WalkSafe:
Finding Safest, Shortest Paths in Manhattan

Kasey Luo
Department of Computer Science
Stanford University, Stanford, CA 94305
{kaseyluo} @stanford.edu

Daniel Guillen
Department of Computer Science
Stanford University, Stanford, CA 94305
{eliasd} @stanford.edu

Shannon Yan
Department of Computer Science
Stanford University, Stanford, CA 94305
{shannony} @stanford.edu

Abstract

Our project aims to use crime data in Manhattan, New York from the past 10 years to map the safest and reasonably timed efficient path for a user to walk in Manhattan, New York. Through our project, we hope to provide a tool for users in Manhattan to protect their personal safety. By representing New York City as a graph, we were able to model finding the safest, shortest path as a search problem. Our final product is a tool that allows users to enter a start and end location, weigh how heavily they prioritize safety versus distance, and output a path that satisfies these requirements to a Google Maps visualization. The current code for this project can be found in this Github repository: `https://github.com/kaseyluo/WalkSafe`.

1 Introduction

Studies have shown that 85% of women have changed their traveled route to avoid dangerous situations such as assault.[1] This is especially true in urban areas, and commuter safety is an issue which reaches almost all groups including children, the elderly, and late-night lone travelers.

To help address this issue, we used crime data in Manhattan, New York from the past 10 years to map the safest and reasonably timed efficient path for a user to walk in Manhattan, New York. Using a graph to represent the city and historical crime data, we were able to find the safest, time efficient path.

2 Literature review

2.1 Related work

We found that a number of researchers using different approaches have attempted to find shortest, safest paths including a past CS 221 team at Stanford. The team at Stanford had a similar goal, but instead of using a graph with nodes and edges, they modeled Chicago as a grid with each coordinate representing a specific coordinate of the city. They then used UCS to find the safest path under a time constraint.[2] One drawback to the grid approach is that it does not take into account the actual streets within the city. Instead, in their implementation, UCS uses the crime grid and chooses whether to go up, down, right, or left. Walking paths do not quite cut across the city moving perfectly vertical or

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

horizontally, so we differed out approach to look at the actual streets and intersections of the city, which we felt may output more accurate directions.

Another approach detailed in a paper titled SafeRoute: Learning to Navigate Streets Safely in an Urban Environment by Sharon Levy, Wenhan Xiong, Elizabeth Belding, William Yang Wang at UC Santa Barbara, used deep reinforcement learning.[1] Each state contained the current location, the possible actions (North, South, East, West), probability of moving from one state to another, and then the reward function. Originally, our team was considering using a reinforcement learning approach but found that using search methods made more sense since it is difficult to determine accurate transition probabilities for something like crime. To determine their transition function, they inputted the state into a neural network with two hidden layers, each with a ReLU activation function. The output used a softmax function and returns a probability distribution which then could be used to determine the transition probabilities. In terms of rewards, paths were assigned a reward proportional to their distance from crimes divided by the path length.[1] Although we used a different approach, we were inspired by their reward function to implement our regional safety costs implementation where the safety cost of a node is dependant on the nodes around it. Thus shorter paths will be rewarded more than longer paths with similar crime rates.

2.2 Assumptions

By using historic crime data, an assumption made is that current crimes will follow similar patterns to the past, which may not always be the case. For example, the installation of more lampposts on a street a given year may significantly decrease crime from that point on. If we were to only looked at historic data from before that point in time, then we would likely return inaccurate results. However, we believed that situations like these happen with low probability, thus this would not affect our results to a great degree. Furthermore, the vast majority of projects attempting to solve similar problems use historic data as the basis of their models. Because of these reasons, we believed that using historic crime data was the right decision for our project. In weighing different felonies, we used the Cambridge Crime Harm Index, a widely adopted measurement developed by Professors Lawrence W. Sherman, Peter Neyroud and Eleanor Neyroud which measures how much harm any given crime causes. The use of CCHI has already been adopted by several UK police forces to determine sentencing lengths for various crimes. An assumption we make is that these weights are accurate and reflect actual harm against commuters.

