FOREST ROAD LOCATION MODELLING WITH DIJKSTRA'S SHORTEST PATH AND ARCGIS

by

Adam Rickards

FACULTY OF NATURAL RESOURCES MANAGEMENT
LAKEHEAD UNIVERSITY
THUNDER BAY, ONTARIO

April 2019

FOREST ROAD LOCATION MODELLING WITH DIJKSTRA'S SHORTEST PATH AND ARCGIS

by

Adam Rickards

An Undergraduate Thesis Submitted in Partial Fulfillment
of the Requirements for The Degree of
Honours Bachelor of Science in Forestry

Faculty of Natural Resources Management
Lakehead University
23 April 2019

Major Advisor

Second Reader

Abstract

Rickards, A.J. 2019. Forest road location modelling with Dijkstra’s shortest path and ArcGIS.34 pp.

Keywords: forestry, forest, road, network, Dijkstra, slope, terrain, GIS, ArcGIS, modelling, LiDAR, DEM

Road network modelling has great potential to reduce forest road construction and planning costs. Forest roads are the essential means to access forest resources and are one of the most expensive factors in forest operations. Route optimization models have been used recently to aid in pre-planning of forest road networks. Modern technology is increasing the feasibility of these models. LiDAR data is becoming less expensive and more freely available aiding in the accuracy of road network models. LiDAR has the potential to greatly reduce the costs of forest roads through selection of optimal route locations. LiDAR will also reduce the manual planning and design of forest roads further reducing costs. The objective of this study is to model forest road locations. Model parameters calculate distance and slope as the factors in selecting locations. The study tested the same areas with three data types 50 m^2 cell resolution DEM, 5 m^2 cell resolution DEM, and 2 point per metre LiDAR. The test cases in this study found that using LiDAR data gave the best location. Overall there is potential using these data and models to assist in the pre-planning of forest roads.

Contents

List of Figures

List of Tables

Introduction

The development, construction and planning of forest roads are an expensive and necessary means to access forest resources. The optimization of road location selection can assist in reducing the costs associated with forest roads.

The purpose of this study was to select an optimal resolution for data, and to test which data would provide the better route location. Three different scenarios were used to test the model, the first was 50 m^2 cell resolution digital elevation model (DEM), the second 5 m^2 cell resolution DEM, and the third was 2 point per metre light detection and ranging (LiDAR). To generate potential road locations Python was used to code an implementation of Dijkstra's shortest path. Python has a module, ArcPy that works with ArcGIS.

Literature review

Forest roads

Road networks in forestry operations are the essential means to access forest resources (Abdi et al. n.d.). Thousand of kilometres of forest roads are annually constructed globally to access resources (Anderson and Nelson 2004). Forest roads are very costly to build and maintain, and require concise planning (Akay et al. 2013). Manual planning of forest roads is labour and time intensive (Anderson and Nelson 2004). Road network locations are essential in reducing overhead costs in forestry operation (Rönnqvist 2003). Forest road networks need to balance and minimize the amount of roads constructed while maintaining access for resource harvesting, tending, silviculture, and recreation (Liu and Sessions 1993).

With increases in personal computer processing, access to Geographic Information Systems (GIS), and GIS data, pre-planning of forest roads becomes faster and more efficient (Abdi et al. n.d.). ArcGIS is a spatial data platform from ESRI that allows for the creation, sharing, analysis, and management of data (ESRI n.d.). These technological advancements allow for the automation of road network planning (Stückelberger 2007).

Digital elevation models are a specialized database that represent topography through a series of points representing elevations either interpolated from measured elevations in the case of traditional DEM, or measured elevations in the case of LiDAR ("What is a DEM - Digital Elevation Model Definition" n.d.). Data are collected through ground surveys and photogrammetry and a digital elevation model is created. GIS software and tools can interpret these DEM data and provide a means for analysis.

LiDAR is an acronym for light detection and ranging. LiDAR uses laser pulses to measure distances and generate surface characteristics (US Department of Commerce n.d.). Surface characteristics can be ground surface, buildings, vegetation, water including seabed and riverbed elevations. (Akay et al. 2009).

Graph theory and operations research

Graph theory is the study of graphs where a graph (G) is defined as G =(N,E), where N is a finite number of nodes, and E is a finite set of edges (links between pairs of nodes) (Wilson 2010). Weighted graphs are defined as G = (N,E; *w*) where *w* represents an attribute for each which is the cost of that edge (Stückelberger 2007). Graphs can be further divided into different categories, directed and undirected, as well as static and dynamic (Cortes et al. 2003). The previously given definitions is an undirected graph with nodes and edges. In directed graphs nodes are referred to as vertices and edges are referred to as arcs (Wilson 2010). Arcs can only be travelled in one direction, and in order to have bi-directionality between vertices two arcs are required. A dynamic graph is defined by the addition and removal of nodes or edges during operations which can be differentiated from a classic graph where the nodes and edges are static and do not change (Cortes et al. 2003).

Operations research is a field that studies practical applications in operating activities and using the scientific method models decision supports for optimizing these activities (Hillier and Lieberman 2005). The application of graph theory in combination with operations research allows for applications such forest road location modelling (Stückelberger et al. 2007). Operations research can be used to apply a shortest path network routing algorithm to a graph and increase road layout efficiency.

Dijkstra's shortest path algorithm

Dijkstra's shortest path is an algorithm for finding the shortest path between two nodes (Dijkstra 1959). Dijkstra's algorithm finds the shortest path from one node to all other nodes until the end node is found (Stückelberger 2007). The shortest path algorithm is only able to work with positive edge or arc weights. One limitation with Dijkstra's algorithm is that it only considers edge or arc cost and is not capable of measuring multiple attributes simultaneously, thus penalties based on these other attributes need to be applied to the costs (Stückelberger 2007).

Forest Road Modelling

Forest roads require specific constraints to limit where a viable road can be located (Bont et al. 2012). Constraints are created by adding weights to edges or removing edges from areas that are infeasible for road locations (Aruga et al. 2005). Road constraints can be applied to water crossings, steep terrain, infeasible turning radii, geological structures, substrates, and or values within the area (Henningsson et al. 2007). When creating a model to output candidate roads all the constraints must be considered and weighted according to the impact they will have on road location selection (Stückelberger et al. 2006).

