The Road Not Taken: Creating a Path-Finding Tool Using Consumer-Grade GPS Equipment

Allison Barlow
ajb@sccs.swarthmore.edu

Lucas Sanders
lsanders@sccs.swarthmore.edu

Abstract

We automate the organization of GPS data to create a visibility map with correctly connected intersections. For our example case study, we use GPS data collected on the Swarthmore College campus. After the data is automatically organized and cleaned, a user is able to select points on a visual map to search for an optimal path between those points.

1 The Problem of Navigation

Swarthmore College, set in the beautiful Scott Arboretum, has many scenic routes and winding paths between the lovely stone buildings. This serene setting is wonderful for long walks, but can be troublesome in trying to navigate from one's dorm to a classroom when one is half-awake and running behind schedule. In order to assist the poor, sleep-deprived students of Swarthmore College, we create an interactive map of the campus to help users discover the appropriate paths between the various buildings on campus. In doing so, we develop a set of tools that can easily create similar systems for data collected in other locations.

1.1 Previous Work

Another group worked on a GPS path-planning project for last year's Senior Conference (Singleton and Woods, 2007). Because their data required quite a bit of manual editing, their system would not easily scale to larger data sets. We also noticed that their interface was difficult to understand immediately. Taking these challenges into account, we create a scalable, user-friendly path planning tool.

We used a large amount of previous research to build our tools. Most notably, Dijkstra's algorithm (Dijkstra, 1959; de Berg et al., 1997) and the more general A* search algorithm (Hart et al., 1968) have been developed to find appropriate paths through our spatial graph. We also implement an improved version of the Douglas-Peucker line simplification algorithm (Hart et al., 1968).

1.2 Approaches to Path Planning

Path planning is essentially a least-cost graph searching problem, a task for which several algorithmic variations have been developed. In the past, Dijkstra's algorithm has been widely used for this task, but we implement the A* search algorithm, which is a generalization of Dijkstra's approach that is both complete and optimally efficient. In typical situations, A* performs slightly faster than Dijkstra's algorithm because it uses a heuristic to help decide which search paths are most promising. This process minimizes the size of the subgraph to be explored.

2 Implementation

This project consists of three main parts: data collection and processing, path planning, and UI development. The first two parts center on problem solving using computational geometry while the third provides easy access to the results.

2.1 Data Collection and Processing

We use self-collected GPS data for our finished mapping system. Swarthmore College kindly provided their survey plans for use with this project, but due to a lack of time, we were not able to complete an integration of this data into our user interface. We prioritized our work with the GPS data, even though that data is not as precise, because we wanted our system to be useful in creating mapping systems in other situations where no such detailed survey plans have been prepared.

2.1.1 GPS Data Collection

Our main dataset is GPS tracking data recorded by walking the paths of Swarthmore's campus with a consumer-grade GPS receiver, the Garmin GPSmap 60CSx. For consistency, we carried the GPS receiver at about waist-height and walked on the center of each footpath, recording tracks of where we had walked and marking a waypoint each time we encountered a path intersection. We used GPSBabel to transfer data from the Garmin unit to the GPX format, an XML-based file format that includes latitude, longitude, elevation, and timestamp data. We collected GPS data for virtually all exterior paved footpaths on the Swarthmore campus; in contrast to last year's project, we did not record indoor footpaths because of the inability to collect GPS data indoors with inexpensive equipment and our desire to avoid manual editing of the collected data (Singleton and Woods, 2007).

2.1.2 GPS Data Processing

Our pre-processing algorithms find clusters of paths with geographically nearby endpoints. We assume that these paths intersect at a common point, so we reassign the endpoint coordinates of all paths in a cluster to the arithmetic mean of the endpoint coordinates in that cluster. While this approach is fairly naive, we found that it works quite well in practice with the relatively coarse resolution captured by consumer-grade GPS equipment.

We experimented with simplifying the data as a final processing step, hoping to reduce the computational power needed for both the user interface display and our path-finding algorithms. We use the Douglas-Peucker line simplification algorithm to do so. This, however, was not particularly helpful for this particular application in part because of the relatively infrequent position sampling provided by our equipment. Also, we constructed our search graph on adjacent path intersections, ignoring the complexity of the path segment between those intersections, which is an even more efficient simplification for searching purposes. It is important, however, to explore the practicability of such simplification techniques for use with larger, higher-precision datasets.

At the end of these pre-processing steps, the resulting data is saved as latitude, longitude, and elevation tuples for each meaningful point of each path segment. Figure 1 shows a clear difference between our raw data and the same data after being processed with our programs.

