

Paper Reference No: MWF PN 55

TITLE: AUTOMATED TRANSMISSION LINE ALIGNMENT SYSTEM (ATLAS) USING VISUAL BASIC AND ARCOBJECTS

Name of the Presenter: R. Monoj Kumar
Student, CEG, Anna University
No.52/55, Mosque Street,
Saidapet, Chennai 600015
monojkumar2001@yahoo.com
044-23719815, 9940305819

Other Project Members: P S Anandan, Rohan Khanna, G Venkatesh

Project Guide: Mr. L. Subbaraj, Scientist, IRS, Anna University

INTRODUCTION

Automation means the act of implementing the control of equipment with advanced technology, usually involving electronic hardware. The conventional methods of alignment involve considerable amount of field work, beginning from reconnaissance survey till the final route surveying is performed. By automating the process, initial steps of reconnaissance can be performed sitting in the comfort of your room.

Electric power transmission is one process in the delivery of electricity to consumers. The term refers to the bulk transfer of electrical power from place to place. Typically, power transmission is between the power plant and a substation near a populated area. This is distinct from electricity distribution, which is concerned with the delivery from the substation to the consumers. Due to the large amount of power involved, transmission normally takes place at high voltage. Electricity is usually sent over long distance through overhead power transmission lines. A power transmission system is sometimes referred to colloquially as a "grid".

Alignment is the process of generating a route between two points for the purpose of transferring Water(Canal alignment), Power (Transmission line alignment) and Vehicle (Highway alignment and Railway alignment). The alignment should be of least cost and must cause minimum inconvenience to people and surroundings.

ATLAS (Automated Transmission Line Alignment System) is an automated solution to the problem of transmission line alignment.

OBJECTIVE

The objectives of this project are:

- To generate an alignment between the source and the destination points based on input layers considering weightages.
- To make a comparative study of the routing algorithms used.

CONCEPTS INVOLVED

- Accumulated Surfaces
- Friction Surfaces
- Dijkstra Algorithm
- A* Algorithm
- Heuristics
- Patterns and Direction

ACCUMULATED SURFACES

Accumulated surfaces are raster models that allow you to calculate a least cost path from point “a” (start) to point “b” (target). The term “accumulated” implies that there is a building up of numbers or values and this is exactly what happens. The cost for a cell in an accumulated surface is a value that represents a cumulative cost from the target. So, to travel across four cells in a row with the first two cells each having a value of one and the second two cells each having a value of two, the accumulated cost to travel through the four cells is six. This value is stored in the start cell.

FRICITION SURFACES

Friction surfaces contain no cumulative values, but contain costs from which the accumulated surface is created. Friction surfaces are a significant factor in developing more realistic accumulated surfaces. Friction Surfaces are matrices that add resistance to travel within the matrix. As a few sources put it, friction is:

“Cost of travel (functional distance) ...” (DeMers, 2002)

DIJKSTRA ALGORITHM

The Dijkstra algorithm requires two or more nodes and searches possible routes for the shortest result based on specific weights of the edges that connect them. All nodes are connected to each other by these edges. The edges contain weight which could be cost, distance, or time. In a network of nodes the shortest path is the result of the lowest sum of weights between two nodes (Street, 2004)

Dijkstra's algorithm works by visiting vertices in the graph starting with the object's starting point. It then repeatedly examines the closest not-yet-examined vertex, adding its vertices to the set of vertices to be examined. It expands outwards from the starting point until it reaches the goal. Dijkstra's algorithm is guaranteed to find a shortest path from the starting point to the goal, as long as none of the edges have a negative cost.

