Implementation of a Cost Function to Model Travel Cost for Shortest Path Routing

Carlos Rivera López
Master in Computer Science
Advisor: Jeffrey Duffany, Ph.D.
Electrical and Computer Engineering & Computer Science Department
Polytechnic University of Puerto Rico

Abstract *— Many times, as we travel in a region from one place to another we start wondering if there's a better, optimal path to travel from point A to point B. In order to figure this out, normally, we would seek for details of some of the possible ways of finding such optimal path. However, during our analysis we then start to see things that are subjective such as "steep uphills followed by steep downhills" and we wonder if a path with those characteristics would indeed be the optimal path from point A to B. Through this project we attempt to create a cost function that can help us answer such question. This cost function would take terrain data such as latitude, longitude, elevation, to compute a cost based on constraints subject to the user's interest. With this information we intend to produce a node graph to model a region of interest in a map that shows the optimal path from point A to B.*

Key Terms *— Bellman-Ford, Cost Function, Shortest Path, Travel Cost.*

INTRODUCTION

Many times, as we travel in a region from one place to another we start wondering if there's a better, optimal path to travel from point A to point B. In order to answer this question we could ask locals for their opinion based on the fact that they may be more familiar with a map or region, or one could look for details of some of the possible ways of finding such optimal path.

However, during our analysis we often start to see things that are subjective such as "steep uphills followed by steep downhills" and we wonder if a path with those characteristics would indeed be the optimal path from point A to B. Sometimes, it may be really complicated to determine what the optimal travel path is because each user may have a different mindset on constraints, or limitations. Some constraints or limitations that could influence one's analysis when determining such optimal path could be:

- Toll Stations
- Road conditions
- Terrain contour (mountainous or relatively flat)
- Traffic congestion
- Distance

Through this project we attempt to create a cost function that can help us answer such question. This cost function would take terrain data such as latitude, longitude, elevation, to compute a cost based on constraints subject to the user's interest. The main driver of this cost function will be the distance and slope of inclination from one point to another. Based on these main parameters for the cost function the cost will be then computed by taking into account a subjective opinion from the user as to how much cost could an uphill affect the base cost of a path, or how a downhill could affect the base cost of a path; these are interpreted as penalties and bonuses, respectively.

With this information we intend to produce a directed node graph to model a region of interest in a map in which we can model the cost function based on the user's subjective opinion of uphill penalties, and downhill bonuses. The end result would be a node graph that can show the map with the costs (i.e. edge weights) that represent how much cost does it take to go from one node to another. After we have properly modeled the map after the cost function and map data, we will be using a shortest path algorithm: Bellman-Ford [1]; that can compute the shortest path from point A to B. Finally, we shall compare two similar maps, with one subtle but significant difference, to observe how the cost function would compute a different shortest path depending on the available routes (i.e. paths) from a source to a destination.

BACKGROUND AND RELATED WORK

The main reason why this topic was selected to be a project idea, mainly was because there is not much work done in this area. Usually when we search for research topics such as Encryption, Steganography, we find that there a very active academic community contributing to that area. For areas involving "Shortest path", as we attempt to find any existing work we start to see that there are little to no contributions.

One famous software we may use on a daily basis is Google Maps [2]. This software mainly answers one question: "How do I get from point A to point B as fast as possible?". This software also allows the user to add certain constraints such as avoiding toll stations. When the software has an answer it outputs such path, with some additional information to the user. However, while this software is really good at doing that we cannot see what's exactly doing to compute such path.

This project then attempts to produce a cost function that could be used to determine the cost of traveling from one location to another. This cost function is to be used with a single source shortest path algorithm such as Bellman-Ford single source shortest path so that we can obtain an answer to a more specific question: "How much does it actually cost to go from point A to point B?"