3 Task definition

Our vision for this project was to build a tool that allows young women and other city-dwellers to find the safest, shortest path to their destination. We envisioned this tool being incorporated into platforms such as Google Maps, Apple Maps, and Waze for a streamlined navigation experience.

3.1 Input-output behavior

For our inputs, we take in the starting location and the end location that the user wants to reach. This is in the form of a pair of pairs, the first pair containing the 2 streets intersecting at the cross street of the starting location and the second containing the cross street of their destination. We also allow the user to input a safety and distance weight. For example, if the user inputs a safety weight of "0.8" and distance weight of "0.2," our algorithm knows to prioritize finding a safer path over a shorter path. As output, we generate a list of street intersections that form the safest, shortest path under the inputted safety and distance weights of the user. Our user interface visualized this path on Google Maps, overlaid on an approximate danger map of Manhattan, for easy visualization of the best path. We also output the different evaluation scores assigned to that path, which will be described more in depth in a following section.

3.2 Project scope

We had several factors to consider in determining the scope of our project: geographical span, location of most impact, data quantity, and dynamism of the data. Our decision to choose New York as our city of focus was predicated on the fact that it is the largest city in the United States, based on

population. We initially wanted our tool to span the entire city of New York and incorporate all crimes that occurred since 2006. After a few iterations processing and scaling our project, we decided, for reasons of practicality and computational processing speed, to narrow our scope to just focus on Manhattan and limit the range of crimes we considered to just past felonies since 2006.

3.3 Datasets

Our crime data is compiled from the NYPD Complaint Data Historic, a dataset of all valid felony, misdemeanor, and violation crimes reported to the New York City Police Department between the years of 2006 and 2017. Geographical data is compiled from two main sources: the Overpass API from OpenStreetMap and the geocoding API from Mapbox.

4 Approach

4.1 Challenges

One of the key modeling challenges that our approach needed to be able to capture is an individual's navigation in Manhattan — that is, our model needed to be able to faithfully represent the most important aspects of how an individual moves through the city and present solutions that are meaningful in the context of navigating in the city.

4.2 Problem model

4.2.1 Model of Manhattan

In order to provide an effective and tractable solution to the safest, shortest path problem, we chose to model Manhattan as a graph composed of *intersection nodes* and *street edges* between intersection nodes. That is, each intersection in Manhattan is uniquely represented as a node by its latitude and longitude coordinate and each street component between two adjacent intersection nodes is considered an edge — we represent an edge between two nodes simply as the pair of both node coordinates.

Under this model of the world, a solution to the safest, shortest path problem would be an ordered sequence of intersection nodes. This node sequence corresponds to the ordered path (from intersection to intersection, along particular street edges) that one would take from the starting intersection to the destination intersection.

One the benefits of this model (as it relates to the real world) is that (for the most part) any particular location in Manhattan (both a starting location and final destination) can be assigned to its nearest intersection — for a person, it would be the equivalent of walking to the nearest intersection to start your navigation and then walking from your ending intersection to your specific final destination.

4.2.2 Search problem definition

Given that we are choosing to model Manhattan as a graph of intersection nodes and street edges, we model the safest, shortest path problem according to the following search problem definition:

$$
\begin{aligned}
&s_{start} = (\text{start node latitude coordinate}, \text{start node longitude coordinate}) \\
&Actions(s) = Neighbors(s) \\
&Cost(s, a) = Distance\ Cost(s,\ Succ(s, a)) + Safety\ Cost(s,\ Succ(s, a)) \\
&Succ(s, a) = a \\
&isEnd(s) = [s = \text{end node}]
\end{aligned}
$$

where "Neighbors" maps each state i.e. a given intersection node to the set of all nodes neighboring s, and the costs are defined in the next section. The states are simply tuples of the latitude and longitude coordinate of an intersection. The start state is the starting coordinates and the end state is the ending coordinates. The successor and cost function simply take in the current state, find the neighboring intersections to the state, and compute the cost to move to the neighboring intersection.