Anderson and Nelson (2004) handle the application of all constraints using increasing weights on edges. This method does not omit areas from selection and instead only reduces likelihood of selection. By removing edges from infeasible areas it can fully prevent the model from selecting these routes, though this option should only be used on areas that have no potential for a candidate road (Stückelberger 2007).

Materials and Methods

Data Acquisition

Data were acquired from the province of New Brunswick, Canada online database GeoNB. The data were in the form of digital elevation model (DEM), and LiDAR tiles. The DEM tiles from the New Brunswick database were interpolated using LiDAR from Natural Resources Canada to 1 m^2 cell resolution. The LiDAR data were shot at 2 points per square metre. The area selected was chosen for its dynamic terrain, and availability of a public library of LiDAR tiles. Having a broad range of elevations across the sample area better illustrates the algorithms routing decisions over terrain.

ArcGIS was used to aggregate the original 1 m^2 resolution DEM tiles into 5 m^2 and 50 m^2 resolution tiles. These new tiles covered the same land area at a coarser resolution. The aggregation technique took the mean elevation over the new cell area and applied it to the new larger cell. The LiDAR data was interpolated to a raster of 5 m^2 cell resolution. Cell elevations were determined by the lowest point in that area, this one done to eliminate outliers that may misrepresent the ground elevation. The tile size of 1 km^2 was used to keep processing time to a feasible amount for this study. At the 1k m^2 tile size with 5 m^2 cell resolution 40, 000 nodes are generated.

<u>The Algorithm</u>

The route optimized forest road location problem finds a shortest path from a start node to an end node. The start and end nodes are preselected. To test the implementation of the algorithm a simple graph was selected (Fig. 1). A solution found using the solver plugin in Microsoft excel as well as a manual calculation was done for verification (Table 1). The steps in Dijkstra's shortest path are:

- Mark all nodes unvisited.
- Select a start node, assign it a cost of zero, assign all other nodes a cost of infinity.
- Calculate the edge costs of all neighbouring nodes to the node currently being visited.
- Mark the current node as visited and proceed to the next node with the least cost from the start node.

- If a node is re-checked from the current visited node and the cost is lower to the start node, adjust the route to the node to least cost route.
- If destination node is marked visited end the function. Otherwise select the next node with the lowest cost from the start and visit it.

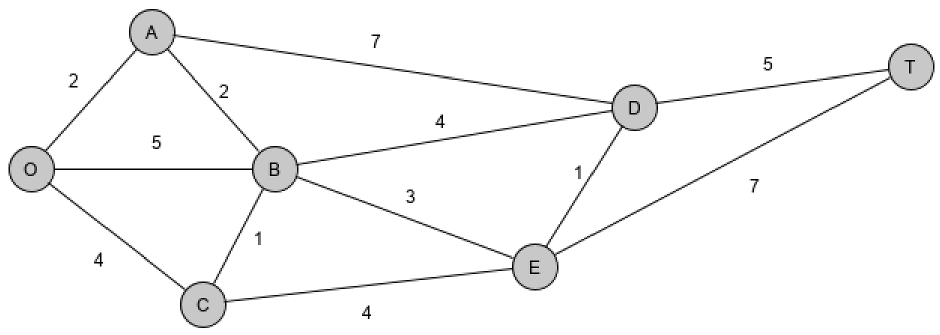

Figure 1: Sample graph used to test the algorithm where 'O' is the start node and 'T' is the destination node

Table 1 shows the procedure and steps that the algorithm works through to find the optimal route.

Table 1: Procedure to find shortest path from Figure 1

Action	Route (Cost)	Neighbours (Cost of path)	Cost from Origin
Select origin (O)	O (0)		
Find neighbouring nodes and cost		O-A (2)	2
		O-B (5)	5
		O-C (4)	4
Visit node with least cost (A)	O, A (2)		
Find neighbouring nodes and cost		A-D (7)	9
		A-B (2)	4
Visit node with least cost (B)	O, A, B (4)		
Find neighbouring nodes and cost		B-D (4)	8
		B-E (3)	7
		B-C (1)	5
Visit node with least cost (C)	O,C (4)		
Find neighbouring nodes and cost		C-B (1)	5
		C-E (4)	8
Visit node with least cost (E)	O, A ,B, E (7)		
Find neighbouring nodes and cost		E-D (1)	8
		E-T (7)	14
Visit node with least cost (D)	O, A ,B, D (8)		
Find neighbouring nodes and cost		D-T (5)	13
Calculate solution	O, A ,B, D, T (13)		

Coding of the Algorithm

Python was selected as the programming language to implement the algorithm as it works with ArcGIS by utilizing the ArcPy module. The code uses the ArcPy module to convert the elevation data into an array where each node contains a value for the elevation at that location. Nodes were given a unique identifier based on the x and y coordinates of the node. ArcPy formats coordinates (y, x), with (0,0) as the top left corner. The nodes in this data format are created in a grid structure as shown in Figure 2 wherein cell (1, 1) is the current cell and the potential neighbours are connected with arrows. With the nodes created 8 neighbours were selected based on moving to the next cell immediately surrounding the current point.

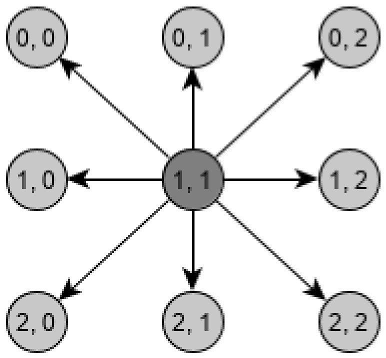

Figure 2: Potential neighbours when cell (1, 1) is the current visited cell

The code generates a list of all nodes. The start point is then selected, and all other nodes are given a tentative distance of infinity. With the start point selected and the queue generated based upon all unvisited nodes the code then iterates over all the neighbours of the start node and calculates the cost to each neighbour node. The next visited node will be selected based upon the least cost to the start node. Once the following node is selected it becomes the next visited node and is removed from the queue. The equation used to calculate cost between nodes was:

Cost = distance + (distance * slope penalty)

Slope penalty used a piece-wise scale to select the weight increment for different slopes, as shown in Table 2. Slope was calculated using absolute values since Dijkstra's algorithm can't function with negative values.

