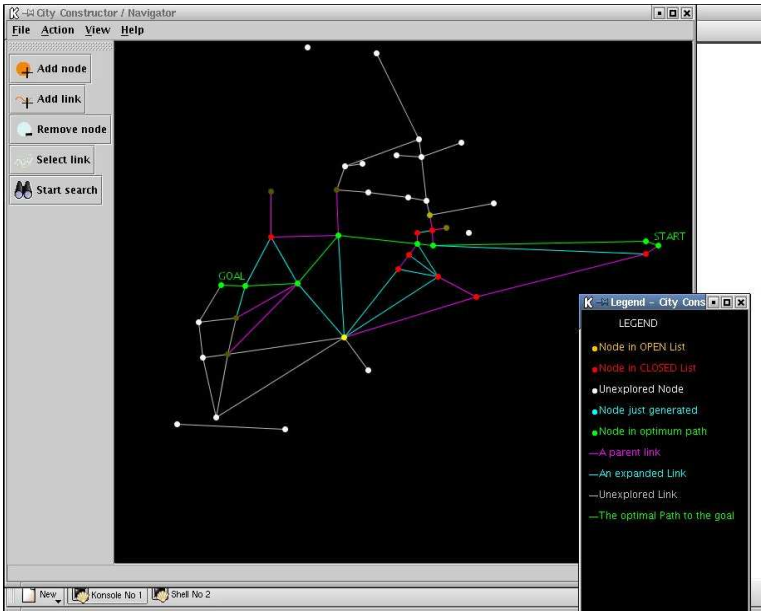

Figure 1: Screenshot of CityConstructor - A Search has been completed. The Legend can also be seen.

7 Conclusion

A program for finding the shortest routes in a city using the A* algorithm was implemented. The guiding power of a heuristic in search was observed when the search concentrated only in the "right" directions. Maps of various localities may be created using the CityConstructor. This program can be useful as an applet in the web pages of organizations with large campuses like IIT and can help visitorsin finding the shortest routes to their destinations!