4.2.3 Cost definitions

Since our task is to find the safest, shortest path, we define two key costs that make up our search problem definition: *distance cost* and *safety cost*. Because we are interested in both minimizing distance traveled (distance cost) and minimizing the "amount" of danger we are in (safety cost), we need to clearly define our (initial) safety and distance costs as well as how these costs are computed for each edge and intersection.

Our initial cost model for moving from one intersection, A, to another intersection, B, i.e. $Cost(A, B)$ is the sum of the safety cost of the edge between both nodes, the safety cost of the destination intersection, B, and the distance between both nodes. In the search problem definition, the sum of the safety cost of the edge and the safety cost of the destination intersection node is placed under "Safety Cost".

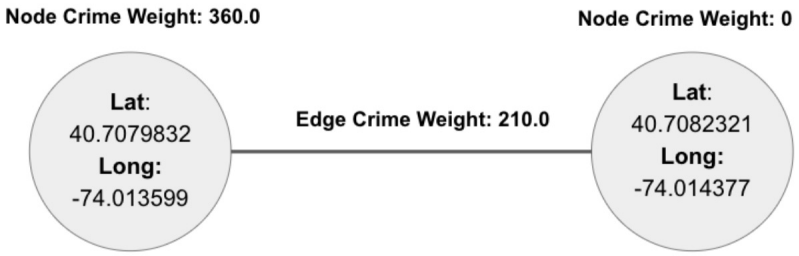
$$\text{Cost}(A, B) = \text{Distance Cost}(A, B) + \text{Safety Cost}((A, B)) + \text{Safety Cost}(B)$$

Our initial definition of the safety cost of a given edge or intersection, x, is a weighted sum of all the crimes that have occurred on the given street or intersection node — a larger safety cost means that a given location is relatively unsafe while a lower safety cost means that a location is relatively safe. The weighting for each crime is assigned depending on how severe each crime was and the magnitude for these weightings are determined in accordance with the Cambridge Crime Harm Index (see section 2.2 for more detail on the CCHI). The data on how many and what types of crimes have occurred at any particular location is compiled from the NYPD Complaint Data Historic (see section 5.1 for more detail).

$$\text{Safety Cost}(x) = \sum_{\text{crime} \in x} \text{Weight(crime)}$$

where crime $\in x$ refers to each crime that has occurred on a given intersection or street, x, and the "Weight" function is a mapping from each crime to the weight of the crime. We assign weights to each crime a depending on how severe each crime was. To determine the magnitude of the weights relative to each other (the larger the weight, the more severe the crime was), we used the Cambridge Crime Harm Index, mentioned earlier in section 2.2.

The distance cost of an edge between two intersection nodes is simply the Euclidean distance between both intersections. Once the safety costs are computed, we can end up with a model that includes subsections like the following:



4.3 Baseline

For our baseline, we implemented greedy search. Starting from the start node, the greedy algorithm simply looked at all the unvisited neighbors of the current node and chose the next intersection node with the lowest cost. The algorithm terminates when it either reaches the destination or when it has already explored all the neighbors of the node it is currently on. The majority of the time the greedy algorithm was unable to find a path because when it simply chose the intersection with the lowest cost, many times it ended up at a node where it had already visited all of its neighbors. While it was able to find paths on certain inputs, the obvious drawback is that it is not optimizing for the total safety cost and distance cost of the entire path, only of the next step.

4.4 Oracles

Within our search problem definition, we chose our oracle to be uniform cost search — UCS works perfectly well with our search problem definition and is guaranteed to find the path that minimizes total cost. However, while this is a good oracle for our defined problem, it's harder to define an oracle for the real world where there is no intrinsic measurement of "safety" for any particular location and, by extension, any path — for this reason, it's hard to perfectly asses how "safe" our paths are aside from comparing paths to neighborhood crime rates in Manhattan and to each other using the safety cost definition.

4.5 Uniform Cost Search & A*

With the search problem defined and an initial cost definition determined, our graph model of Manhattan allows us to start to employ two key search algorithm inference solutions to efficiently and effectively answer the safest, shortest path problem: Uniform Cost Search and A*. Both of these operate under the search problem definition from section 4.2.2, using the initial cost definition we provide but we also experimented with different cost models (see section 4.6).

