Where is the Worst Place in New York City to Have a Heart Attack?

Miranda Chaiken, mrc4@williams.edu, 19mrc4
Nick Post, nbp1@williams.edu, 18nbp1

Introduction

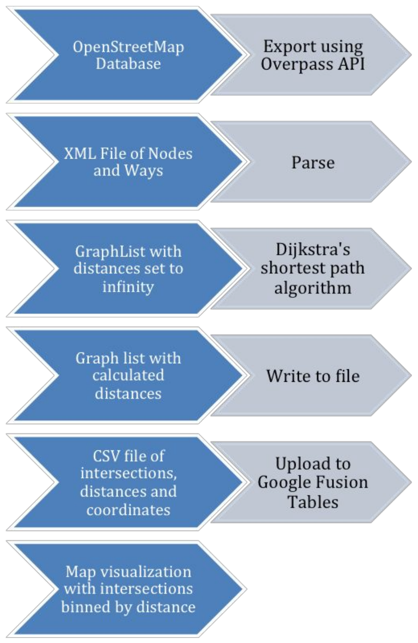
The majority of the world's cities were not carefully designed to optimally serve their residents. Instead, cities developed organically over time with new infrastructure built on top of preexisting networks. As a result, certain locations in large cities may be underserved by vital services such as emergency rooms and fire departments. In this paper, we examine the availability of acute medical care in New York City. We assume that a critical medical incident may be more or less dangerous depending on where it occurs in the city and as a result we hypothesize the degree of danger may not be evenly distributed but instead concentrated in certain areas. To provide empirical support for this hypothesis we apply computational techniques on a graph structure to calculate the distance to an acute care hospital from every street intersection in New York City.

Methodology

Approach and Design

To calculate the distance from every street intersection in NYC to an acute care hospital we reframed our question in the language of graph theory. The streets of NYC can be described by a graph with vertices representing intersections and edges representing streets between them. We assigned to each edge a weight according to the distance (in meters) between the vertices it connects in the actual road network. Additionally, we made the simplifying assumption that the graph is undirected because in an emergency we assume an ambulance, or other emergency vehicle, can travel either way on any road, regardless of traffic laws. To find the shortest path between a set of source vertices and every other vertex in a graph we used a slightly modified version of Dijkstra's shortest path algorithm with all 11 acute care hospitals in NYC as source vertices.

The general design and data flow through our program is illustrated by the flow chart on the right. A brief overview of significant implementation decisions made for key components will be provided in the next section.


OpenStreetMap Database
Export using Overpass API
XML File of Nodes and Ways
Parse
GraphList with distances set to infinity
Dijkstra's shortest path algorithm
Graph list with calculated distances
Write to file
CSV file of intersections, distances and coordinates
Upload to Google Fusion Tables
Map visualization with intersections binned by distance