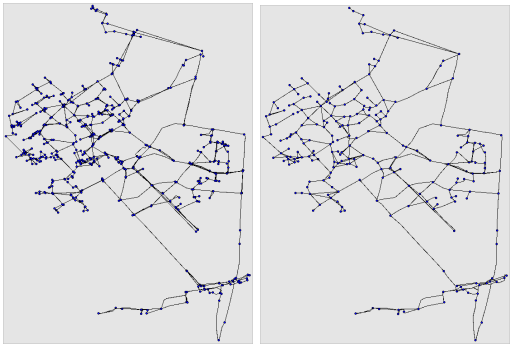

Figure 1: Before and after data processing cleanup on a subset of our GPS data

Figure 2: GIS overlay of our GPS data and the survey drawings

2.1.3 CAD Survey Drawings

Thanks to the maintenance staff at Swarthmore College, we also acquired survey-grade mapping data for the Swarthmore campus in AutoCAD format. We convert the several layers of data from these

drawings to the Shapefile format and simplify the data using the Douglas-Peucker algorithm. The final output data uses the same simplified data format described at the end of our GPS data pre-processing algorithms. We had hoped to use information from these drawings as a base layer to help orient users to the paths as displayed in our user interface, but ran out of time before we could complete that effort. Figure 2, however, shows an overlay of these datasets that has been prepared with a GIS package.

2.2 Path Planning

Path planning using the A* search algorithm is fairly straightforward. We create a graph of the connections between path intersections; each edge in the search graph is weighted with the path distance between its endpoint vertices. Then, we apply the search algorithm to this graph, finding the shortest path between the two selected nodes.

A* uses a heuristic function: the sum of distance to get to the current node plus the cost to get to the next node. Using this function, A* chooses which nodes to visit based on the routes that appear to be shortest. The partial paths it has explored are stored in a priority queue, and when its cheapest path becomes relatively expensive, other potentially shorter paths are explored. When a path is found to reach the goal with a lower heuristic value than any other (partial) path in the queue, A* has found the shortest path. A* is both complete and optimal, making it an obvious choice for path finding.

We choose to implement the A* search algorithm because its implementation is not significantly more difficult than implementing Dijkstra's algorithm. Given this, the heuristic in A* provides a slight performance improvement in typical situations.

2.3 UI Development

We provide a python program for visualizing the paths in our search space and the optimized paths between selected points. Search endpoints are selected simply by clicking on intersections in our search graph, and the optimal path between those points is then highlighted on the map. A screenshot of this interface is provided in Figure 3, where the first point selected is green, the second point selected is red, and the shortest path between them is shown in blue.

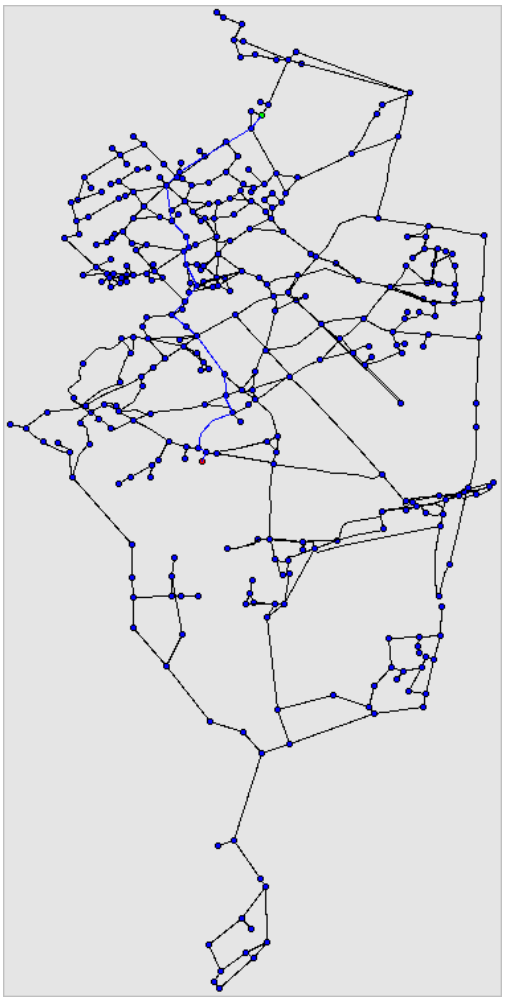

Figure 3: Screenshot of our UI with a path selected

We did not have time to complete further UI development work, although this was not the largest focus of our project. The UI would be even more intuitive, however, if the paths were overlaid on a map that shows other landmarks, such as buildings.

3 Results & Conclusions

Our system appears to be robust and to provide accurate results for a user's queries. The interface can be quickly and simply explained, but is not yet intuitive enough to be used by most individuals without a brief explanation.

We are quite happy with the way that our system meets the goals set out at the beginning, especially with regards to scalability. For example, we developed our processing routines with a subset of our collected GPS data, then were able to add the remainder of our data with less than five minutes of additional work. We are confident that our system would scale well even to much larger datasets than the ones used in this project, and that this represents a good approach to developing such path-finding systems much more quickly and cheaply than has been previously possible.

References

M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. 1997. *Computational Geometry: Algorithms and Applications*. Springer.

E. W. Dijkstra. 1959. A Note on Two Problems in Connexion with Graphs.

P. E. Hart, N. J. Nilsson, and B. Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths.

Matt Singleton and Bronwyn Woods. 2007. Finding Your Inner Blaha: GPS Mapping of the Swarthmore Campus.