In addition to this, the US Department of Energy (DOE) has also conducted some research regarding the fuel efficiency of heavy vehicles [3] based on terrain conditions such as: road grade, travel speed, vehicle weight, among others. A complimentary research to the previous research was conducted by the National Laboratory of Renewable Energy (NREL), as part of the DOE, which consisted of studying the consumption of energy of modern automobiles [4]. For that research, different types of vehicles (i.e. gasoline, electric, High Efficiency Vehicles) were put through simulation of a different series of trips with varying road grades and terrain conditions to determine how these conditions could affect the energy consumption of similar or comparable vehicles.

For both of these researches that were conducted by the DOE we can see that road grade and terrain conditions can affect the energy consumption of any vehicle. With this project we look forward to build an equation to estimate the cost of traveling through different paths in a given terrain to determine which path is the most cost-effective based on distance and changes in elevations.

PROBLEM

The main problem of this project was to model travel cost depending on terrain data such as elevation and distance. As we travel constantly to our day-to-day places, we may be asking ourselves "Is there any better way of doing this?" The problem with this question, albeit a simple one, is that the answer can be hard to answer since there is no clear-cut way of determining when a steep uphill becomes "very" costly to travel through, or when a downhill becomes "very" relieving to travel through. With so many variables being taken into account it just becomes too hard to come up with a simple answer.

Out of all the possible variables that could be involved when attempting to answer the main question we have:

- Slope
- Distance
- Initial costs required to just travel though a certain path (e.g. toll stations)
- Medium (e.g. on foot, bicycle, gasoline car, electric vehicle, etc)
- Road conditions
- Congestion
- Traffic lights

Through this project, we look forward to create a cost function that can compute a cost of travelling through a certain path such that we can weigh all the paths in a certain region of a map, and use an existing algorithm for shortest path to find a path that can answer our main question.

METHODOLOGY

To solve this we then modeled locations as a 3D coordinate composed of latitude, longitude, and elevation. Since this information is, mostly, publicly available through many sources such as the United States Geological Survey (USGS) or Google we could choose a local region to create, test, and tune our cost function.

Any region that we would choose for our project could be modeled properly by using a directed graph whose vertices would contain: latitude, longitude, elevation, and neighbors; and edges would contain: source, destination, and cost. The reason why a directed node graph was chosen over a non-directed node graph was because the former would allow us to properly model cases in which the paths (i.e. streets) are one-way only. The cost function would take a source and a destination as inputs and output the computed cost of traveling through that path.

$$costUp(a,b) = i + d\left(1 + \frac{p * s}{m}\right) \tag{1}$$

$$costDown(a,b) = i + d\left(1 - \frac{b * s}{m}\right) \tag{2}$$

Because we look forward to make this project scalable for many applications we left some variables in the cost equations for uphill and downhill scenarios. In both equations (1) and (2) we have some terms that are being used to compute a travel cost from point *A* to point *B*, among these terms we have some variables:

- d to represent a distance cost. For this project we computed the Euclidean distance between a, and b.
- p to represent an uphill penalty (e.g. 0.70 for 70%) – used only for uphill cases.
- b to represent a downhill bonus (e.g. 0.25 for 25%) – used only for downhill cases.
- s to represent the slope between two points.
- m to indicate a maximum allowed slope.

With these variables we expect to leave some room for different scenarios such as high-slope or low-slope scenarios where a certain threshold of inclination is possible. For this project these variables were set to meet some basic assumptions; a maximum penalization for uphill paths was set to 70%, and maximum bonus of 25% based on the slope of the path and the distance between the two points. If, however, a path with a slope greater than 45 degrees of uphill is found the cost will be set to infinity; the same would apply for any downhill with 45 degrees of inclination.

To compute the slope between two coordinates in a 3D space we can use vector math to build a right triangle. Since we have coordinates from point *A* to point *B* we only need to compute a coordinate point *C* for the triangle base, as shown in Figure 1. At this point we can obtain the vector $\overrightarrow{AB}$ and vector $\overrightarrow{AC}$, normalize them, and obtain the angle between these two vectors by computing $\cos^{-1}(\overrightarrow{AB} \bullet \overrightarrow{AC})$. For downhill scenarios, the angle would be computed by performing the same operation but with vectors $\overrightarrow{BA}$ and $\overrightarrow{BC}$.