Table 2: Slope multipliers

Slope (°)	Weight Multiplier
0-5	0
6-10	1
11-15	2
16-100	1000000

To constrain the solution from having 90 degree turns the algorithm was slightly modified to limit the available neighbours based upon the route of entry to the current visited node. Figure 3 illustrates an example of how, if the current point is (1,1) and point of entry was (0, 0), the possible list of neighbours becomes (1,2), (2, 2), (2, 1).

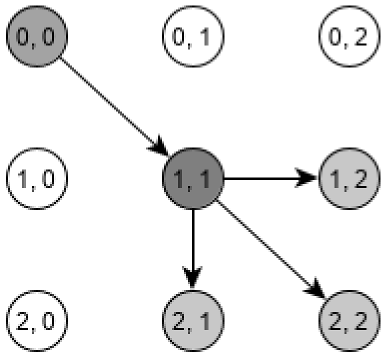

Figure 3:Radius of curve constraint route (0, 0) to (1, 1) limited list of neighbours

By altering the potential neighbours to accommodate the 90 degree radius of curve constraint the graph becomes a dynamic graph with edges being added and removed. To verify that the optimal route was being output, five sample runs were done over a 5 by 5 cell grid. Figure 4 illustrates the algorithm working over the 5 by 5 cell array and outputting the optimal route given the constraints. The route traveled start at node (0, 0) and ends at (4, 4) with a total cost 1847.

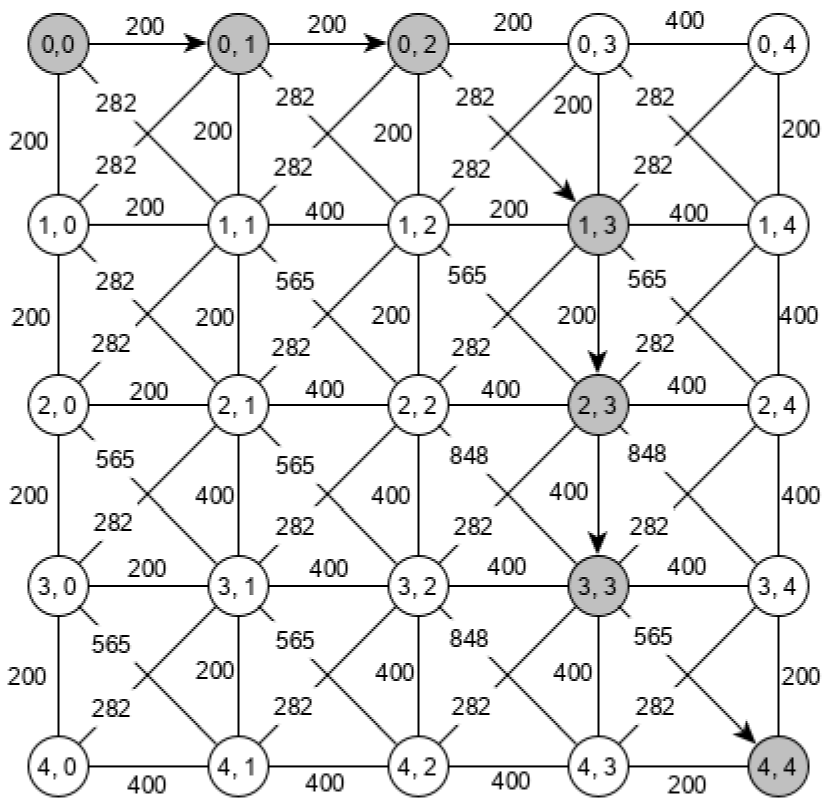

Figure 4: Validation of algorithm with 90-degree constraint

After selecting the shortest path, the array is then cleared of all elevation values. They were replaced with "No Data". The points along the selected route were given a value of 1. By creating this binary variable, the data can be returned to the Arc GIS environment and viewed spatially over the location.

Results

The results are divided into three sections with each section based on one test case demonstrating the implementation of the algorithm over a 1 km^2 tile. Each tile was separated into three categories 50 m^2 DEM, 5 m^2 DEM, and 5 m^2 LiDAR. The 50 m^2 resolution was selected based on its use in previous papers analyzing shortest paths for road networks. The 5 m^2 resolutions were selected to test efficacy of DEM compared to LiDAR data for route optimization of road networks. Considerations for optimality will be given primarily to cost, followed by visualizations of route selection over changes in elevations. Tiles will have different contour frequencies based upon the rate of change of the elevation in each tile. Cost is more direct comparison between implementations on the same cell resolution, whereas route over terrain is a more significant comparison between different cell resolution implementations.

Tile 1

Figure 5 demonstrates the three routes output by the algorithm over the first tile at each of the three categories: a) 50m DEM, b) 5m DEM, c) 5m LiDAR. Figure 6 demonstrates the routes over a topographic map with contours drawn at 5m intervals, and the routes transformed into a line to better illustrate the results.