4.6 Optimizations & improvements

In addition to our initial problem approach, we also chose to see if we could improve on our model and inference approaches and achieve different and potentially different results. One key optimization was a Euclidean distance heuristic as part of A*. More specifically, the heuristic was the Euclidean distance between the current node and the destination node.

4.6.1 Regional safety cost definition

In addition to computing the safety cost for each intersection node and street edge based on crimes that happened at that specific location, we wanted to compute a set of safety costs that would represent a more generalized understanding of regional safety. We did this by allowing crimes within a certain radius of an intersection node to impact the safety cost of that particular node. This would allow for a more realistic model of how a user would navigate the streets of a city, with some generalized knowledge about a region, but not with specific knowledge about the danger of a particular street. We refer to these new safety costs as *regional safety costs* and the initial safety cost definition from 4.2.3 as *local safety costs*. More concretely, the regional safety cost for an intersection node x is

$$\text{Regional Safety Cost}(x) = \sum_{y \in \text{Nearby}(x)} \frac{[\text{Local Safety Cost}((x, y)) + \text{Local Safety Cost}(y)]}{\text{Distance}(x, y)}$$

$$\text{Nearby}(x) = \{y \mid y \in \text{Nodes},\ \text{Distance}(x, y) < \text{Radius}\}$$

where Nodes is the set of all intersection nodes in Manhattan and Radius is some exploration radius that we choose. Note that the regional safety costs are only defined for intersection nodes so there is no associated cost for a street edge and the "safety cost" term in the search problem definition only evaluates the regional safety cost at the destination node.

4.6.2 Normalization and bias coefficients

Another one of the areas that we experimented with to see if we could achieve interesting results is redefining our overall cost model i.e. "Cost(A, B)" from the search problem definition. Modifications to our cost model included adding coefficients to *bias* our safety and distance cost components and also *normalizing* each of our cost components using their respective average values in our model. This is the modification using the localized safety costs:

$$\text{Cost}(A, B) = \alpha \cdot \left(\frac{\text{L. Safety}((A, B))}{\mathbb{E}[\text{L. Safety}((A, B))]} + \frac{\text{L. Safety}(B)}{\mathbb{E}[\text{L. Safety}(B)]} \right) + \beta \cdot \left(\frac{\text{Distance}(A, B)}{\mathbb{E}[\text{Distance}(A, B)]} \right)$$

Here is the modification using the regional safety costs:

$$\text{Cost}(A, B) = \alpha \cdot \left(\frac{\text{Regional Safety Cost}(B)}{\mathbb{E}[\text{Regional Safety Cost}(B)]} \right) + \beta \cdot \left(\frac{\text{Distance}(A, B)}{\mathbb{E}[\text{Distance}(A, B)]} \right)$$

The motivation behind this is that by normalizing each of our components, they are all on the same scale of magnitude which prevents any one metric from dominating the total cost — that is, each component contributes equally to the overall cost. The bias coefficients allow us to weight both metrics (and thus, proportionally amplify or diminish each of their contribution to the cost computation) — this allows us to experiment with different paths generated when we focus more on one metric over the other.

4.7 Evaluation metrics

Since our task is to find the safest, shortest path from a start to end destination, we define two corresponding evaluation metrics, *total distance cost* and *total safety cost*, for evaluating how well different paths achieve this goal — a smaller cost corresponds to a shorter or "safer" path, respectively. The total safety cost metric is defined as follows:

$$\text{Total Safety Cost} = \sum_{x \in \text{path}} \text{Safety Cost } (x)$$

where x is an intersection node or street edge on the given path and "Safety Cost" is a function (defined below) that evaluates the particular "safety cost" at a given intersection node or street edge — in our case, we chose to use the local safety cost definition as our safety cost evaluation function.

The total distance cost of a path is simply defined as the total distance traveled — that is, we simply sum the length of each street edge s of the given path:

$$\text{Total Distance Cost} = \sum_{s \in \text{path}} \text{Distance Cost}(s)$$

where the "Distance Cost" function computes the Euclidean distance between the two nodes of street edge s.