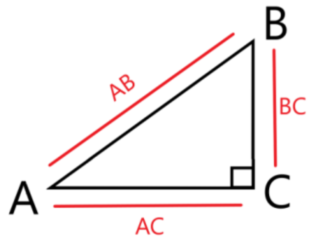


Figure 1
Right Triangle Made with the Start and End Coordinates (Latitude, Longitude, Elevation)

With the slope of the path from A to B at hand, we then compute a penalty or bonus to the distance cost based on a percentage of the slope divided by the max slope. In uphill scenarios, the cost function will favor short-distance, low incline paths over long-distance, high incline paths; that is, the shorter and the lower slope of the path will result in a lower the penalty. For downhill scenarios, the cost function will favor long-distance, high incline paths over short-distance, low-incline paths. It's important to note that, based on the cost functions (1) and (2), we can see that the cost is mainly based, or applied, to the base distance from point *A* to *B*.

This cost function will output a number indicating the total cost of traveling through that edge in the node graph. In Figure 2, we can see a line plot to better visualize how the cost function will determine the cost of traveling through an uphill path. Additional to the computed cost, there is an additional cost that can be added to the equations to take into account other subjective external costs that the cost function is not able to model. This initial value can be used to influence the total cost from one vertex to another. Some applications in which this initial cost can be beneficial is when trying to add a cost attributed to the road conditions, congestion, traffic lights, or for toll stations. The cost produced by equations (1) and (2), even without the initial cost taken into account (i.e. $i = 0$) is expected to grow linearly according to the variables that can affect the cost.

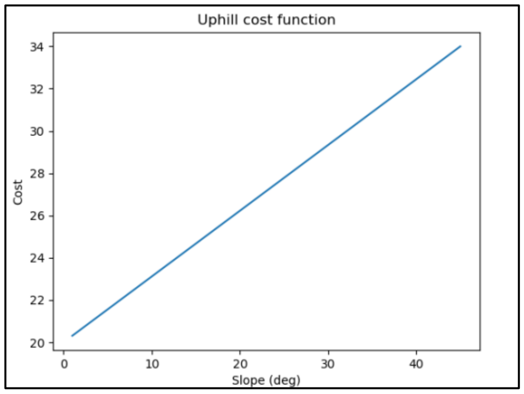


Figure 2
Uphill Cost Function Plot, with Slopes (*s*) Ranging from 1° to 45°, Maximum Slope (*m*) of 45°, a Maximum Penalty (*p*) of 70%, Initial Cost (*i*) of 0, and a Base Distance (*d*) of 20 Units

With the cost function now at hand we can then compute the cost of all the edges in the node graph. After all the edge weights have been computed we can use Bellman-Ford single source shortest path algorithm to determine what is the optimal path from point *A* to *B*.

In order to visualize the data, along with the computed shortest path based on our cost functions from equations (1) and (2) we used a Python [5] package that could help us plot data and visualize graphs: Matplotlib [6], and NetworkX [7], respectively.

Results and Discussion

First off, we tested this cost function with different scenarios with two algorithms for single source shortest path, namely, Bellman-Ford and Dijkstra's [8] algorithms. The maps that were used to test the cost functions were similar to the map presented in Figure 3; five nodes with one of these placed at a higher elevation to the other four nodes, and six edges to connect the vertices.

