Finding Your Inner Blaha: GPS Mapping of the Swarthmore Campus

Matt Singleton and Bronwyn Woods
{msingle1,bwoods1}@cs.swarthmore.edu

Abstract

Swarthmore College is a largely unmapped, dangerous region of southeasters Pennsylvania. Or rather, we treat it as such for the purposes of this paper. We present an interactive tool for calculating the shortest path between two points on the Swarthmore campus. We develop our tool using a combination of GPS technology and knowledge of Swarthmore's buildings. We allow users to specify a *Blaha factor*, which scales the weights of indoor paths, causing them to be treated as shorter or longer than their real lengths in the shortest path calculations. In this way, users can express a preference for traveling primarily indoors or outdoors, depending on personal preference and weather conditions.

1 Introduction

People familiar with a place often have strong intuitions about the most efficient ways of traveling between locations they frequent. However, people's intuitions are sometimes in disagreement. Additionally, special circumstances such as extraordinarily nice or foul weather may influence a person's preference for possible routes. We present a tool for identifying the shortest path between two points on the Swarthmore College campus, allowing for preferences for indoor or outdoor paths.

We mapped the outdoor paths on the campus using GPS technology and estimated the indoor paths based on our knowledge of the buildings. Using a combination of manual and algorithmic techniques, we transformed our raw point data into a graph on which we perform shortest path routing using Dijkstra's algorithm. We allow indoor and outdoor paths to be weighted differently, effectively discounting or penalizing the distance traveled inside.

Our tool is presented as an interactive GUI that allows the user to select points on a map of Swarthmore's campus. The tool graphically displays the shortest path according to the value of the *Blaha factor*, or weighting of the indoor paths, set by the user.

2 Related Work

2.1 Global Positioning System

The NAVSTAR Global Positioning System (GPS) provides precise information about location by using signals transmitted by 24 satellites in Earth's orbit. Originally designed for exclusive military use, the system was opened for civilian use as it became fully operational in the early 1990s. GPS satellites transmit ranging signals which encode information about the satellite's location at the time the signal was sent. By combining information received from several satellites, this signal allows GPS receivers to calculate their 3D location on Earth's surface. The accuracy of locations determined by GPS can range dramatically depending on the quality of the GPS receiver. Commercial quality GPS receiver units have typical errors of between 10m to 30m, while more expensive systems can reach an accuracy at the subcentimeter level (US , 2003).

Many factors contribute to the overall accuracy of measurements taken using GPS. These include, in addition to the quality of the receiver, atmospheric conditions, the environment of the user and the position of the GPS satellites relative to the user. GPS measurements with commercial receivers can only be performed outdoors and can be disrupted by dense tree cover or other large obstacles (US , 2003).

2.2 Dijkstra's Algorithm

Dijkstra (1959) describes two algorithms for finding shortest paths on a graph, one for finding the minimum spanning tree and the other for finding the shortest path between two nodes. For the purposes of this project, we were concerned only with the latter. The operation of this algorithm is discussed in section 3.3.

3 Methods

3.1 Data Collection

We collected raw data about the paths on the Swarthmore College campus using a Garmin GPSMap70 GPS unit. We marked paths by recording points manually at regular intervals. Manually recording points allowed us to determine the frequency with which we recorded points and to ensure that we recorded points at the intersections and endpoints of paths. Each point we recorded was given a unique ID, allowing us to keep track of which points started and ended any given path. In total, we collected 910 points. In addition to the data points delimiting the paths, we recorded individual points representing the doors into the campus buildings. The self-reported accuracy of the GPS unit averaged around 10m for all of our data collection.

We recorded our data using the UTM coordinate system. The UTM system breaks the globe into zones, or bands running north to south. A location is defined by its zone, an easting and a northing. The easting represents the distance from the edge of the zone, while the northing gives the distance from the equator.

3.2 Processing Techniques

Once we had gathered our raw data, we needed to make a number of additions and changes to prepare it for screen display and path computation.

3.2.1 Hand Cleanup (First Pass)

Our raw GPS data was surprisingly good, but it still contained a number of erroneous data points. At this point we had a preliminary GUI that allowed us to view the data as a collection of numbered points and lines. Given this view, it was relatively easy to identify the erroneous data points visually and then remove them by hand.