5 Infrastructure & data processing

5.1 Identifying every intersection in Manhattan

In order to create an accurate model of Manhattan, we needed to accurately generate a graph whose nodes would reflect the true locations of intersections in Manhattan and whose edges would represent the true street components between intersections. In order to achieve this, we used the Overpass API from OpenStreetMap in order to generate a preliminary list of intersection coordinates in Manhattan and then used various scripts in order to either filter inaccurate intersection coordinates or combine duplicate / similar intersection coordinates.

5.2 Computing localized safety costs

In order to compute the localized safety costs of each node and edge, each crime from the NYPD dataset is assigned to either an intersection or a street. Using the coordinates of the intersections and the coordinates of the crimes, we used the Mapbox API to map each crime to its nearest intersection or edge if it wasn't close enough to be considered as located at an intersection.

Date	Crime Key	Offense Type	Offense Description	Crime Type	Borough	Latitude	Longitude
10/12/17	106	FELONY ASSAULT	ASSAULT 2,1,UNCLASSIFIED	FELONY	MANHATTAN	40.7911519	-73.884372

Figure 1: Example of cleaned data from the NYPD Complaint Data Historic.

6 Results

We will demonstrate the findings of our project through 7 figures. For each example, we will be using the following start and end intersection:

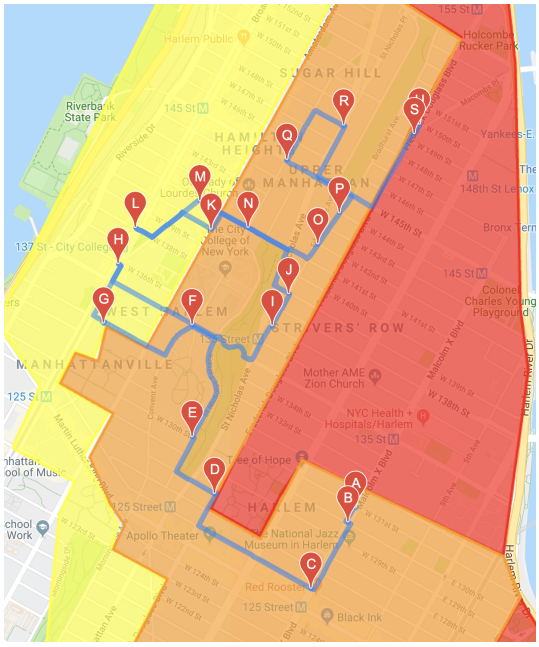
$$\text{Start Intersection} : (\text{`Malcolm X Blvd'}, \text{`West 132nd Street'})$$
$$\text{End Intersection} : (\text{`Frederick Douglass Boulevard'}, \text{`West 150th Street'})$$

For context, these intersections are located in north Manhattan in the Harlem area. For each figure, we will report both the safety and distance evaluation metric scores. In figures that incorporate safety and distance bias coefficients, we will use the following notation:

(S, D): (*safety coefficient*, *distance coefficient*).