Key Implementation Decisions

Programming Language

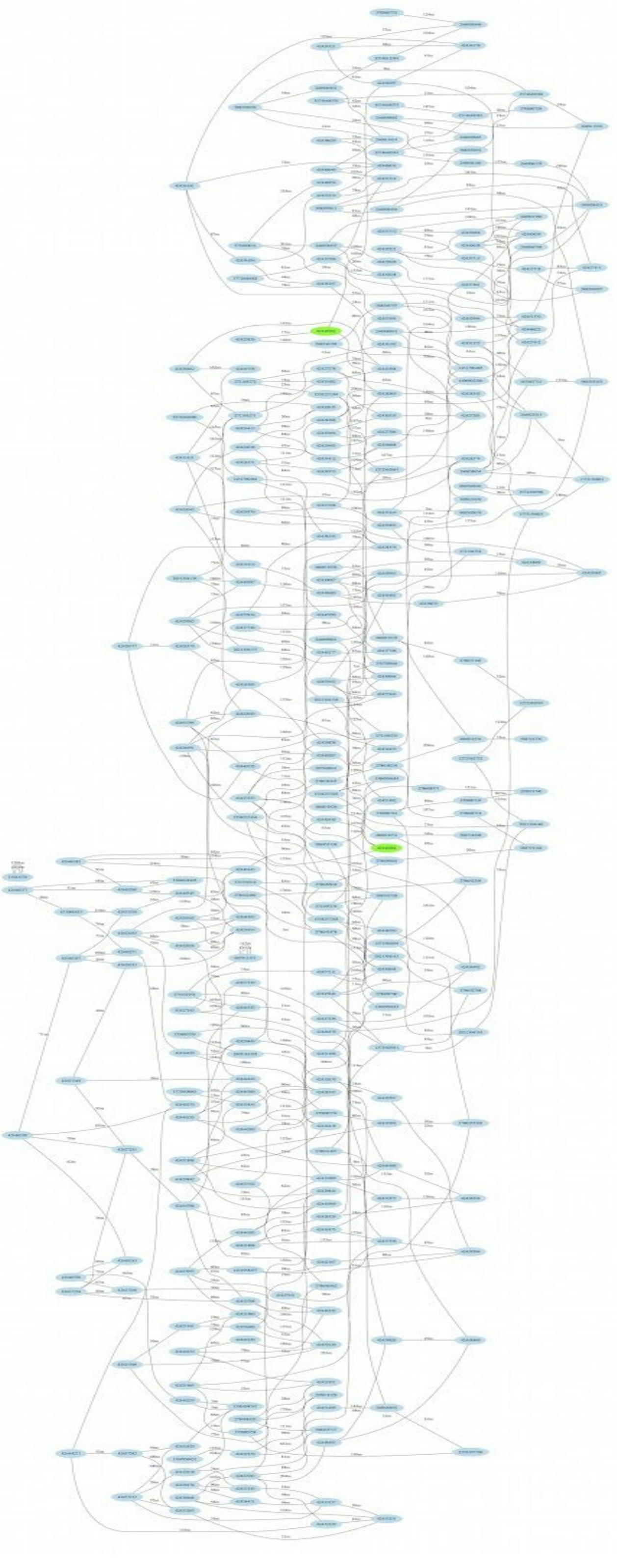
We chose to implement our program in Python for a few key reasons. First was ease of programmer use. Second, the built in XML parser made loading the data sets from OpenStreetMap easier and the availability of libraries such as graphviz made visualizing data simple. We weighed these benefits against the advantages of C++, namely speed of processing, but C++ is faster only by a constant factor. Since our program is answering a scientific question and simply needs to produce an answer, not be instantaneously responsive, we determined the ease of use and readability of Python to be worth its slower run time.

Data Source and Parsing

For our calculations we needed to obtain data containing every street intersection in NYC and the distances between adjacent intersections. OpenStreetMap is an open source mapping system with data available for export to XML using the Overpass API. The data, however, defines streets as lists of nodes consisting of solely latitude and longitude coordinates and includes many irrelevant features.The low-level nature of the data and sheer volume of content leads to a lot of data; for NYC our data file is 1.28 GB. As a result, we needed to intelligently parse the data to ensure a reasonable running time and to minimize memory usage. To accomplish this we constructed our parser using the Simple API for XML (SAX) rather than the Document Object Model (DOM) approach to parsing XML to avoid the DOM's need to load the entire file into memory. SAX is an event driven parsing technique where a parser streams through the XML file, invoking callbacks when certain events such as the start or end of an XML element occur. With SAX we were able to parse the XML data in 1.5 passes through the file: once to identify the necessary data–the streets and the nodes defining them–and a second half-pass to extract the location data from only previously identified relevant nodes. This approach loads the minimum amount of data needed to calculate edge weights and build our graph.

Graph Implementation

Even though we may have been dealing with a map in the real world, this problem from a computer science standpoint is all about graphs. We implemented the graph using a Graph List implementation with edge lists stored at each vertex instead of an adjacency matrix because of the sparseness of the graph. The average intersection is connected to approximately four other nodes, meaning the edge lists for nodes are relatively small compared to 50,000 nodes in the total graph. A Graph Matrix implementation would consume space for a possible edge between every pair of vertices for an $O(V^2)$ space bound. The amount of unnecessary space consumed by a Graph Matrix implementation would be problematic as the size of the data set increases. We did not implement any remove methods for the graph under the assumption that once intersections and streets are loaded into the graph they will not change. Below is a visualization of a portion of the abstracted graph of the city, with source nodes highlighted in green.