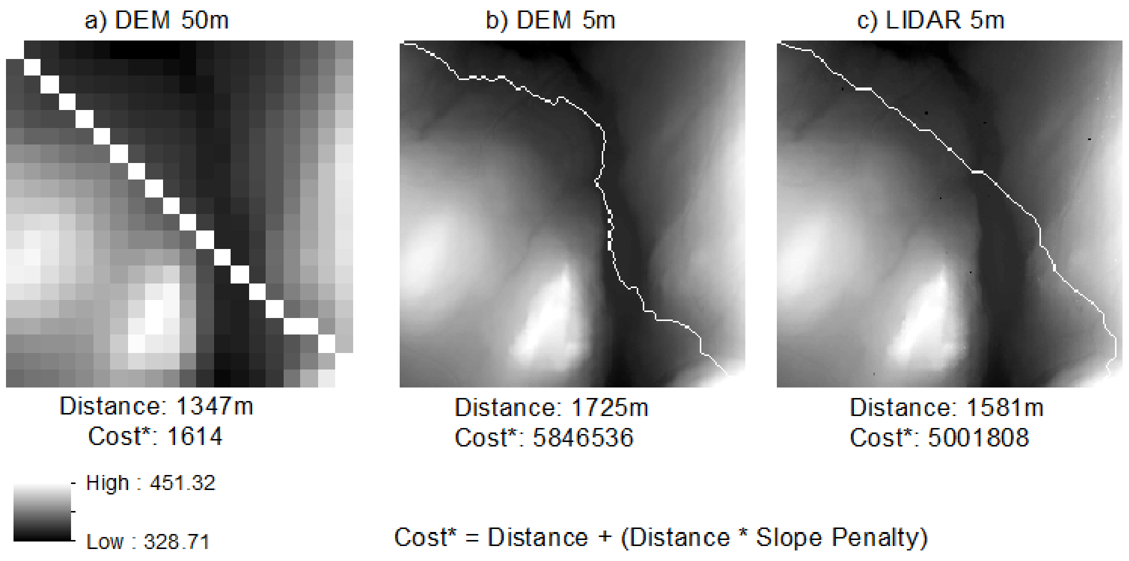

Figure 5: Tile 1 routes over digital elevation models

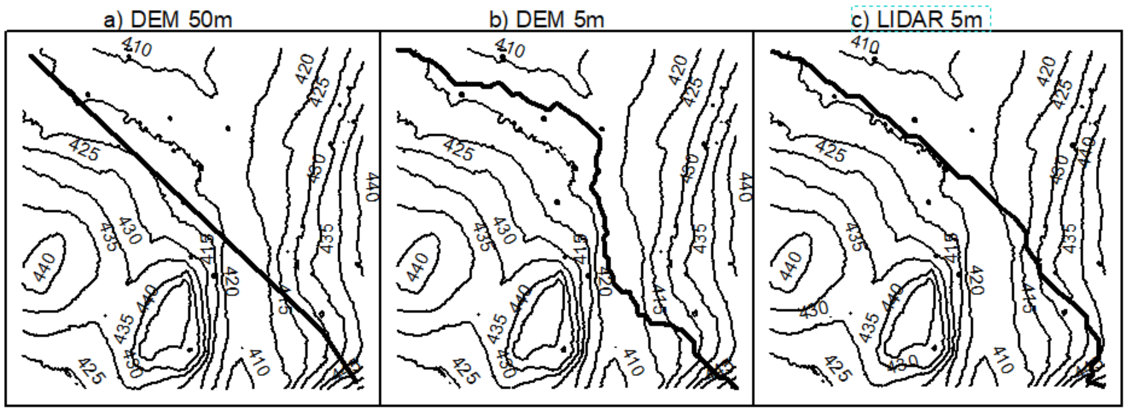

Figure 6: Tile 1 routes over topographic imagery 5m contour lines

Figures 5 and 6 a) DEM 50m demonstrates the course representation when using a 50m cell size. The coarse resolution allows the algorithm to generate a more direct route ignoring areas that at a finer resolution would have a greater slope penalty.

The trends in the 5 m^2 DEM and LiDAR show that at this resolution the route is more subjected to the terrain. The additional number of times the slope penalty is applied causes the algorithm to select a route that follows the terrain.

Differences in the DEM 5m compared to the LiDAR 5m can be attributed to the difference in interpolation techniques. The DEM having been initially taken from LiDAR and converted to a 1m cell resolution DEM and further aggregated to 5m cell resolution DEM will have elevation values more smoothed over the area than the direct conversion of LiDAR shot at 2 point per meter to a 5m cell resolution raster. The slight increase in ground elevation accuracy causes more drastic route selection due to slope penalties.

The route selected on the 50 m^2 resolution is too direct and ignores too much topography to be a viable forest road. The 5 m^2 resolutions better follow terrain making them better for pre-planning potential road locations. The LiDAR has the lowest cost which demonstrates the higher accuracy data allowed for a more optimal route selection.

Tile 2

Figures 7 and Figure 8 illustrates the results of the algorithm working over another tile with more varied topography. The contour lines drawn in Figure 8 are at 20m intervals.