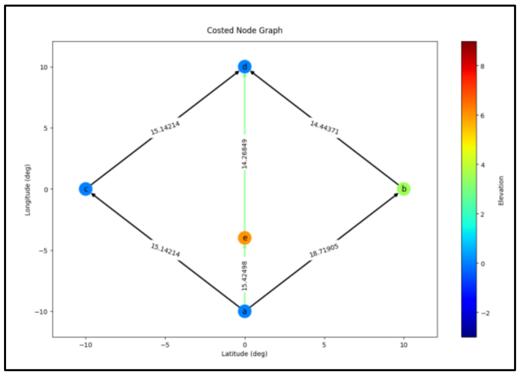


Figure 3
Node Graph with Edge Weights Modeled after the Cost Function

To output the shortest path modeled by the cost function, we then represented the node graph as a 2D plot by having the X axis represent latitudes, and the Y axis represent the longitudes in a 2D coordinate system. The elevation was represented through a Color Map. When a shortest path is computed the edges of the graph are painted in green color to highlight such path, the rest of the paths are plotted in black color; with all edges having their respective weights. This visualization can be observed in Figure 3.

All tests produced results that were consistent to the design of the cost function, and as early as in Figure 3 we can see the some of the results of modeling the edge weights with the cost function. We can see that the cost to travel from node A to node E is nearly the same as travelling from node A to node C, considering that the only difference between these two scenarios is the change in elevation; the former is an uphill scenario while the latter is a plain terrain scenario (i.e. no change in elevation).

As we tested the cost function with different maps we obtained interesting results. However, another scenario was tested; this scenario introduced two nodes and three edges to the map; this new map can be observed in Figure 4. The main purpose of testing whether or not the cost function would indicate that a better, cheaper, alternate shortest path would be favored instead of going through uphill and downhill scenarios as shown in Figure 3. In this map, we can see that all of the nodes in the left side of the graph have the same elevation, allowing us to introduce a shortest path by travelling through a relatively flat surface.

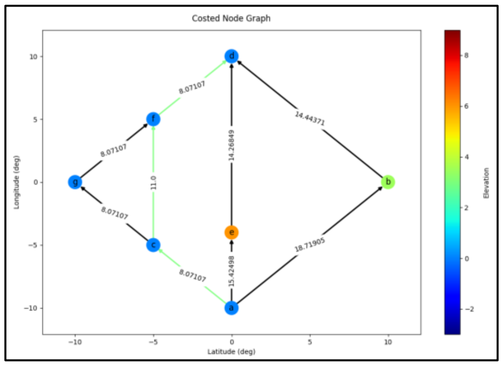


Figure 4
Shortest Path by going through a Flat Path around the Uphill

By comparing the results obtained when modeling the travel costs and the computed shortest path we can see how the cost function affects the computed shortest path. In the first map, we provided three possible paths to go from node A (bottom) to node D (top): A flat path {A, C, D}, a path with subtle changes in elevation {A, B, D}, and a path with harsh changes in elevation {A, E, D}. For the second map, although with different node labeling, we introduced an additional path from A to D: {A, C, F, D} which geometrically shorter than {A, C, D} in the first map. Both maps had their cost computed with the same parameters of the equations, leaving map (i.e. geometric) differences to produce changes in results.

To accomplish this, we created a Python script (Version 3.7.2; Python Software Foundation) that would read the data from a Comma Separated Values (CSV) file to model a weighted, directed node graph, compute all the costs for all the graph edges, compute the shortest path and output a visualization of such shortest path. A flowchart of this process can be observed in Figure 5.

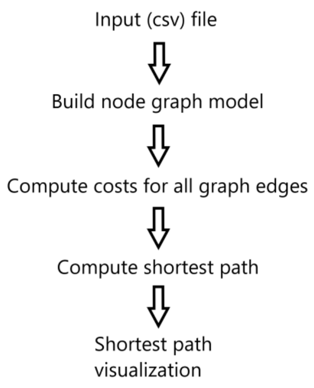
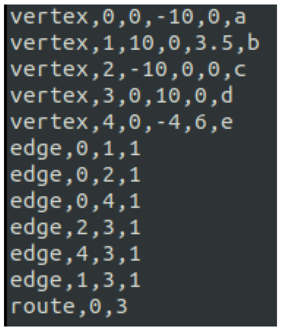