In addition to the erroneous data points, the raw GPS data is presented as one unbroken line. The result is long segments connecting the end of one path to the beginning of another. We divided the data into the individual paths again by visual examination of the data.

3.2.2 Line Intersection

Our next task was to find the points at which the paths intersected. Because the number of points in our data set was small, we decided to do this using a brute-force algorithm. Each path is made up of a number of straight line-segments, so we simply check each segment for intersections with every other segment. If an intersection is found, that point is added to both paths. This operation is $O(n^2)$ where n is the number of line segments.

3.2.3 Hand Cleanup (Second Pass)

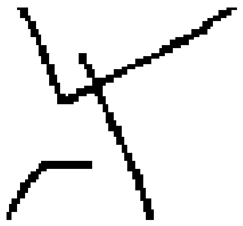


Figure 1: In the raw path data, some paths end a bit too soon, while others end a bit too late.

While GPS did a very good job of gathering data with good relative positioning (straight paths are straight and curved paths curve where they are supposed to), it did a much poorer job at absolute positioning. As a result, paths often end slightly before or slightly after they should (see figure 1). We used our preliminary GUI to visually identify where these problem areas were. We then added or removed points from the data by hand, as appropriate.

3.2.4 Adding Doors and Indoor Paths

As noted above, we gathered individual points marking the entrances to buildings in addition to our path data. Unfortunately, due to the absolute positioning problems with the GPS data, many of these points were significantly wrong. Based on our knowledge of the buildings on campus, we were able to identify which doors points were worth keeping and which we needed to be adjusted manually. GPS does not work inside, so we needed to add indoor paths by hand. We approximated these based on our knowledge of the buildings and the locations of the doors.

3.2.5 Creating a Graph

Finally, with all the paths and intersections in place, we needed to convert our data into a graph that that we could use to compute shortest paths using Dijkstra's algorithm. As our data was then, a single path could contain multiple vertices and span multiple edges. We needed to segment it so that each path corresponded to one edge in the graph, and each endpoint corresponded to a vertex in the graph. With the data in this format, it was relatively easy to build the graph structure in one pass through the data.

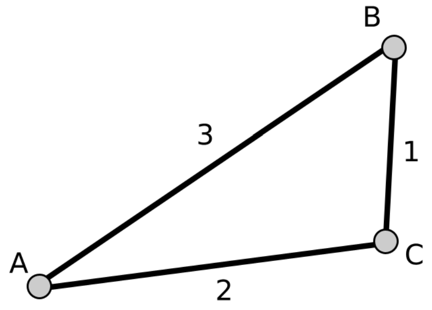


Figure 2: A simple graph.

Our graph is implemented as Python dictionary (essentially a hash table) where the keys are vertex IDs for every vertex in the graph and the value is a dictionary where the keys are vertex IDs for all connected vertices and the value is the weight of the corresponding edge. Because hash table lookups can be done in constant time, building the graph is an $O(n)$ operation where n is the number of paths.

```
{‘A’: { ‘B’: 3, ‘C’: 2},
 ‘B’: { ‘A’: 3, ‘C’: 1},
 ‘C’: { ‘A’: 2, ‘B’: 1}}
```

Figure 3: An example of our graph representation describing the graph displayed in figure 2

3.3 Computing the Path

Now that we have constructed our graph, we can compute the shortest path between any two vertices using Dijkstra's algorithm (Dijkstra, 1959). Dijkstra's algorithm, left to its own devices, will compute the entire minimum spanning tree of a graph, starting at a given node. Since all we care about is a single shortest path, we can stop computation as soon as our destination vertex is added to the tree.

The algorithm is simple, and can be easily implemented in Python. To begin, the algorithm sets the distance to the start vertex as 0 and the distance to all other vertices as ∞. Initially, the tree contains only the start vertex. The algorithm proceeds by incrementally adding adjacent vertices to the tree until every reachable vertex is added. The next vertex to be added to the tree is always the vertex whose addition will minimize the length of the longest path. Refer to figure 4 for an example.