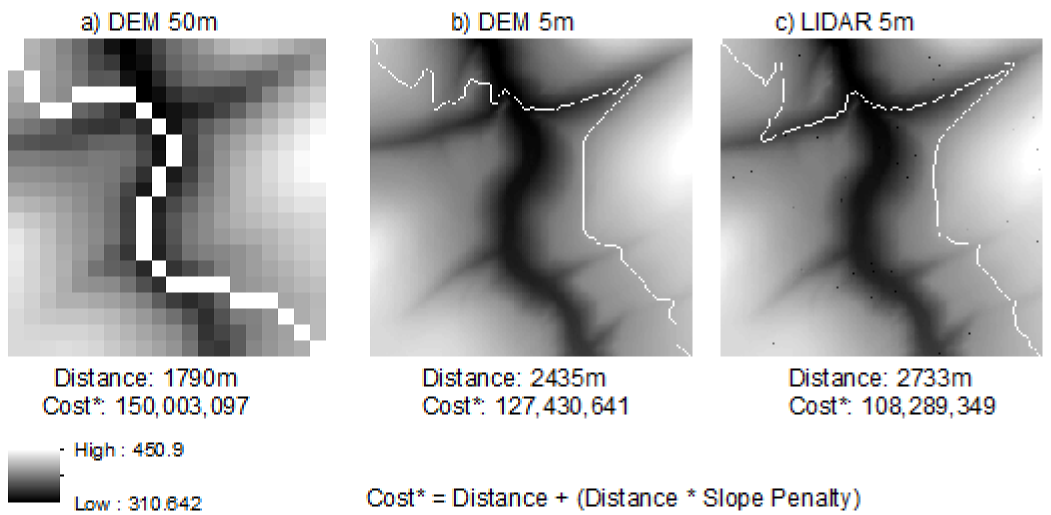

Figure 7: Tile 2 routes over digital elevation models

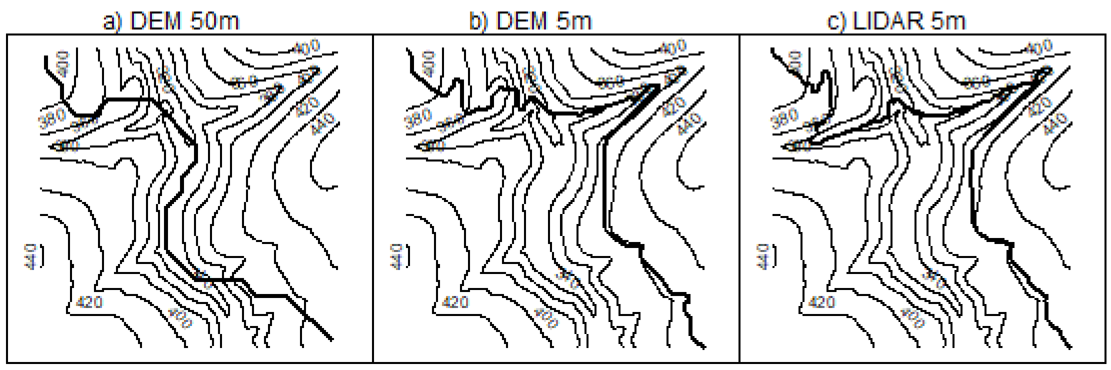

Figure 8: Tile 2 routes over topographic imagery 20m contour lines

The case of this more dynamic route is illustrated in the results shown in Figures 7 and 8. The 50m DEM route weaves into a valley to find a more static elevation, though the cost with the applied slope penalty is the highest of all the runs. The high slope penalty on the 50m DEM can be attributed to the course resolution removing potential routes to bypass infeasible areas.

All three test cases on this tile found no possible solutions without absorbing the slope penalty of 1,000,000. The results in Figures 7 and 8 b) Dem 5m and c) LiDAR 5m have more similarities to each other than to c) DEM 50m. The trend illustrated by a) and b) can be attributed to the algorithm avoiding the slope penalty as much as possible. Both the a) and b) can be seen oscillating significantly and not taking a straight a path (Fig. 9).

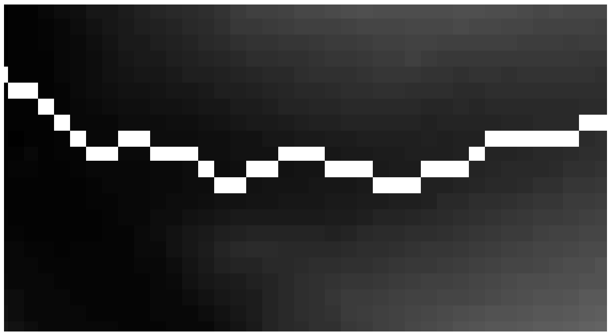

Figure 9: Example of route oscillating, close up of Figure 7 c)

The 5m DEM and 5m LiDAR demonstrate some similarities in path selection with the DEM 5m route being shorter with a higher cost. This case illustrates how the LiDAR 5m has more fine adjustments in the route compared to the 5m DEM. The fine adjustments are likely attributed to the aggregation method used on the 5m DEM. Both routes can be seen to avoid areas of high slope finding the narrowest points to cross the valleys.

Figures 7 and 8 a) DEM 50m ignores significant topography, and absorbs the slope penalty on the majority of the edges making it a sub-optimal route for a forest road. Figures 7 and 8 a) and b) demonstrate similarities in paths which indicates a more optimal solution though differences in cell elevations cause different routes. The LiDAR had the lowest cost and thus had to absorb the slope penalty making it the most optimal of the three solutions and the best potential route for a forest road.

Tile 3

Figures 10 and 11 show the routing selection over tile 3. Tile 3 had the largest change in elevation of the three tiles. The contour lines in Figure 11 are drawn at 10m intervals.