Input (csv) file
Build node graph model
Compute costs for all graph edges
Compute shortest path
Shortest path visualization

Figure 5
Flowchart of Python Script to Compute a Shortest Path for a Given Start and Destination Nodes in a Graph

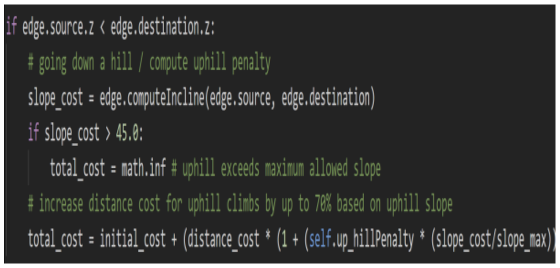
For the CSV input file, we can observe in figure 6 we can observe how the input would be formatted. The input file format is formatted such that we can read all the necessary input to model such node graph from one file only. The input(s) are classified as "vertex", "edge", "route"; each contains a specified set of values. The input for "vertex" contains a logical ID (key), latitude (as X component), longitude (as Y component), elevation (as Z component), and a label. The input for "edge" contains values for a source, destination, and initial cost (by default all initial costs were set to 1). The last type of input we can process in this input file is the "route" which contains values for a start and end, to indicate from which node we want to compute a shortest path for a given destination.



```
vertex,0,0,-10,0,a
vertex,1,10,0,3.5,b
vertex,2,-10,0,0,c
vertex,3,0,10,0,d
vertex,4,0,-4,6,e
edge,0,1,1
edge,0,2,1
edge,0,4,1
edge,2,3,1
edge,4,3,1
edge,1,3,1
route,0,3
```

Figure 6
Sample CSV Input File Used to Model a Weighed, Directed Node Graph

After all the node and edges from the directed node graph would be put together, all the weights (costs) were computed using the cost function described in equations (1) and (2) accordingly (i.e. uphill cases were applied equation (1) and downhill cases were applied equation (2)); in Figure 7 we see a snippet of the Python script that computes the cost for uphill cases.

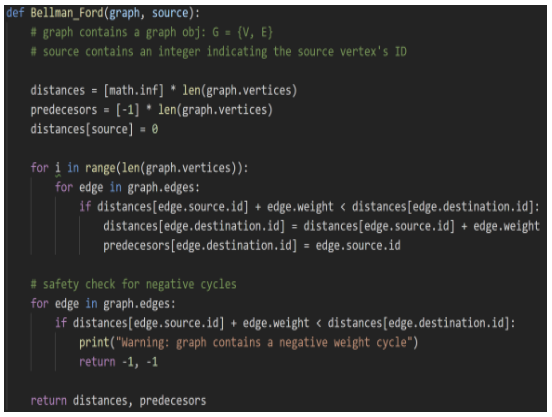


```python
if edge.source.z < edge.destination.z:
    # going down a hill / compute uphill penalty
    slope_cost = edge.computeIncline(edge.source, edge.destination)
    if slope_cost > 45.0:
        total_cost = math.inf # uphill exceeds maximum allowed slope
    # increase distance cost for uphill climbs by up to 70% based on uphill slope
    total_cost = initial_cost + (distance_cost * (1 + (self.up_hillPenalty * (slope_cost/slope_max))
```

Figure 7

Snippet of the Python Script that Computes the Cost for an Uphill Path

When all of the edge costs (weights) in node graph are computed we use the Bellman-Ford algorithm for single-source shortest path. An implementation of this algorithm can be observed in figure 8. To build the node graph model we created classes to represent Vertices and Edges in a graph, this allows us to manipulate data more easily while maintaining the data organized (i.e. encapsulated). The Graph class would contain the cost function, and some other additional elements such as uphill penalty and downhill bonus variables.



```python
def Bellman_Ford(graph, source):
    # graph contains a graph obj: G = {V, E}
    # source contains an integer indicating the source vertex's ID

    distances = [math.inf] * len(graph.vertices)
    predecesors = [-1] * len(graph.vertices)
    distances[source] = 0

    for i in range(len(graph.vertices)):
        for edge in graph.edges:
            if distances[edge.source.id] + edge.weight < distances[edge.destination.id]:
                distances[edge.destination.id] = distances[edge.source.id] + edge.weight
                predecesors[edge.destination.id] = edge.source.id

    # safety check for negative cycles
    for edge in graph.edges:
        if distances[edge.source.id] + edge.weight < distances[edge.destination.id]:
            print("Warning: graph contains a negative weight cycle")
            return -1, -1

    return distances, predecesors
```

