MOO-pQuest: Crime-Fighting CS51 Project

Derek Choi, Ramsey Fahs, Alice Tang, Jessica Zhu

Video

http://youtu.be/D0JxFq355GI

Overview

We implemented an algorithm for combining crime data with routing software to allow for safety-adjusted walking directions of Manhattan.

How it works

Our project uses Djiktra's algorithm, a method of finding the shortest path between points. It operates by representing each intersection as a node. Streets are organized into ways, or series of nodes. We ripple outwards from the starting node, visiting neighboring nodes until we find the best (i.e. the shortest) path to the destination.

Using historical crime data provided by the City of New York, we modified the weights of certain roads and streets to make them more or less desirable to walk on. Given that the algorithm attempts to minimize distance, we effectively manipulated the algorithm to "trick" it into thinking that the more dangerous roads were longer.

Usage Instructions (also in `code/README.md`)

Testing Notes

OSM data and PyRoute are very inefficient. We have provided three data files: `small.osm`, `smallway.osm`, and `manway.osm`. The first of these, `smallway.osm`, is the processed version of `small.osm`, and these two files only contain 100th-120th streets in Manhattan. All of Manhattan with crime parameters is in `manway.osm`. We did not give you the original Manhattan file because (a) it is very large and (b) it took a very long time to process.

Data Processing

From within the `MOO-pQuest/code` folder, run `python process.py [path/to/osm/file] [path/to/new/osm/file]`. The same effect can be achieved in multiple steps by using `smalldict.py`, `crimetonode.py`, and `nodetoway.py` (in that order) using the command line arguments detailed within each file. Within `data/small` are examples of the `.txt` outputs that are created through this process.

Routing

From within the pyroute subfolder, run `python execute.py [path/to/processed/osm]`. You will then be met with a series of prompts detailing:

1) crime types (optional)
2) safety importance on a scale of 1 to 10
3) start and end addresses
4) maximum distance to walk (optional)

Reasons for failing may include:

- typos in the input addresses (3)
- not providing a number for safety importance (2)
- not typing crime types exactly (1)
- (4) is greater than the distance traversed by the calculated path, but not greater than the absolute distance between the origin and destination

Note that failing to meet the specifications for the prompts will result in that parameter being ignored. Also note that the address input will map to the nearest routeable node using PyRoute functionality. In practice this often means that the route will start out from the closest intersection to your input address.

For a visual depiction of your route, input the list of lat/lon coordinates into http://www.darrinward.com/lat-long/.

Planning

See `report/draftspec.pdf` and `report/finalspec.pdf` for specific annotations where green means successful completion while purple symbolizes modification.

We planned to implement:

- A module that took OpenStreetMap data and added NYC crime data
- An algorithm that utilized the crime data to build a walking directions engine
- Coverage of Manhattan between 100-120th Streets

We completed these primary goals, and also completed many of our additional feature goals:

- User ability to designate the maximum length of travel
- User ability to designate how seriously they take the risk of crime
- User ability to designate which types of crime they are concerned about
- Measurement of distance walked and time estimate in walking directions
- Coverage of all of Manhattan

Design and implementation

Data Processing

This was essentially what was outlined within the draft and final specs. In our final implementation, `process.py` generates a new OSM file by:

1) restricting the `crime.geojson` data to the boundaries of the given OSM file, resulting in a dictionary
2) matching crimes within this dictionary to nodes in the OSM file, generating a node-crime dictionary
3) using this node-crime dictionary to modify the OSM file's ways: in each way, we insert a crime if the way has a node with that crime
    a) note that crimes have IDs so as to not be double-counted

Pyroute Modifications

- `route.py`: removed command-line action, only using the `Router` class
    - also added a `coords` method to return the lat/lon of a given node ID
- `loadOsm.py`: added functionality for crime weights
    - in `endElement`:
        - initialize a list of crimes for each way
        - use the crime list to weight the way
    - the `CrimeWeights` dictionary comes from `crimeweights.txt (which is created during each run and destroyed after the route is complete)`
- `gen_dict.py`: a new file to modify `CrimeWeights` dictionary, outputs `crimeweights.txt`
    - ratios of crime types hard-coded in
    - `weight_crime` scales these hard-coded values, then ignores the types specified
- `addtocord.py`: uses Google Maps API to translate addresses into lat/lon coordinates
- `execute.py`: brings functionality together into one file executable from command line
    - parses user input of crime types to ignore and safety importance for `gen_dict.py`
    - parses user input of addresses with `addtocord.py`, then uses `loadOsm` method to get node id of lat/lon
    - calculates as-the-crow-flies distance, parses user specification of distance
    - routes using `Router` from `route.py`
    - prints out lat/lon list path, distance of path, and time required to traverse path

Implementing the address to latitude/longitude feature was reasonably straightforward. We explored several different possible APIs that offered this feature but ultimately settled on Google Maps. It was the fastest and most accurate of the services that we tried. Using the API turned out to be extremely helpful for testing, because using node IDs was very limited in PyRoute.

Reflection

Group Work

For brainstorming, conceptualizing, and modularizing, we worked as a group. In terms of actual coding, Jessica worked on the core functionality and integrating all the pieces together. Derek and Ramsey worked on essential features such as the address API and managing user inputs so that address input was possible, as opposed to node ID inputs. Alice pulled together the project through working on the video, tying together OSM and algorithm concepts.

We were pleasantly surprised by…

How easy it ended up being implementing crime weights into the routing algorithm once we had added crime tags to the pertinent nodes.

How easy python was to learn.

How nicely our original modularization allowed for us to break up the project.

<u>We were not-so-pleasantly surprised by…</u>
Run-time. First off, python as a language is generally less efficient. More importantly, OSM data sets are massive, even for a localized area like Manhattan, and it took longer than expected to add the relevant crime tags to the OSM data because for each crime, a full run through the OSM data was needed. The code that added the crime had to go through each OSM node in the data set for each crime object added. This was incredibly inefficient.

Additionally, PyRoute's documentation is pretty horrid/nonexistent. We really had to look at the code closely before we began modifying it or doing the add-ons that allowed the user to input an address rather than a node id.

<u>If we had more time, we would…</u>
Find some way to make it give directions, as opposed to just a list of coordinate pairs that outline your route. Perhaps we would try to figure out a way to lay the route onto a GUI. We also might try optimizing the data loading and run time.

<u>Overall Takeaways</u>
It's really enjoyable to take full control of a project. Even with all the bumps along the way, the idea of reaching our goal drove the project forward. We were very satisfied with what we managed to complete!