Analysis of Geospatial Data for Computation of Best Path for Non-Motor Commutation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prithvi Vihari Ramesh Rao Missouri University of Science& Technology Rolla, Missouri USA prhy8@mst.edu |  | Manas Sangram Missouri University of Science& Technology Rolla, Missouri USA mspf7@mst.edu |  |  |

ABSTRACT

Historically, when human beings are travelling by foot the path of least resistance could mean avoiding mountain ranges, or following a stretch of terrain or avoiding large bodies of water. These paths are taken because they are easier to traverse and allow the traveler to expend the least amount of energy. In this paper we intend to explain how we could find the ideal path between two points keeping elevation in mind for individuals who either bike/hike. We also intend to include a functionality to show the calorific count to help make better health choices.

KEYWORDS

Optimal Path, Enhanced Dijkstra, PIG

ACM Reference format:

G. Gubbiotti, P. Malagò, S. Fin, S. Tacchi, L. Giovannini, D. Bisero, M. Madami, and G. Carlotti. 1997. SIG Proceedings Paper in word Format. In *Proceedings of ACM Woodstock conference, El Paso, Texas USA, July 1997 (WOODSTOCK’97)*, 4 pages.

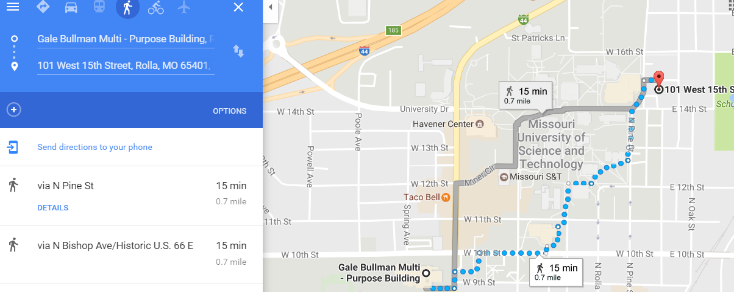
DOI: 10.1145/123 4

1 INTRODUCTION

The project deals with computing the best path or a preferable path between two points in a map. When we request a path between two points, we end-up with the shortest path between them. The shortest path considered is given with respect to the gradients and contours, which is mostly used by the bikers and hikers. The algorithm that is used in computing the shortest path are Dijkstra's algorithm. The basic problem with the shortest path from an origin to a destination could be going uphill which could be avoided just my taking a small detour. For an individual who want to choose the route that has least efforts, taking a detour is a preferred choice. In 1955, George Dantzig presented a conference paper that included the first formulation of the shortest path problem. His paper was subsequently published in Operations Research. Based upon that paper, Minty suggested a format for solving the shortest path problem using a network represented as a web of strings and knots, and suggested an algorithm to solve the shortest path problem. These are the various papers which deal with finding the shortest path until Dijkstra’s algorithm is made.

We intend to show the user the optimal path and calorific count of the current path to help them have a more informed session of biking/hiking.

In this paper we show our proposed algorithm and then explain how we intend to scale it to a real-life scenario with real time data. We also provide an experimental analysis of our implementation to give a gist of how the actual algorithm may run.



**Figure 1:** A snippet from google showing only the time and distance between two points.

2 RELATED WORK

A lot of related work has been done on the issue, but it wasn’t addressing the main issue considering the biking/hiking scenarios. A paper by W. Zeng, finding shortest paths on real road networks deals with the problem of identifying the shortest path along a road network, ranging from route guidance in a navigation system to solving spatial allocation problems. This paper also states that after a lot of research done on various types of algorithms for finding the shortest path they concluded with Dijkstra’s saying it is the most efficient algorithm among all.

Another paper by Yasha Pushak et al, Multiple-Path Selection for new Highway Alignments using Discrete Algorithms taught us how to calculate the cost between two points in a map and ways to increase the accuracy of the cost formulation. By properly modifying the algorithm according to the problem can reduce the computation time by 56% without compromising the quality of the results.

Finally, the thesis work of Audrey Waschura, A Collaborative Framework for Least Cost Caloric Paths with Constraints on High Resolution Data gave us insight on calculating the calorific values for paths with weight and elevation.

Even though this work has extensively helped us design the framework of our algorithm, they lacked the application of such elements to a scenario applicable to data generated by users daily.