A* ALGORITHM

The A* algorithm is a combination of the Dijkstra algorithm and heuristics. Furthermore, A* will guarantee the shortest path. It is considered an admissible heuristic algorithm because it never over estimates the distance or cost from a cell to the end point. A* begins by searching adjacent cells from the starting point. All of the directly adjacent cells and the starting point are put in the open list and the starting point is considered to be the parent of the others. The open list is just a list of cells that need to be considered for the least-cost path. Cells that are barriers are ignored and kept out of this list. Since the starting point is the parent it can be dropped from the list and sent to a closed list which verifies the first cell in the least cost path. Next, one of the adjacent cells must be added to the closed list. To do this a simple equation is used to find “g” the cost for the cell represents the cell width of 100 feet. The diagonal width is 141 feet and will be used for diagonal movement because it is calculated as the square root of two (1.41) times the horizontal or vertical width.

$$F=g + h$$

The other calculation needed is “h” the heuristic estimate. This can be completed in a variety of ways like Manhattan, Diagonal and Euclidean. Implement these equations to calculate costs for the adjacent cells the lowest “F” value is the next cell added to the closed list. Now there are two cells adjacent to the current cells that contain the lowest “F” value. It is computationally faster to pick the last one added to the open list (Lester, 2004). No matter which is picked they will both result in paths of similar accumulative cost. It is important to note that barrier cells are totally ignored and are never put into the open or closed list. This process is repeated for each cell added to the closed list until the end point is added to the open list or the open list becomes empty in which case there is no least-cost path. Once the end point is put into the open list it is sent to the closed list and the least-cost path can be traced back to the start point.

HEURISTICS

The heuristic function $h(n)$ tells A* an *estimate* of the minimum cost from any vertex n to the goal.

Manhattan distance: $h(n) = D * (\text{abs}(n.x-\text{goal}.x) + \text{abs}(n.y-\text{goal}.y))$

Diagonal distance: $h(n) = D * \max(\text{abs}(n.x-\text{goal}.x), \text{abs}(n.y-\text{goal}.y))$

Euclidean distance: $h(n) = D * \text{sqrt}((n.x-\text{goal}.x)^2 + (n.y-\text{goal}.y)^2)$

A*'s Use of the Heuristic

The heuristic can be used to control A*'s behavior.

- At one extreme, if $h(n)$ is 0, then only $g(n)$ plays a role, and A* turns into Dijkstra's algorithm, which is guaranteed to find a shortest path.
- If $h(n)$ is always lower than (or equal to) the cost of moving from n to the goal, then A* is guaranteed to find a shortest path. The lower $h(n)$ is, the more node A* expands, making it slower.
- If $h(n)$ is exactly equal to the cost of moving from n to the goal, then A* will only follow the best path and never expand anything else, making it very fast. Although you can't make this happen in all cases, you can make it exact in some special cases. It's nice to know that given perfect information, A* will behave perfectly.
- If $h(n)$ is sometimes greater than the cost of moving from n to the goal, then A* is not guaranteed to find a shortest path, but it can run faster.
- At the other extreme, if $h(n)$ is very high relative to $g(n)$, then only $h(n)$ plays a role, and A* turns into BFS.

PATTERNS AND DIRECTIONS

- 4 Directions (Rook)
- 8 Directions (Queen)
- 16 Directions (Knight)
- 32 Directions

METHODOLOGY

- Data Collection
 - Spatial Data
 - Non-Spatial Data
- Why Raster
- Application Development
- Steps in Application

DATA COLLECTION

The data collection includes spatial data for the area and the AHP weightages (both intra and inter weightages) for the various layers.

SPATIAL DATA

The following Layers were obtained, Land use, Geology, Soil, Roads and National Highways, Railways, Towns

NON-SPATIAL DATA

Weightages for the sub – classes within a layer as well as the inter weightages were obtained from the experts by circulating questionnaire. Cost for the various sub – classes of landuse and other layers was also obtained from the field experts.

WHY RASTER

Choosing Raster format was considered to be a better option when compared to Vector Shape files for getting the discrete cost surface and subsequently using it in the routing algorithm. This is because the vector approach falls far short in planning to traverse off-road terrain. Raster data illustrate numerical and geographic patterns best for elevation. In off-road applications, the use of elevation data is necessary and the vector model is not reasonably implemented. Instead using a raster environment for route planning is much more suitable, as it does not restrict the ability to travel over a continuous surface.