4 Results

4.1 Raw Data

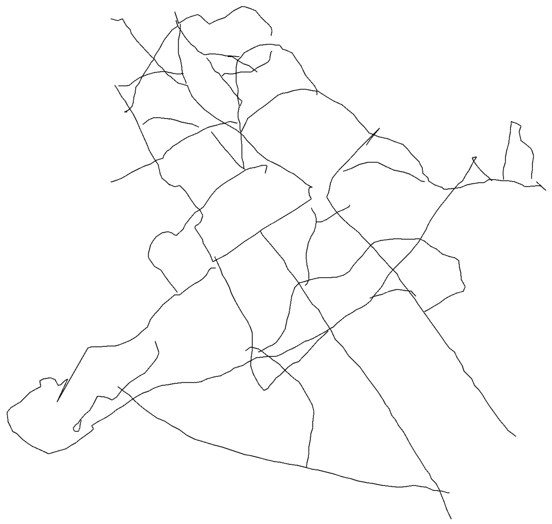


Figure 5: The raw path data from the GPS unit.

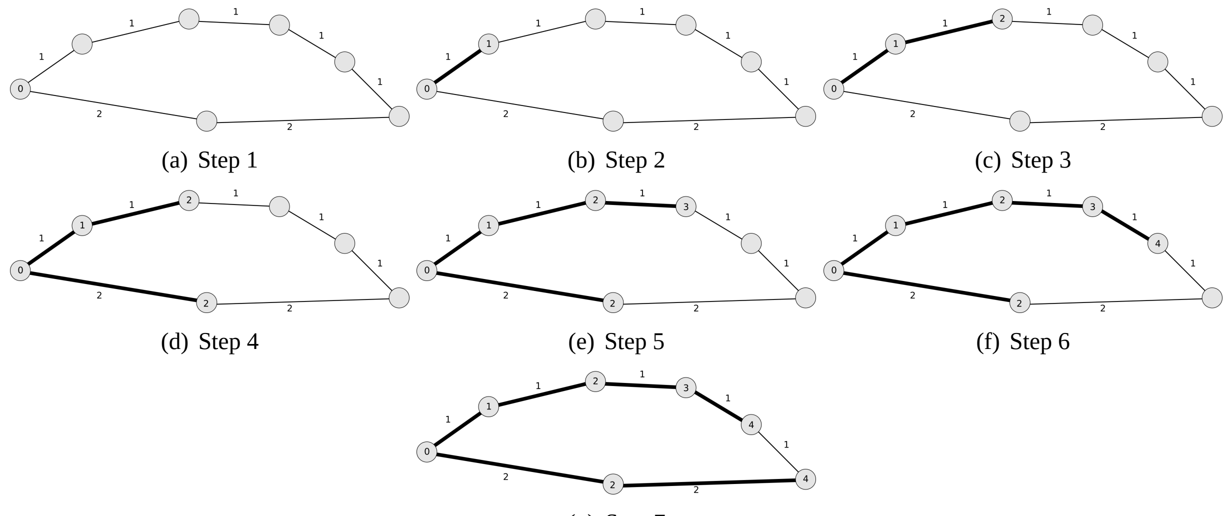

(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4 (e) Step 5 (f) Step 6 (g) Step 7

Figure 4: A trivial example of the execution of Dijkstra's algorithm.

Figure 5 shows our raw path data. The map is clearly recognizeable as the Swarthmore campus, but there are many flaws to be corrected. We can see several immediate problems with this data. Some paths that should intersect fall short and do not meet. Other lines do intersect, but extend past the intersection when they should end. Lastly, some points are clearly inaccurate by a large margin causing spikes in a few of the paths.

4.2 GUI

Once we had all of the underlying structure built, we created a GUI to concisely present all of the data and provide an easy method for getting user input. The GUI is comprised of three distinct areas.

Map canvas The largest and most important part of the GUI is the map canvas. This is where the map is displayed and the user can select the start and end points of their desired path. Once two points are selected, the shortest path is calculated based on the current Blaha factor and drawn own the map.