3 PROBLEM STATEMENT

The problem involves analysis of Geospatial Data for Computation of Best Path for Non-Motor Commutation. It involves taking a certain source and destination and thereby output a definite path satisfying all the constraints like elevation, gross energy spent (force exerted) and provide the Best path for commutation along with calorific data specific to user.

4 PROPOSED ALGORITHM

4.1 Enhanced Dijkstra’s Algorithm

*4.1.1 Structure of Dijkstra’s Algorithm.* Dijkstra’s algorithm is the standard used for calculating the shortest path from one point to the other using weights. [8] Suppose you would like to find the shortest path between two [intersections](https://en.wikipedia.org/wiki/Intersection_(road)) on a city map: a starting point and a destination. Dijkstra's algorithm initially marks the distance (from the starting point) to every other intersection on the map with infinity. This is done not to imply there is an infinite distance, but to note that those intersections have not yet been visited; some variants of this method simply leave the intersections' distances unlabeled. Now, at each iteration, select the current intersection. For the first iteration, the current intersection will be the starting point, and the distance to it (the intersection's label) will be zero. For subsequent iterations (after the first), the current intersection will be the closest unvisited intersection to the starting point (this will be easy to find).

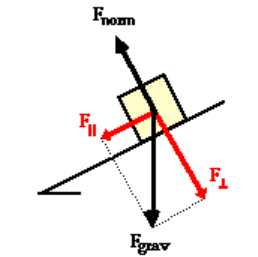
From the current intersection, update the distance to every unvisited intersection that is directly connected to it. This is done by determining the sum of the distance between an unvisited intersection and the value of the current intersection, and [relabeling](https://en.wikipedia.org/wiki/Graph_labeling) the unvisited intersection with this value (the sum), if it is less than its current value. In effect, the intersection is relabeled if the path to it through the current intersection is shorter than the previously known paths. To facilitate shortest path identification, in pencil, mark the road with an arrow pointing to the relabeled intersection if you label/relabel it, and erase all others pointing to it. After you have updated the distances to each [neighboring intersection](https://en.wikipedia.org/wiki/Neighbourhood_(graph_theory)), mark the current intersection as visited, and select the unvisited intersection with lowest distance (from the starting point) – or the lowest label—as the current intersection. Nodes marked as visited are labeled with the shortest path from the starting point to it and will not be revisited or returned to.

Continue this process of updating the neighboring intersections with the shortest distances, then marking the current intersection as visited and moving onto the closest unvisited intersection until you have marked the destination as visited. Once you have marked the destination as visited (as is the case with any visited intersection) you have determined the shortest path to it, from the starting point, and can trace your way back, following the arrows in reverse; in the algorithm's implementations, this is usually done (after the algorithm has reached the destination node) by following the nodes' parents from the destination node up to the starting node; that's why we keep also track of each node's parent. The computational complexity of Dijkstra’s shortest path algorithm is O (|E| + |V |) log|V |.

*4.1.2 Deviation from standard approach.* Although this weight has very often been referenced to distance we have used a different metric to assign weights to edges. Our metric involves the logic that the energy spent by an individual traversing along a certain path is directly proportional to the elevation of the path. In other words, energy spend on a path is much less if the path has a reduced elevation rather than a path that has a higher elevation.

4.2 Calculating the Edge-Weights

The program consists of a function which takes in the elevation and distance data. To this we apply our logic of the energy spent (force exerted) to travel across the path would be the metric to distinguish one path from another.



**Figure 2:** Force exerted by an object on an inclined slope.

Here,

E – Energy consumed

F – Force exerted

m – Mass

e – Relative Elevation

d – Distance between points

Based on the mass of the individual we could compute the energy he spends to conquer a path. This would in turn present a personalized path which is more convenient for him/her.