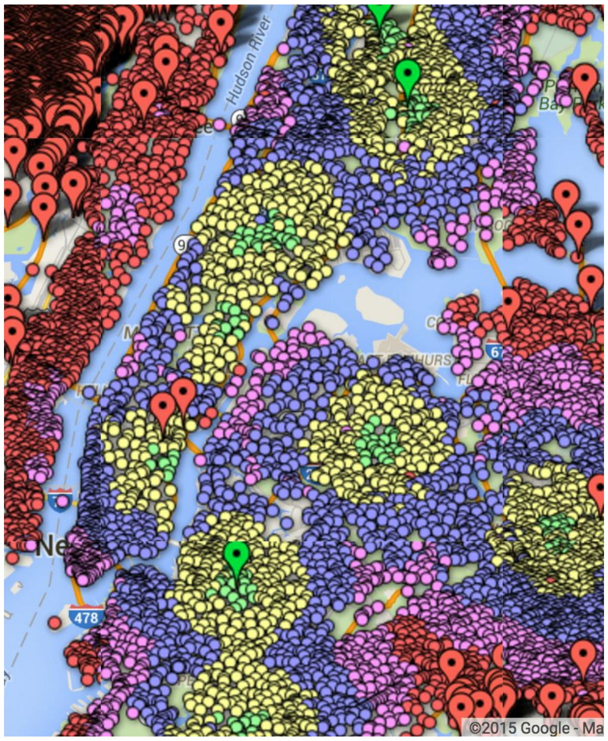
Dijkstra's Shortest Path

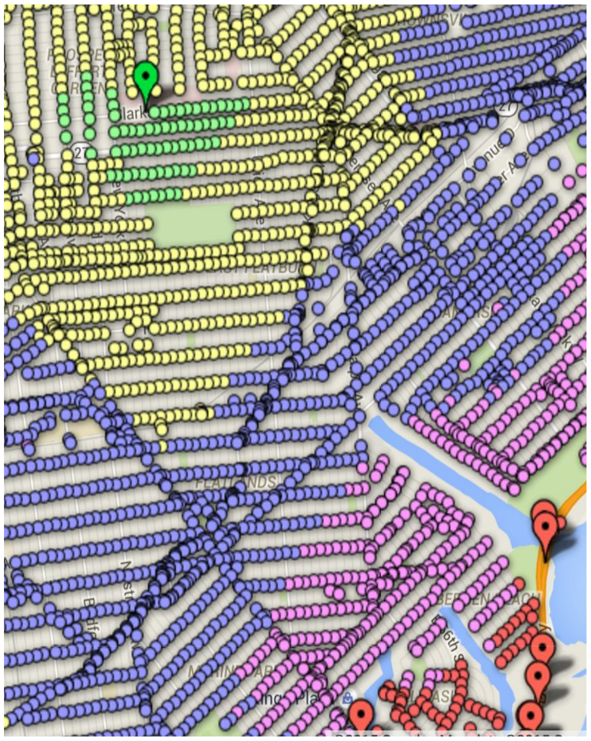
To calculate the shortest distance from every intersection in NYC to a hospital we use an implementation of Dijkstra's shortest path algorithm. Dijkstra's shortest path algorithm is a variant of breadth-first search that visits the closest unvisited vertex to the source–determined by storing the unvisited vertices in a min priority queue–at each iteration such that every node is visited exactly once and assigned its final distance when visited. To address our question about the minimum distance from every intersection to any of a set of sources (11 for the case of NYC acute care hospitals), we modified Dijkstra's algorithm such that it visits the closest vertex to any source at each iteration. With our modification the algorithm still visits each node exactly once, but the search stops advancing in a given direction when it reaches a node in that direction closer to another source. We accomplished this modification by creating a min priority queue for each source vertex containing the unvisited neighbors with priority based on distance from that source. We then placed each of these priority queues (by reference, as is the default in Python, to avoid copying large data structures) into another priority queue with priority according to the front vertex in each source queue's distance. At each iteration we pop from the queue of source queues and then pop from that source queue to obtain the closest vertex to any source, perform the standard Dijkstra's algorithm on that vertex and its neighbors, then push the source queue back into the larger queue with a new priority determined by its new minimum vertex distance.