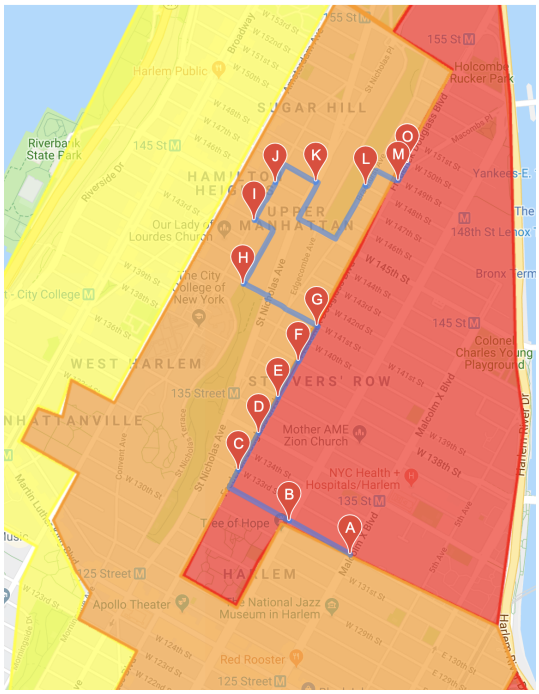
6.1 Baseline algorithm: greedy

Total Safety Cost: 14148.0
Total Distance Cost: 7819.3



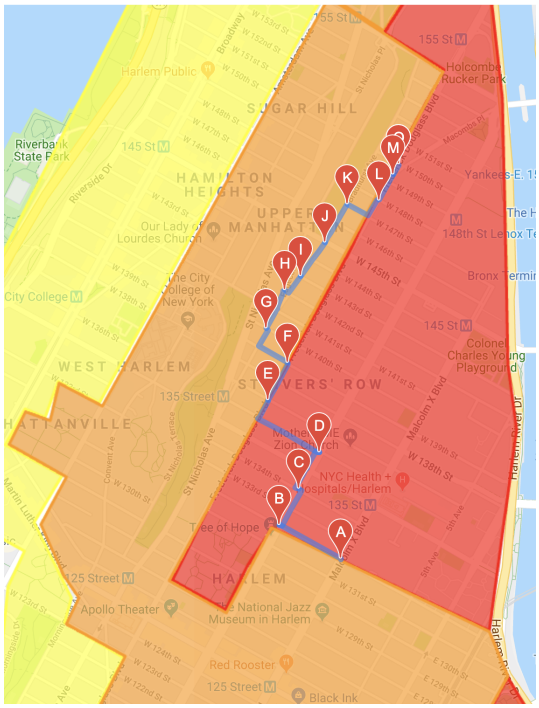
6.2 Localized safety costs: unnormalized

Total Safety Cost: 3375.0
Total Distance Cost: 2826.2



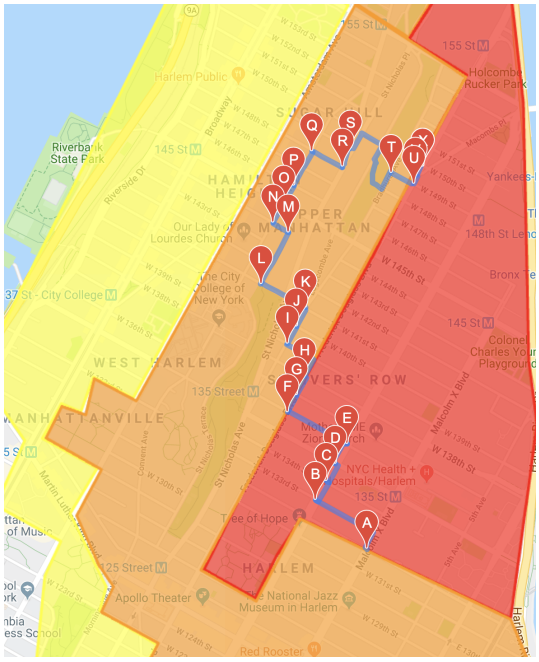
6.3 Localized safety costs: normalized

Total Safety Cost: 5292.0
Total Distance Cost: 2250.4



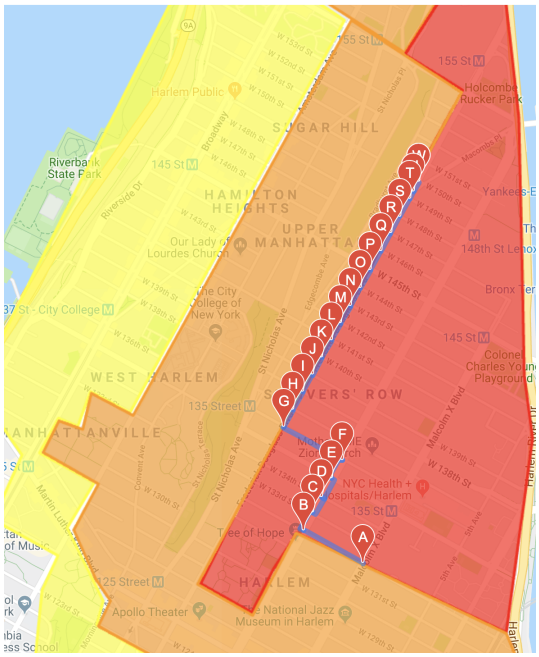
6.4 Localized safety costs with bias coefficients: safety prioritization [1]