APPLICATION DEVELOPMENT

The application is created using Visual Basic and ArcObjects. Visual basic is a language that has an integrated development environment (IDE). The structure of Visual Basic is very simple; particularly as to the executable code. The VB-IDE has been highly optimized to support rapid application development (“RAD”). It is particularly easy to develop graphical user interfaces and to connect them to handler functions provided by the application.

The ArcObjects components collaborate to serve every data management and map presentation function common to most GIS applications. ArcObjects provides an infrastructure for application customization that lets you concentrate on serving the specific needs of your clients. Arc Objects is a collection of Objects, Classes and Interfaces dealing with Maps and Spatial components (ESRI product). Arc Objects Can be integrated in VB for producing light weight Map Applications

The application is stand alone, given the input layers in the raster format, the source and destination points, the necessary algorithm parameters the program will solve the least cost path and show the new alignment.

The following are some of the functions provided in the map frame.

Zoom In	:	To enlarge the map for greater clarity.
Zoom Out	:	To diminish the map to get the larger area of view.
Pan	:	To browse through the map area.
Fit View	:	To make the map fit in the Map frame View.

STEPS IN APPLICATION (TO CREATE A NEW ALIGNMENT)

1. Selection of the input Layers for the discrete cost Layer

The various layers required for the preparation of the discrete cost layer are obtained during runtime from the user

2. AHP weightages for the various layers and intra layer weightage

The AHP weightages for the various layers are obtained using this window. This window makes use of a Flex grid control to get the weightages from the user. Therefore this window has the capability to accept weightages for any number of layers.

3. Grouping of sub – classes within a layer

This window allows the grouping of sub – classes in a layer and the user can group the classes by selecting a name from the combo box. The users can also define their own sub - class by typing the new class name in the combo and then by pressing the enter key.

4. Opening the discrete cost layer and selecting the source and destination points

After the overlay operation, the discrete cost surface is displayed in the map window. The source and the destination points are given as input by the user either by typing the coordinates in the text box or it is obtained by the mouse clicks on the discrete cost layer.

5. Selection of the algorithm and pattern to be used for producing the alignment

After opening the discrete cost layer and providing the source and destination points, the next step is to select the algorithm and the pattern from the dialog and click on the ‘draw route’ button to get the alignment between the source and destination. The user can also view the accumulation surface in case if he wishes to study the accumulation pattern for the study area with the given source and destination points. In case of A* algorithm being selected by the user, he has to provide the percentage weightage given for the heuristics.

6. View Statistics of the route generated

After generating the route the statistics of the route generated can also be viewed. The length of the route, the deviation from bee –line (sinuosity), time taken to produce the route and the time taken to produce the route and the approximate number of towers that will be required for the route are various statistics that can be viewed in the application.

RESULTS AND DISCUSSIONS

- Dijkstra Algorithm Comparison
- A* (star) Algorithm Comparison
- Comparison of A*(Star) and Dijkstra (1st Set)
- Comparison of A*(Star) and Dijkstra (2nd Set)
- Future Orientation

DIJKSTRA ALGORITHM COMPARISON

Algorithm	Time Taken (in sec)	Deviation from the bee-line	Total Route Length (m)	Relative Cost of the Route	Towers Required (app)
D (8)	4.36	1.09	25610	4498	64
D (16)	7.05	1.09	25648	4337	64
D (32)	13.28	1.09	25781	4289	64

Table 6.1 Dijkstra Algorithm Comparison

From the above table it is observed that

- As the number of directions that can be modeled increases the time taken increases as more computation is required for modeling more directions.
- The relative cost of the route decreases as the pattern increases the directions that can be modeled in the algorithm.