Visualizing the Results

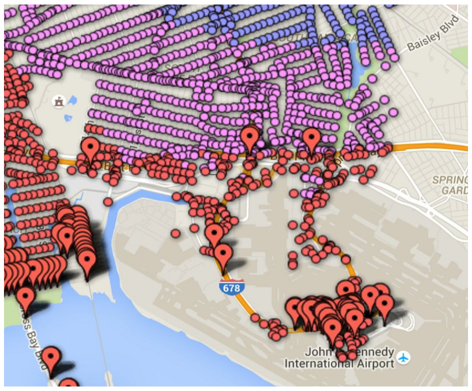
Our program produces a CSV file providing the ID, distance from a hospital, latitude, and longitude for every vertex representing a street intersection in our graph. We chose to output our data as CSV to allow easy interfacing to available data analysis tools. Specifically, we used Google Fusion Tables, an experimental Google app to analyze our data. As a Google app, Fusion Tables is directly tied into Google Maps. From our Fusion Tables spreadsheet we were able to bin our data by distance from a hospital and generate a map with markers at each intersection color-coded by distance from a hospital.

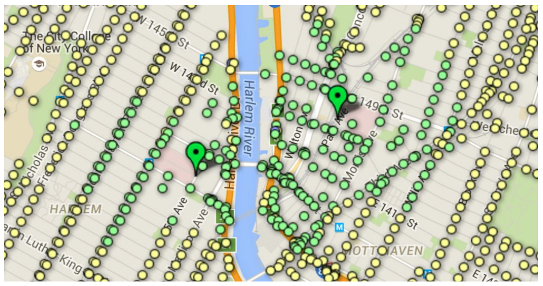
Results





The final product is a histogram map with color-coded markers representing proximity to the nearest acute care center. The areas in green are under 1,000 meters away, yellow are less than 3,000 meters, purple 5,000 meters, pink are up to 7,000 meters, red dots are up to 10,000, and red pins are any greater distance. As you can see, the acute care centers in the city turn out to be reasonably evenly distributed, although there are large underserved areas in Brooklyn and Queens. It is important to note that some of the "underserved" areas on the outskirts of the map may have hospitals nearby not within the confines of New York City. Interestingly, JFK Airport appears in the red zone of our map, indicating isolation from an acute care center. This is concerning given the potential for catastrophic accidents at an airport. However, because JFK lies on the outskirts of NYC, there could be a closer hospital somewhere on Long Island, mitigating the apparent risk our results show.





Another interesting observation is that while most acute care centers are evenly distributed, there are two nearly adjacent ones on either side of the Harlem River in Harlem and the Bronx. While this may be necessary as a result of difficulties crossing the river, it is possible that some of these resources might be put to better use in neighborhoods like Bay Ridge that are severely underserved.

We must note that a few nodes appear as red pins in otherwise yellow or green areas, a result of imperfect open source data. It is also important to bear in mind that these locations only take acute care centers into account, and there may be other medical facilities in closer proximity to underserved areas. We chose to focus on acute care centers only because they are necessary to treat the most dangerous medical issues.

Link to Google Fusion Table spreadsheet containing data for interactive analysis:

https://www.google.com/fusiontables/DataSource?docid=1BHoiQPmFgnEYxjTZhbcHmizQYZAAZFY_GnJENi5v

Because of the generality of the algorithms used, they could be applied to any region available on OpenStreetMap, and any utility in that region. Areas for future exploration might include firehouses or police stations in other cities or dining halls on the Williams College campus.

References and Resources

Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". *Numerische Mathematik* **1**: 269–271

OpenStreetMap http://www.openstreetmap.org/

Google Fusion Tables https://sites.google.com/site/fusiontablestalks/stories

List of Acute Care Centers
https://data.cityofnewyork.us/Health/Health-and-Hospitals-Corporation-HHC-Facilities/f7b6-v6v3