(S, D): (0.7, 0.3)
Total Safety Cost: 4615.0
Total Distance Cost: 2843.8



6.5 Localized safety costs with bias coefficients: distance prioritization

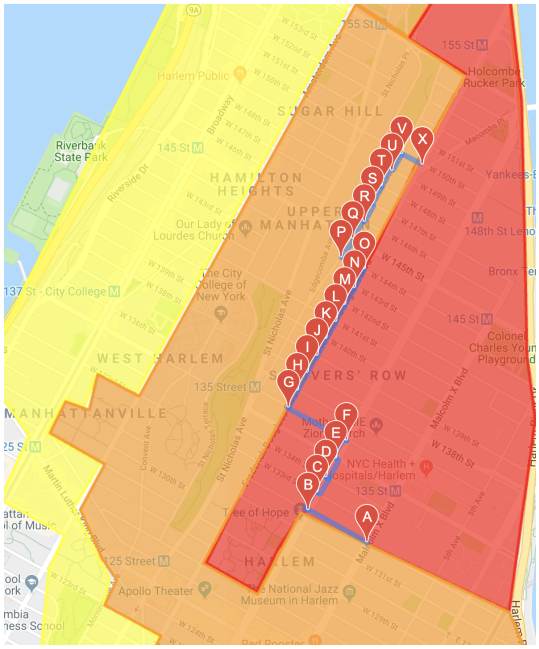
(S, D): (0.1, 0.9)
Total Safety Cost: 7223.0
Total Distance Cost: 1998.5



[1] From this point forward, we use normalized costs

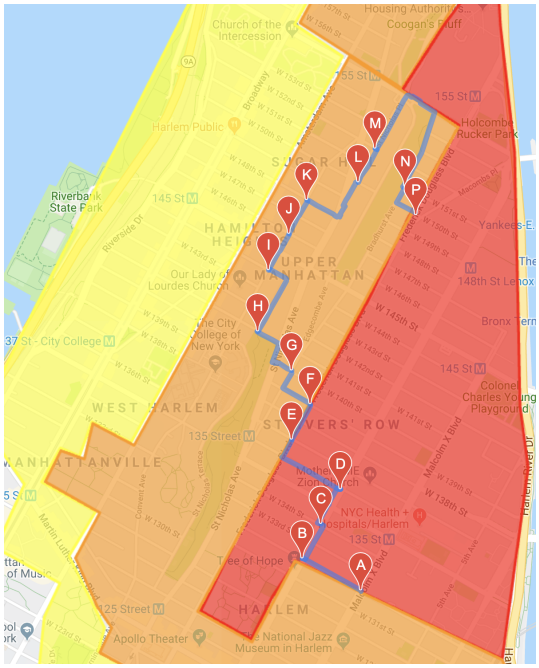
6.6 Regional safety costs with bias coefficients: safety prioritization

(S, D): (0.7, 0.3)
Total Safety Cost: 9323.0
Total Distance Cost: 2199.6



6.7 Regional safety costs with bias coefficients: convergence

(S, D): (0.9, 0.1)
Total Safety Cost: 6023.0
Total Distance Cost: 3143.1



7 Analysis

7.1 Greedy algorithm: worst performance

As expected, our greedy baseline in 6.1 performed the worst compared to all other algorithms and optimizations. Its total safety and distance costs were the highest (worst) of all other figures. Because it always chose the street that was immediately the safest, its outputted path is convoluted, occasionally even circular. It is interesting to note that when overlaid on top of our approximate crime map, it does venture in the yellow region, which historically is a safer region of the city. On other combinations of start and end intersections, our greedy baseline sometimes returned no path, as it too greedily exhausted all neighbors and became cyclic.