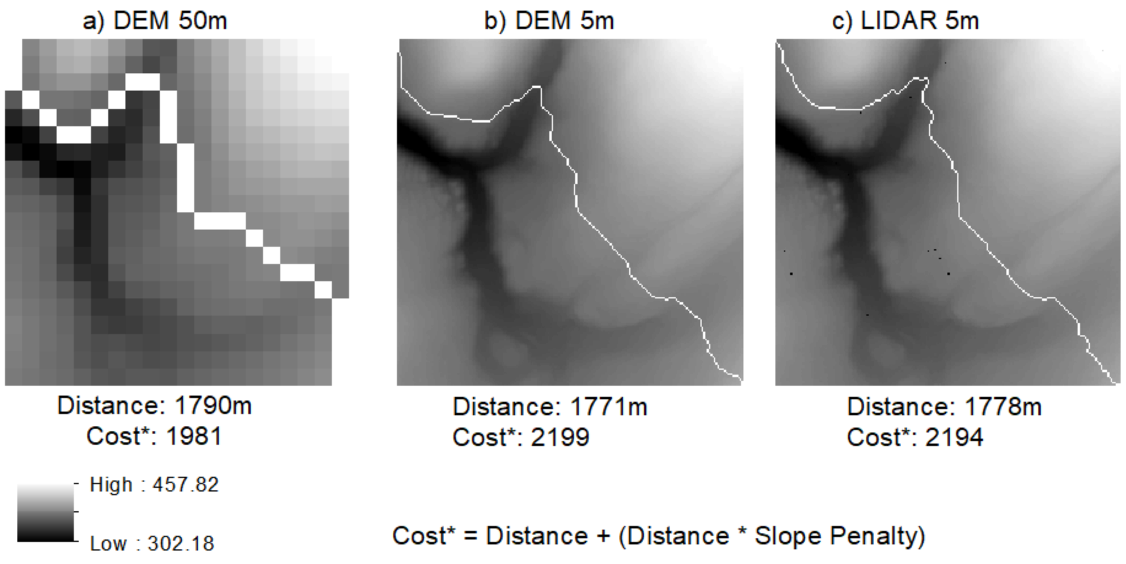

Figure 10: Tile 3 routes over digital elevation models

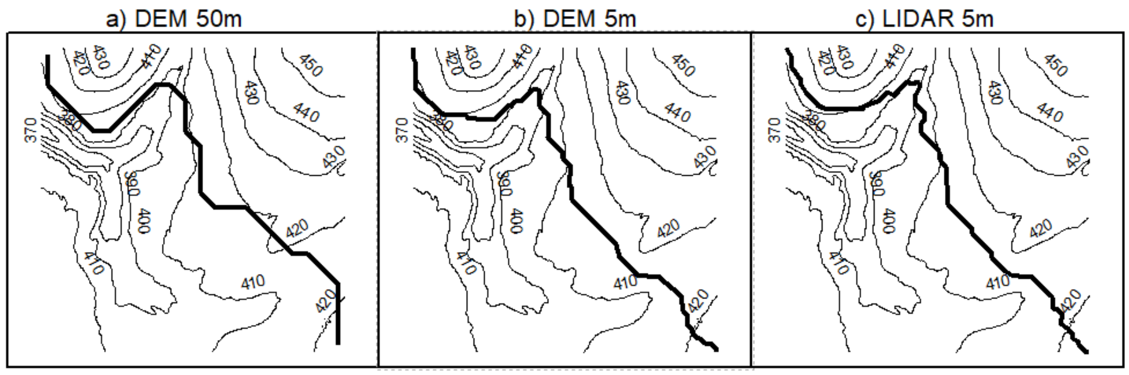

Figure 11: Tile 3 routes over topographic imagery 10m contour lines

Tile three is the only tile where the algorithm route didn't extend the slope penalty for 16-100 degrees was not applied, as shown in Figure 10. This tile also illustrates the three most similar routes, with all three crossing the top valley in a similar area before heading downwards. Figure 11 a) illustrate a route unnecessarily crossing over steeper terrain, this is due to the coarse resolution not registering the elevation changes.

Figures 10 and 11 b) and c) show very similar route section, the minor difference is likely attributed to difference in the aggregation method of the two raster images. The trend illustrated in Figures 10 and 11 c) demonstrate the algorithm providing the optimal solution, navigating through a valley and following the terrain very closely and at the lowest cost.

The coarse resolution of a) in Figures 10 and 11 is the least feasible for a forest road as it unnecessarily crosses terrain features.in Figure 10 b) has a slightly higher cost than the c). Figure 10 c) had the lowest cost making it the optimal choice for a forest road location.

Discussion

At a 5 m^2 resolution it becomes apparent that the radius of curve implementation requires further research. Working at 50 m^2, resolution the radius of curve application generates a feasible result for real world road design. Application of a radius of curve constraint along edges generates an infeasible result at 5 m^2 resolution. The infeasible curve is apparent in Figure 9 where the selected route oscillates in steep terrain. This poses an algorithmic challenge since the algorithm is designed to assess edges between three nodes, but current research on road location modelling does not address this problem. The duration of the curve would be required to extend further than a single node.

The radius of curve constraint issue could see its resolution in different forms. Modification of the graph structure and design may create feasibilities. Using a different selection criterion for neighbours may hold a solution. Changing the base algorithm may have better results.

In previous studies the graph structure used has typically been a static graph (Anderson and Nelson 2004, Stückelberger 2007). This study used a dynamic graph to implement the radius of curve constraint. Using a dynamic graph creates additional complications as Dijkstra’s shortest path is not designed to be run over this graph structure (Sunita and Garg 2018). There are many implementations and modifications to Dijkstra’s algorithm to assist in using it over dynamic graphs. The addition and deletion of edges as the code runs creates a complication. Potential routes only exist depending on the route of entry to a visited node. This implementation worked for the modification but that is because the structure of the data was a grid. Having a grid structure ensured that there were always three points of entry to and from any point. This variability allowed an optimal solution to be presented. Given a different graph structure, such as the graph in Figure 1, this modification can run into errors where it will be unable to find a feasible solution.

Previous studies on this subject have used DEM imagery, but currently LiDAR is becoming more freely available. The added accuracy of LiDAR combined with the lower cost demonstrated by LiDAR make it a likely to be used in future research. LiDAR was not used and was less available during the previous study periods for the papers referenced in this study.

With increases in computational power and an increase in high accuracy data availability it is becoming more feasible to allocate forest roads using computational models. Models such as the one in this study can be used to greatly reduce the planning costs associated with forest road planning and reduce manual planning endeavours.

Future potential in these models would be to build in harvest allocation selections. Modelling both the harvest selection and associated road networks simultaneously has the potential to greatly reduce road costs. Route modelling could also be expanded to simultaneously assess not only the route but also the width feasibility, and cut and fill requirements.

Conclusion

The goal of this study was to demonstrate the viability of forest road location models for planning forest roads. This model used Python, ArcGIS and a modified version of Dijkstra's shortest path algorithm to calculate optimal road locations.

Testing of the model and running it over various graphs of increasing complexity gives confidence that the result obtained through the model is the optimal solution. The results confirm that with a larger cell resolution the algorithm will produce sub-optimal solutions as significant topography is ignored. Smaller cell resolution data outputs more nuanced routes that follow the topography more closely. Both 5 m^2 resolution DEM and LiDAR produce very similar results. LiDAR data had the lower cost.

The study demonstrates that with increased data accuracy road network routing models will produce more feasible solutions when accounting for changes in elevation across a landscape. The advancements in LiDAR technology and its recent availability will enhance the potential of forest road optimization.

Acknowledgements

I would like to thank the following people for their guidance and contributions. Dr Kevin Crowe, for guiding me through this process and not letting me take shortcuts. Tomislav Sapic for assisting me and talking me through issues as they arose. Andrew Vester for teaching me how to organize code and properly use an interactive development environment and the debugger function. Philip Stankowski for aiding me in writing structure and clarity.

References

Abdi, E., Majnounian, B., … A.D.-J. of F., and 2009, undefined. (n.d.). A GIS-MCE based model for forest road planning. agriculturejournals.cz. Available from https://www.agriculturejournals.cz/publicFiles/52_2008-JFS.pdf [accessed 5 April 2019].