Figure 8

Snippet of the Implementation of the Bellman-Ford Algorithm for Single-Source Shortest Path in our Python Script

The end result is what we have discussed at the beginning of this section, in Figures 3 and 4. With this Python script we are now able to model basic terrain (or map) data using a weighted, directed node graph and compute a shortest path using our cost function.

CONCLUSION

The results that we obtained as part of this work seem to be really promising when thinking about creating a cost function that can accurately model our way of determining optimal travel paths. With the progress that was made as part of this project alone, we can now model optimal travel paths for simple maps such as a town map.

I believe that keeping the uphill and downhill constants as part of the cost function are key computing cost because users may have different limitations, apart from the different cases in which this cost function could be used to model.

FUTURE WORK

As part of future work, we could see this project be expanded to take into account more complex elements such as road conditions, maximum speed allowed, physics to observe how the cost could vary depending on the weight of the object that will be travelling through the node graph. Finally, additional work could be dedicated to take into account the physical shapes of the paths (i.e. curves).

Because the cost function is may still be “immature”, applying this to real world data could still be a challenging milestone. With the current progress of this project we could apply this to real world data where paths are consists only of straight lines and that is precisely one limitation that the project now possesses.

Another area in which this project can have additional work incorporated is when attempting to discover what are the appropriate values for uphill penalties and downhill bonuses for cases in which the user: rides a bicycle, motorcycle, or an electric vehicle. In the case of electric vehicles, downhills could very well yield significant downhill bonuses since most of these vehicles are designed to recharge their batteries when going downhills.

REFERENCES

[1] J. Bang-Jensen & G. Gutin, “Section 2.3.4: The Bellman-Ford-Moore algorithm”, in *Digraphs: Theory, Algorithms and Applications* (1st ed.), 2000. ISBN: 978-1-84800-997-4.

[2] Google Maps. (n.d.). Available: https://maps.google.com.

[3] O. Franzese & D. Davidson. (2011). *Effect of Weight and Roadway Grade on the Fuel Economy of Class-8 Freight Trucks, ORNL/TM-2011/471* [Online]. Available: https://info.ornl.gov/sites/publications/files/Pub33386.pdf.

[4] E. Wood, E. Burton, A. Duran & J. Gonder. (2014). “Contribution of Road Grade to the Energy Use of Modern Automobiles Across Large Datasets of Real-World Drive Cycles”, in *SAE World Congress 2014 Detroit, Michigan* [Online]. Available: https://www.nrel.gov/docs/fy14osti/61108.pdf .

[5] *Python Software Foundation*. (2019). [Computer Software]. Available: https://www.python.org.

[6] J. D. Hunter, “Matplotlib: A 2D Graphics Environment”, 2007. *21-9615/07/$25.00 © 2007 IEE.*

[7] A. A. Hagberg, D. A. Schult, P. J. Swart, G. Varoquaux, T. Vaughtn & J. Millmam, “Exploring network structure, dynamics, and function using NetworkX”, in *Proceedings of the 7th Python in Science Conference (SciPy2008) (Pasadena, CA USA),* Aug. 2008, pp. 11–15.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest & C. Stein, “Section 24.3: Dijkstra's algorithm”, *Introduction to Algorithms (2nd ed.)*, MIT Press and McGraw–Hill, 2001, pp. 595–601. ISBN: 0-262-03293-7.