7.2 Localized safety costs: unnormalized vs. normalized

The total safety cost of the unnormalized localized costs in 6.2 was better (smaller) than the normalized version in 6.3. However 6.2's distance score was worse. This means that the safety costs played a larger role in affecting the total cost of a street or intersection compared to distance costs in the unnormalized version of the costs. When we normalize safety and distance accordingly, the algorithm optimizes more for distance comparatively. Visually, we see that 6.3 is a smoothed version of 6.2, a more reasonable path for a user to walk.

7.3 Localized safety costs: safety vs. distance prioritization

Contrasting when we prioritize safety in 6.4 to when we prioritize distance in 6.5, we see that, as expected, the path for 6.4 has a lower total safety cost but higher distance score compared to 6.5. When we evaluate the paths visually, we see that 6.4's path ventures more into the orange region of the city, which in theory, is safer than the red region of the city. 6.5's path, on the other hand, returns a path that has the lowest distance score compared to any other figure.

7.4 Regional vs. localized safety costs: safety prioritization

Using the same safety and distance bias coefficients (0.7 and 0.3, respectively), we see that using regional costs in 6.6 returned a path that was more optimal for distance compared to 6.4. Upon reflection, we realized that this is likely due to the fact that when accounting for regional safety costs, the difference in cost between adjacent streets is smaller, since regional costs factor in all crimes in a given radius and neighboring nodes will have overlapping regions. Thus, the model would be less inclined to choose a path that strays from the optimally distanced path. We can also see visually that 6.6 is a smoother version of 6.4, and represents a more realistic path for a user to walk on. We interpret our model under the regional safety costs implementation to be a more realistic representation of a user navigating the streets of Manhattan. That is, a user might have a general sense of regional safety, but would not have the granularity of street-by-street safety that our localized safety costs models. Thus, this is why our regional safety costs lead to a more reasonable path.

7.5 Regional vs. localized safety costs: convergence under safety prioritization

When we heavily prioritize safety using regional safety costs in 6.7, we see that the path returned is more similar to 6.4. This convergence makes sense. Though regional costs tend to optimize more for distance as justified in 7.4, when we heavily weight safety, small differences in cost between nodes become more pronounced. Thus, the model would be more likely to choose paths that lie off the optimally distanced path in order to lower safety costs.

7.6 Unnormalized costs: lowest safety cost

One surprising finding through this process was that the path returned in 6.2 with unnormalized costs and no bias coefficients actually returned the path with the lowest total safety cost. This was at first quite unexpected, but upon reflection, makes sense. The total safety cost itself is calculated by summing the *unnormalized* crime weights of all past crimes on the returned path. Thus, 6.2's search problem was optimizing on the same weights that it was evaluated on. In contrast, figures 6.3-6.7

optimized on normalized weights but were evaluated on unnormalized weights. Thus, their larger total safety costs are more understandable.

8 Future Improvements

In the future, we are looking to create more heuristic functions to speed up the search algorithm. We also want to make our crime weights more dynamic, so that the model can adjust based on real-time data. We would also want to factor in time of day, region, season into how we calculate our safety costs and make those features more flexible. We would also like to penalize path complexity in future iterations–that way the path returned would not be overly complicated. An overly complex path might lead a user to get lost, which could lead to dangerous outcomes. We would also hope to expand our data set to include other cities, as well as improve our user interface to be even more user-friendly.

9 Acknowledgments

Thanks to Professor Moses Charikar and Professor Dorsa Sadigh for their fanastic instruction and a great quarter. We are especially thankful to Benjamin Petit, our mentor for his guidance, and the entire teaching staff for supporting us!

10 References

[1] Calgary, Open. "NYPD Complaint Data Historic." NYC Open Data, data.cityofnewyork.us/Public-Safety/NYPD-Complaint-Data-Historic/qgea-i56i/data.

[2] CS221 Search Problem/UCS Starter Code

Google Maps Platform | Google Developers, Google, developers.google.com/maps/documentation/. remove item

[3] Levy, Sharon, et al. (2018) SafeRoute: Learning to Navigate Streets Safely in an Urban Environment. Cornell.

[4] "Mapbox." Mapbox, www.mapbox.com/.

[5] Ramaswamy, Akshay, et al. (2018) Predicting Commuter Danger and Modeling Path Safety. Stanford.