4.3 Calculating Distance

Each point is represented with latitude, longitude and elevation. Hence, we need to calculate the distance between a set of points by considering their latitude and longitude. Amarasinghe et al [6] have given us a detailed set of formulas for calculating the distance between them.

dLat= lat2−lat1;

dLon=lon2−lon1;

lat1= lat1;

lat2= lat2;

Here-

dLat – Difference in Latitudes

dLon – Difference in Longitudes

a, c - Constants

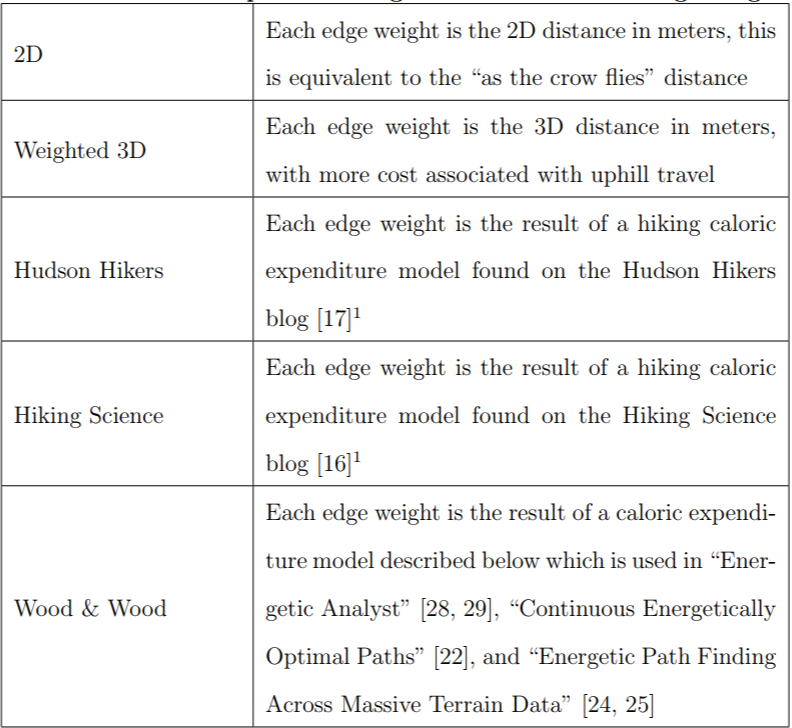
R – Radius of the Earth

d – Distance between points

4.4 Calculating the Calories Burnt

The model we are designing is based on the individual’s height, weight, distance travelled, slope and type of terrain. The equation below gives the metabolic rate in Watts for uphill travel.

and



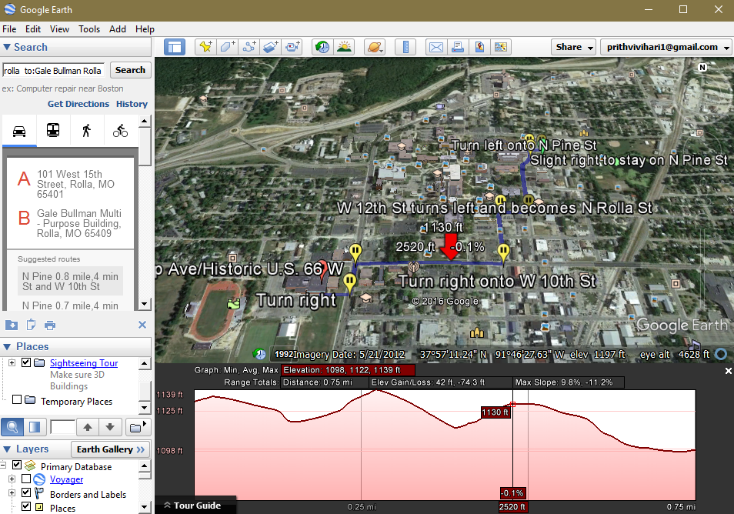
**Figure 3**: The five options for algorithms to calculate edge weights. [5]

By calculating the number of calories burnt, we are providing a way in which the individual can note the calories burnt.

4.5 Large Scale Data

When applying this information to real scale data we tend to have a large amount of data for a small map. To process such magnanimous quantities, we look towards Big Data tools such as Hadoop, Pig and so on.

We need to collect the latitude, longitude and elevation data. This information is available to us using the Google Maps API.



**Figure 4:** We can generate k shortest paths with the Google Earth API

However, we will have to gather the paths used by Bikers/Hikers which is available using the OpenMaps API. By amalgamating the data obtained from these API’s we could generate a credible dataset which can give us the relative path. However, this would be a challenge as the data provided by such API’s is not ordered and require lot more processing which is currently beyond the scope of the application. To address this problem, we have made use of a smaller dataset to test our application. This enabled us to have fasters results with the desired accuracy.