Akay, A.E., Aruga, K., Bettinger, P., and Sessions, J. 2013. Using Optimization Techniques in Designing Forest Roads and Road Networks. (1998): 1–2.

Akay, A.E., Oğuz, H., Karas, I.R., and Aruga, K. 2009. Using LiDAR technology in forestry activities. Environ. Monit. Assess. **151**(1–4): 117–125. Springer Netherlands. doi:10.1007/s10661-008-0254-1.

Anderson, A.E., and Nelson, J. 2004. Projecting vector-based road networks with a shortest path algorithm. Can. J. For. Res. **34**(7): 1444–1457. NRC Research Press Ottawa, Canada . doi:10.1139/x04-030.

Aruga, K., Sessions, J., Akay, A., and Chung, W. 2005. Simultaneous optimization of horizontal and vertical alignments of forest roads using Tabu Search. Int. J. For. Eng. **16**(2): 137–151. doi:10.1080/14942119.2005.10702522.

Bont, L.G., Heinimann, H.R., and Church, R.L. 2012. Concurrent optimization of harvesting and road network layouts under steep terrain. Ann. Oper. Res. **232**(1): 41–64. Springer US. doi:10.1007/s10479-012-1273-4.

Cortes, C., Pregibon, D., and Volinsky, C. 2003. Computational Methods for Dynamic Graphs. J. Comput. Graph. Stat. **12**(4): 950–970. Taylor & Francis. doi:10.1198/1061860032742.

Dijkstra, E.W. 1959. A note on two problems in connexion with graphs. Numer. Math. **1**(1): 269–271. Springer-Verlag. doi:10.1007/BF01386390.

ESRI. (n.d.). What is ArcGIS? | ArcGIS for Developers. Available from https://developers.arcgis.com/labs/what-is-arcgis/ [accessed 12 April 2019].

Henningsson, M., Karlsson, J., and Rönnqvist, M. 2007. Optimization Models for Forest Road Upgrade Planning. J. Math. Model. Algorithms **6**(1): 3–23. Kluwer Academic Publishers. doi:10.1007/s10852-006-9047-0.

Hillier, F.S., and Lieberman, G.J. 2005. Introduction to operations research. McGraw-Hill Higher Education.

Liu, K., and Sessions, J. 1993. Preliminary Planning of Road Systems Using Digital Terrain Models. J. For. Eng. **4**(2): 27–32. Taylor & Francis Group . doi:10.1080/08435243.1993.10702646.

Rönnqvist, M. 2003. Optimization in forestry. Math. Program. **97**(1): 267–284. Springer-Verlag. doi:10.1007/s10107-003-0444-0.

Stückelberger, J. 2007. A weighted-graph optimization approach for automatic location of forest road networks. Available from https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/150287/eth-30141-02.pdf?sequence=2 [accessed 11 April 2019].

Stückelberger, J., Heinimann, H.R., and Chung, W. 2007. Improved road network design models with the consideration of various link patterns and road design elements. Can. J. For. Res. **37**(11): 2281–2298. doi:10.1139/X07-036.

Stückelberger, J.A., Heinimann, H.R., and Burlet, E.C. 2006. Modeling spatial variability in the life-cycle costs of low-volume forest roads. Eur. J. For. Res. **125**(4): 377–390. Springer-Verlag. doi:10.1007/s10342-006-0123-9.

Sunita, and Garg, D. 2018. Dynamizing Dijkstra: A solution to dynamic shortest path problem through retroactive priority queue. J. King Saud Univ. - Comput. Inf. Sci. Elsevier. doi:10.1016/J.JKSUCI.2018.03.003.

US Department of Commerce, N.O. and A.A. (n.d.). What is LIDAR. Available from https://oceanservice.noaa.gov/facts/LiDAR.html [accessed 12 April 2019].

What is a DEM - Digital Elevation Model Definition. (n.d.). Available from https://www.caliper.com/glossary/what-is-a-digital-elevation-model-dem.htm [accessed 12 April 2019].

Wilson, R.J. 2010. Introduction to graph theory. Prentice Hall/Pearson.