Status bar Along the bottom of the window, the status bar displays the current x- and y-coordinates of the mouse pointer as well as the current Blaha factor.

Input area To the right of the Map canvas, the input area allows the user to change the Blaha factor and reset the map.

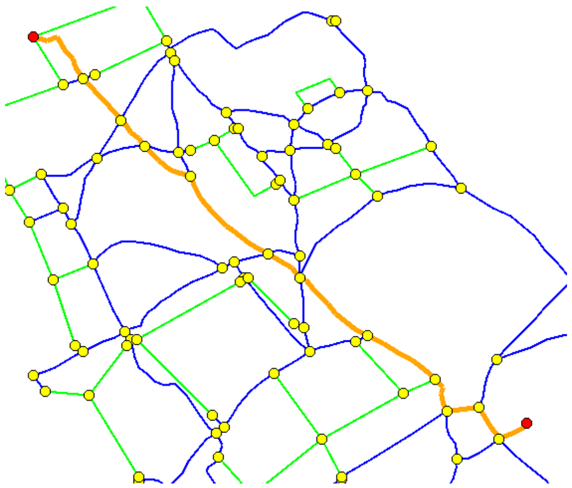
5 Discussion

Our final product is an interactive tool for shortest path routing on the Swarthmore campus. Though it might seem that the tool would be superfluous given the familiarity of students with the campus, anecdotal evidence shows that some shortest paths, especially with modified Blaha factors, are surprising even to Swarthmore students.

Though our map is created from GPS data, there are several possible sources of inaccuracy which might affect the shortest path calculations. For one thing, the lengths of the indoor paths are only estimated, and do not take into account stairs that must be climbed or doors that must be opened. We also do not consider elevation for the outdoor paths, though this is unlikely to make a significant difference.

There is a potential to expand our tool in a variety of directions. For instance, we could expand our map to cover a greater portion of Swarthmore College and the surrounding areas. We could allow the user to specify which buildings he could not pass through due, for instance, to not having a key. We might also be able to improve the accuracy of the door data points by sampling several points at the doors and averaging their positions.

Though the methods we used for creating a searchable map of the Swarthmore campus worked


(a) The shortest path with the Blaha factor set to 1.

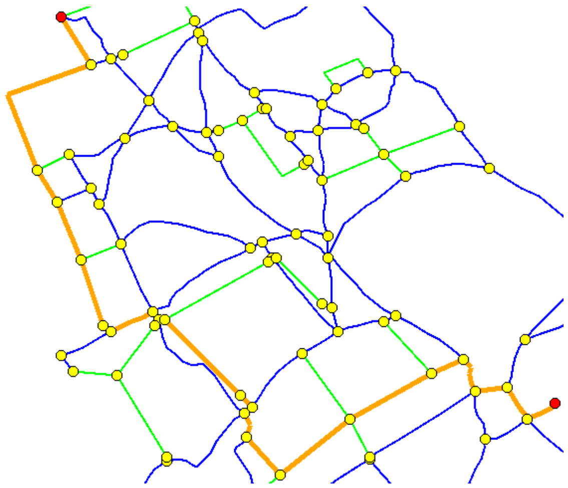

(b) The shortest path with the Blaha factor set to 0.1

Figure 6: The final display given by the GUI, showing the shortest paths from the Science Center to the McCabe Library with the Blaha factor set to 1 and 0.1.

for this task, they would not be scalable. This is due to the large amount of hand cleanup involved. However, the amount of error present in the GPS data we obtained necessitates this hand clean-up. It seems that the task of creating a map of the type we present for a larger area would require a different approach.

6 Conclusion

We present in this paper an interactive path finding tool for the Swarthmore College campus. The tool allows for differential treatment of indoor and outdoor paths, allowing the use to specify a preference for travel. Though the methods that we used for creating the map and searchable graph would not be scalable to larger maps, the techniques were effective for our task. Our interactive GUI allows the user to discover efficient paths which are occasionally surpsising even to individuals familiar with the area.

References

E. W. Dijkstra. 1959. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271.

US Army Corps of Engineers, 2003. *Engineering and Design - Navstar Global Positioning System Surveying*, July.