A* (STAR) ALGORITHM COMPARISON

Algorithm	Time Taken (in sec)	Deviation from the bee-line	Total Route Length (m)	Relative Cost of the Route	Towers Required (app)
A (80)	11.19	1.12	26410	3988	66
A (70)	34.06	1.12	26340	3713	66
A (60)	67.45	1.19	27960	3895	70
A (50)	122.75	1.22	28660	3967	72

Table 6.2 A* (star) Algorithm Comparison

In the above table it is observed that

- If more weightage is given for the Heuristics then the algorithm completes faster but the relative cost of the route is high (A(90)).
- Consider A (70), it takes more time than A (90) but the relative cost of the route is lesser.

COMPARISON OF A*(STAR) AND DIJKSTRA (1ST SET)

Algorithm	Time Taken (in sec)	Deviation from the bee-line	Total Route Length (m)	Relative Cost of the Route	Towers Required (app)
A (70)	50.45	1.11	26170	2901	65
D (16)	7.09	1.08	25323	3086	63

Table 6.3 Comparison of A*(Star) and Dijkstra (1st Set)**COMPARISON OF A*(STAR) AND DIJKSTRA (2ND SET)**

Algorithm	Time Taken (in sec)	Deviation from the bee-line	Total Route Length (m)	Relative Cost of the Route	Towers Required (app)
A (70)	48.5	1.14	28320	4667	71
D (16)	7.05	1.06	26309	4767	66

Table 6.4 Comparison of A*(Star) and Dijkstra (2nd Set)

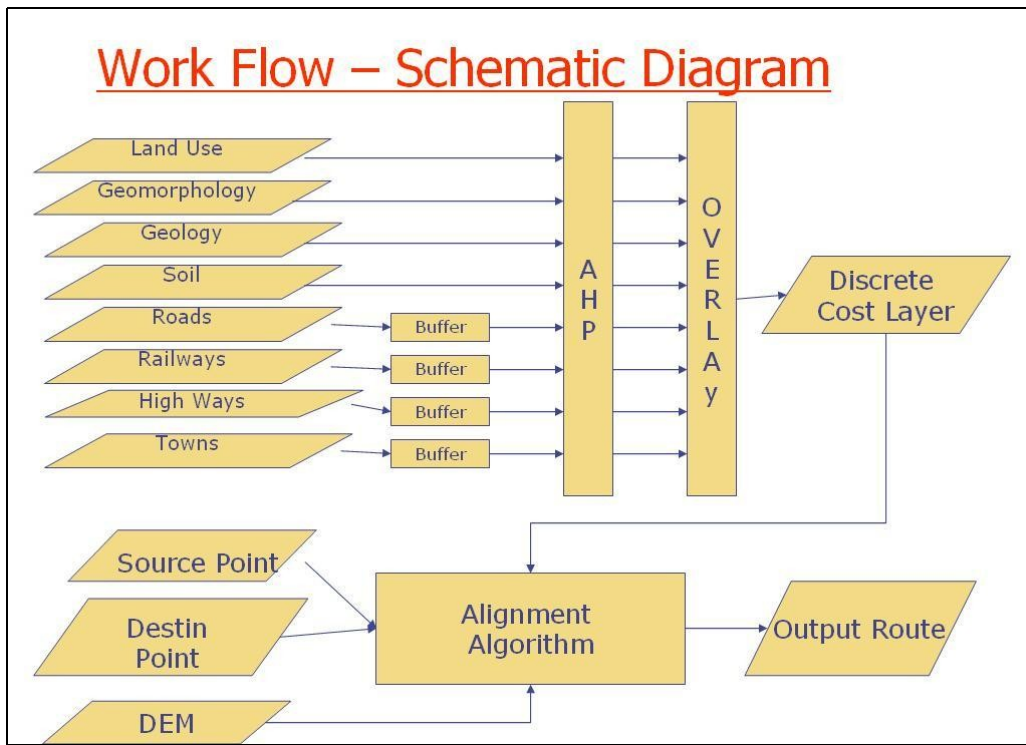
From the above tables it is found that

- A* Algorithm works better for homogeneous areas and takes time to go through heterogeneous areas.
- Dijkstra (16) takes a considerable amount of time but guarantees the least cost path.