Appendices

The code

Attached at the end of this document is a copy of the python code in a self contained 4-page document. This one done in order to maintain the formatting and readability of the code.

```


import sys
import arcpy
import numpy
import math
import timeit


NEIGHBOUR_CELL_NUMBER_MAX = 1

#Find the shortest path between start and end nodes in a graph
def shortestpath(graph,start,end, max_x, max_y, visited=[],distances={},predecessors={}):

    unvisited_nodes = create_nodes_list(graph)



    # we've found our end node, now find the path to it, and return
    while start != end:

        print start


        # detect if it's the first time through, set current distance to zero
        if not visited: distances[start]=0

        start_x, start_y = get_node_coordinates(start)

        if predecessors == {}:
            previous_node = start
        else:
            previous_node = predecessors[start]


        xprev, yprev = get_node_coordinates(previous_node)

        x_relative = xprev - start_x
        y_relative = yprev - start_y

        # process neighbors as per algorithm, keep track of predecessors
        neighbours = find_neighbour_nodes(start_x, start_y, x_relative, y_relative,
        max_x, max_y)

        for neighbour in neighbours:
            if neighbour not in visited:

                neighbour_x, neighbour_y = get_node_coordinates(neighbour)
                neighbour_distance = calculate_distance(start_x, start_y, neighbour_x,
                neighbour_y)

                neighbour_slope = calculate_slope(neighbour_distance,
                graph[start_x][start_y], graph[neighbour_x][neighbour_y])
                neighbour_weight = calculate_weight(neighbour_slope, neighbour_distance)

                neighbourdist = distances.get(neighbour, sys.maxint)
                tentativedist = distances[start] + neighbour_weight
                if tentativedist < neighbourdist:

                    distances[neighbour] = tentativedist
                    predecessors[neighbour]=start

        # neighbors processed, now mark the current node as visited
        visited.append(start)
        unvisited_nodes.remove(start)

        # finds the closest unvisited node to the start
```

```
            unvisiteds = dict((k, distances.get(k,sys.maxint)) for k in unvisited_nodes)
            closestnode = min(unvisiteds, key=unvisiteds.get)

            start = closestnode


        path=[]
        while end != None:
            path.append(end)
            end=predecessors.get(end,None)
        return distances[start], path[::-1]

    #creates a list of all nodes in the graph
    def create_nodes_list(graph):
        node_list = []
        for x in range(0,len(graph[0])):
            for y in range(0,len(graph)):
                node_list.append(get_node_id(x,y))

        return node_list

    # #create a list of all arcs in the graph
    # def generate_arc_list(graph, list_of_nodes, neighbour_nodes):
    #     arc_list = []


    # #create dictionary af all arcs in the graph
    # def dictionary_arcs_w_weight(graph, list_of_nodes, neighbour_nodes, weights):
    #     arc_dictionary = {}



    #calculates distance between current node and neighbour
    def calculate_distance(x1, y1, x2, y2):
        return math.sqrt(((x1-x2) * cellSize)**2 + ((y1-y2) * cellSize)**2)

    #calculates weight from distance and slope
    def calculate_weight(slope, distance):
        return distance + (distance * slope)


    #calculates slope as an absolute vale, applies piecewise scale for interpreting slope
    def calculate_slope(distance, z1, z2):
        slope = abs((((z1) - (z2)) / distance) * 100)

        weighted_slope = 0

        if slope >= 0 and slope <= 5:
            weighted_slope = 0
        elif slope > 5 and slope <= 10:
            weighted_slope = 1
        elif slope > 10 and slope <= 15:
            weighted_slope = 2
        elif slope > 15:
            weighted_slope = 1000000

        return weighted_slope

    # #finds all neuighbour nodes, based on the global NEIGHBOUR_CELL_NUMBER_MAX value
    # def find_neighbour_nodes(x1, y1, max_x, max_y):
    #     neighbours = []

    #     for x_value in range(-1*NEIGHBOUR_CELL_NUMBER_MAX, 1*NEIGHBOUR_CELL_NUMBER_MAX+1):
    #         for y_value in range(-1*NEIGHBOUR_CELL_NUMBER_MAX,
    1*NEIGHBOUR_CELL_NUMBER_MAX+1):
    #             neighbour_x = x1+x_value
    #             neighbour_y = y1+y_value
```

```

#             if not (neighbour_x >= max_x or neighbour_x < 0 or neighbour_y >= max_y or neighbour_y < 0 or (neighbour_x == x1 and neighbour_y == y1)):
#                 neighbours.append(get_node_id(neighbour_x, neighbour_y))

#     return neighbours

    #finds all neuighbour nodes, based on the global NEIGHBOUR_CELL_NUMBER_MAX value
def find_neighbour_nodes(x1, y1, xprev, yprev, max_x, max_y):
    neighbours = []
    x_values = []
    y_values = []

    if xprev == 0 and yprev == 0:
        x_values = [-1, 0, 1]
        y_values = [-1, 0, 1]
    elif xprev == -1 and yprev == -1:
        x_values = [0, 1]
        y_values = [0, 1]
    elif xprev == 0 and yprev == -1:
        x_values = [-1, 0, 1]
        y_values = [1]
    elif xprev == 1 and yprev == -1:
        x_values = [-1, 0]
        y_values = [0, 1]
    elif xprev == -1 and yprev == 0:
        x_values = [1]
        y_values = [-1, 0, 1]
    elif xprev == 1 and yprev ==  0:
        x_values = [-1]
        y_values = [-1, 0, 1]
    elif xprev == -1 and yprev == 1:
        x_values = [0, 1]
        y_values = [-1, 0]
    elif xprev == 0 and yprev == 1:
        x_values = [-1, 0, 1]
        y_values = [-1]
    elif xprev == 1 and yprev == 1:
        x_values = [-1, 0]
        y_values = [-1, 0]

    for x_value in x_values:
        for y_value in y_values:
            neighbour_x = x1+x_value
            neighbour_y = y1+y_value



            if not (neighbour_x >= max_x or neighbour_x < 0 or neighbour_y >= max_y or neighbour_y < 0 or (neighbour_x == x1 and neighbour_y == y1)):
                neighbours.append(get_node_id(neighbour_x, neighbour_y))

    return neighbours


#interprets cell position to generate a string identifier for the cells
def get_node_id(x, y):
    return str(x) + "," + str(y)

#splits the coordinates into x or z and numeric value
def get_node_coordinates(id):
    split = id.split(',')
    return int(split[0]), int(split[1])

#takes list of coordinates and changes the z values in the array to 1
def selected_cells(path_coordinates, graph):
    for coordinate in path_coordinates:
        x, y = get_node_coordinates(coordinate)
        graph[x][y] = 1
    return graph

# makes all points not in the path 0
def zeroing_graph_values(graph):
    for x in range(0,len(graph[0])):
        for y in range(0,len(graph)):
            graph[x][y] = 0
    return graph

#creates a weight for 90 degree turning radius
def radius_of_curve(x1, y1, x2, y2, x3, y3):
    pass




if __name__ == "__main__":



    start_timer = timeit.default_timer()
    print start_timer

    raster =
    arcpy.Raster(r'C:\Users\Administrator\Documents\Lakehead\thesis\lid_rast\Run8\lid_8.t
    if')
    lowerLeft = arcpy.Point(raster.extent.XMin,raster.extent.YMin)
    cellSize = raster.meanCellWidth

    graph = arcpy.RasterToNumPyArray(raster, lowerLeft, nodata_to_value=100000)

    max_x = len(graph[0])
    max_y = len(graph)

    weight, shortest_path = shortestpath(graph, "0,0", "199,199", max_x, max_y)
    print weight, shortest_path

    zeroed_graph = zeroing_graph_values(graph)

    route = selected_cells(shortest_path, zeroed_graph)

    myraster = arcpy.NumPyArrayToRaster (route, lowerLeft, cellSize, cellSize, 0)
    myraster.save
    (r'C:\Users\Administrator\Documents\Lakehead\thesis\lid_rast\Run8\lid_8_route.tif')


    stop_timer = timeit.default_timer()
    print stop_timer


    print ('Time: ', stop_timer - start_timer)

```