5 EXPERIMENTAL STUDY

In summary, we have performed both an experimental and theoretical study to get a set of paths. We made use of an experimental dataset which has several nodes representing source/destinations. The edges between the sources would be a representation of the weight. The weight is calculated by a function which would return the energy spent by an individual for taking the path between those two vertices.

5.1 Analysis of the Algorithm

The algorithm consists of six classes- namely Graph, Vector, Dijkstra, Metric and CSVInput.

The metrics file consists of the calculation of our edge weights which is the energy spent by the user based on his force exerted. The graph vector and Dijkstra classes would do the basic framework of the algorithm by representing each location (set of Latitude and Longitude) as a node/vertex.

The CSVInput is a class used for having us input our data using a CSV File and reading from it. This would ease our data collection which is refined into CSV format.

5.2 Results and Discussions

Consider an example graph given in Figure 5, where vertices represent locations and edges represent these weights as mentioned above. The arrows would represent the direction in which we are assigning the elevations between vertices.

We could apply our algorithm to this example to get an optimal path which is ideal for our application. We would notice that by considering the source node as ‘A’ and the destination node as ‘H’. The ideal route to consider is A->C->F->H. The reason for this choice is we could find the trade-off between the elevation and distance rather than the other routes which gave us a higher energy spent value. This route took a total of 566937 Units of energy compared to 602700 Units which is given by the next best route (A->B->F->H). This would allow the user to make use of multiple path options which could enable them to choose a difficult path for improving the burnt calorie count.

A close up of a map

Description generated with very high confidence

**Figure 5:** Sample Data for the algorithm

5.3 Stress Test

Even though the example may seem having a small dataset for ease of understanding, we have considered testing our algorithm with considerably larger datasets to check its complexity. We have tested our datasets with several records to check the execution time. This way we could check how the algorithm performs with huge datasets.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Set | No.of Vertices | Small Test Case (10 Test cases)  (ms) | Long Test Case (40 Test cases)  (ms) |
| Graph1 | 25 | 13 | 32 |
| Graph2 | 50 | 22 | 37 |
| Graph3 | 75 | 27 | 40 |
| Graph4 | 100 | 31 | 46 |

**Table 1**: Performance collected from several datasets

A close up of a map

Description generated with high confidence

**Figure 5:** Performance of the algorithm with datasets

Through this we can conclude that our algorithm gradually increases performance with load. However, using HDFS we could improve the performance further as we keep increasing the number of records.

6 CONCLUSION AND FUTURE ENHANCEMENTS

We have completed our basic framework planned for the first phase of our algorithm. This involved researching a lot into the dynamics of human commutations. We have made several realistic assumptions for our subject while assuming the physical dimensions and influencing factors. However, in the next phase of our project we would like to involve a huge dataset to optimize our algorithm to reduce any overhead.

We believe our work has thrown more light on the improving the influence of technology on hikers/bikers who would now be able to plan a very efficient path with least effort for their commute.

REFERENCES

|  |  |
| --- | --- |
| [1] | Pushak, Y., Hare, W., and Lucet, Y. (2016). Multiple-path selection for new highway alignments using discrete algorithms. European Journal of Operational Research, 248(2), 415-427. |
| [2] | Zeng, W., and Church, R. L. (2009). Finding shortest paths on real road networks: the case for A. International journal of geographical information science, 23(4), 531-543. |
| [3] | Mwemezi, J. J., and Huang, Y. (2011). Optimal facility location on spherical surfaces: algorithm and application. New York Science Journal, 4(7), 21-28. |
| [4] | Stubenschrott, M., Kogler, C., Matyus, T., and Seer, S. (2014). A dynamic pedestrian route choice model validated in a high-density subway station. Transportation Research Procedia, 2, 376-384. |
| [5] | Waschura, A. (2017). A COLLABORATIVE WEB APPLICATION FRAMEWORK FOR LEAST COST CALORIC PATHS WITH CONSTRAINTS ON HIGH RESOLUTION DATA. |
| [6] | Amarasinghe, S., Kodikara, N. D., and Sandaruwan, D. (2014, December). Location estimation in a maritime environment using a monocular camera. In Advances in ICT for Emerging Regions (ICTer), 2014 International Conference on (pp. 21-28). IEEE. |
| [7] | *https://nextcity.org/daily/entry/pedestrian-open-data-safest-shortest-walk-map* |
| [8] | https://en.wikipedia.org/wiki/Dijkstra%27s\_algorithm#Description |