FUTURE ORIENTATION

1. The A* algorithm can be extended to 16 or 32 directions.
2. Try out the numerous modified versions of the alignment algorithms. For example we can incorporate Diagonal or Euclidean heuristics in place of Manhattan heuristics in A* algorithm.
3. This application provides only the relative cost of the alignment; suitable modifications can be made to get the actual cost of the alignment.
4. Angle tower determination can be automated.
5. The correlation coefficient between the layers can be used during the overlay operation to reduce the number of layers involved.
6. The entire application can be converted into a DLL and hence it can be incorporated as a module in ArcGIS.

Fig 1:



Workflow of the Application: A Schematic Model

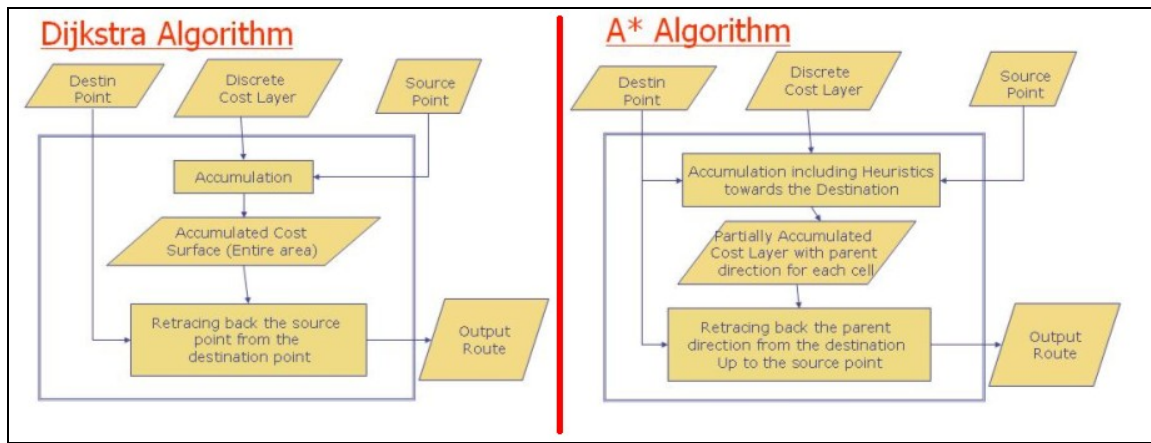


Fig 2: Workflow in the Algorithms

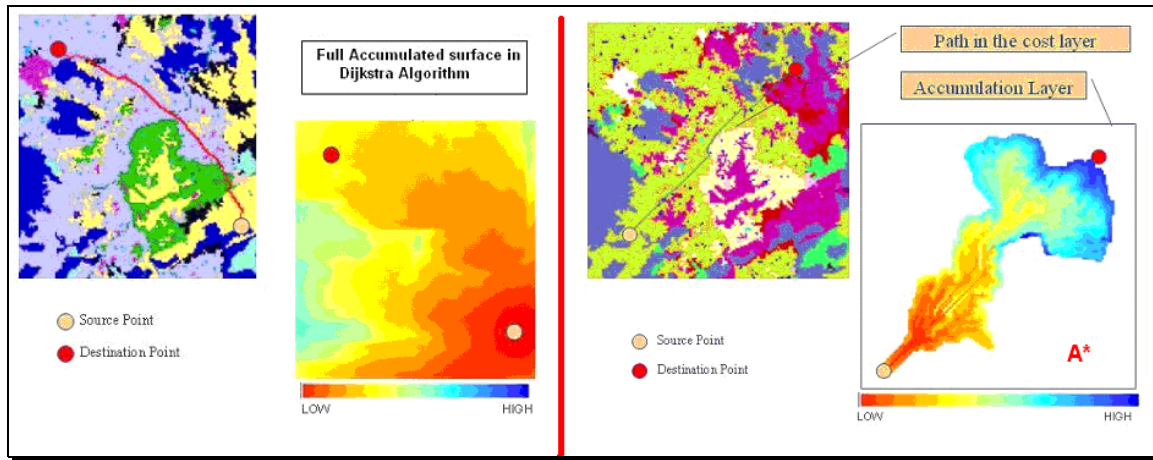


Fig 3: Accumulated Regions

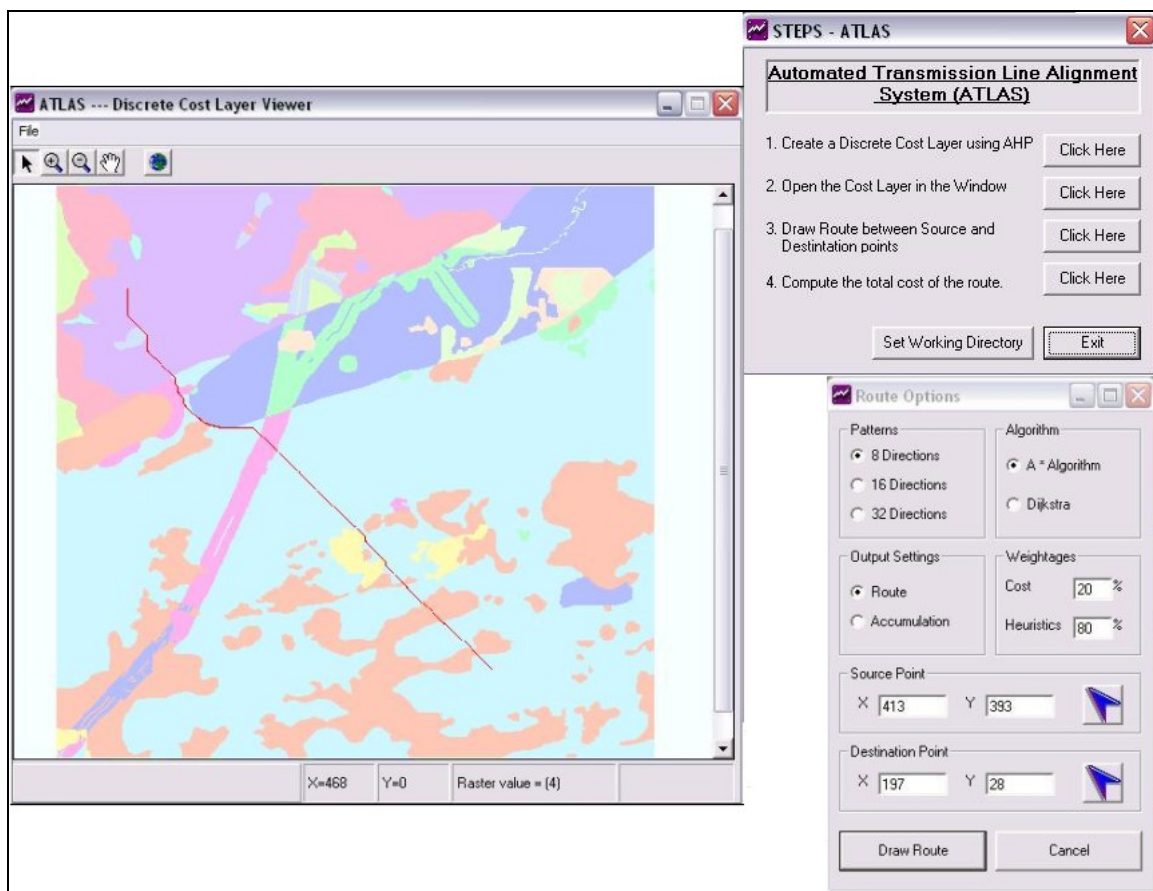


Fig 4: User Interfaces created using VB and ArcObjects