AI for Safe Walking Routes in San Francisco

Eghosa Amadin, Shane Berger, Sahar Markovich

Stanford University

Stanford
Computer Science

Introduction

When you use Apple or Google Maps to get from one place to another, the route given to you primarily optimizes distance. On its surface, this seems like an ideal model for a map, but when we scrutinize this more carefully, we see that it has its pitfalls. The most egregious shortcoming of the current state of map apps is the lack of emphasis on safety. In metropolitan cities, pedestrians often run the risk of becoming victims of harassment, muggings, and assault, among other crimes. For our project, we created a new path-finder that optimizes for both distance *and* safety. We focus on only walking routes since pedestrians face the highest risks when navigating unsafe regions. Our goal with this project is to create an algorithm that allows people to find distance-efficient routes that do not compromise their safety.

Problem Definition

Our implementation formalizes finding the safest route as a uniform cost search with the following characteristics:

States: Each state is a tuple containing the latitude and longitude coordinates of a SF street intersection.

Actions: At each state (intersection), the possible actions are traveling to each of the neighboring street intersections

Costs: The general cost for any given state-action tuple is the product of two components:

- the distance cost: (penalize longer routes)
- the danger cost: (penalize routes that take pedestrians through dangerous parts of town, by finding the danger score of the nearest crime cluster)
  - To calculate the distance between 2 latitude/longitude coordinate pairs, we use a modified version of the Harversine formula[1]
  - To calculate danger score, weight each crime in the cluster by how they affect pedestrians

Data

Our primary dataset documents crimes reported in San Francisco in 2016.

- Classifies each crime by putting it into groups such as assault, theft, missing person, among other categories. Filter by removing crimes that don't affect pedestrians.
- Also contains coordinates, time, and date of when/where the crime occurred.
- Goal: create crime clusters for the current problem by running k-means on this data.

Graphing San Francisco

- Use dataset from DataSF that contains the name of a street, followed by the name of an intersecting street and a number that represents an ordering for the intersections.
- Use the Google Maps Geocoding API to get coordinates for each intersection
- Restructure data into dictionary of neighboring coordinates using the ordering

Stanford
University

Approach

01	Given the origin and destination address, we convert them into coordinates using the Google Maps API.
02	Set up a crime and travel boundary with a 0.5 mile radius in each direction to limit the search radius
03	Partition the crime data into Voronoi cells using k-means (each cell is assigned a danger score proportional to the number of crimes that occured in that cell).
04	Perform a uniform cost search to find an optimal path that is reasonably efficient and minimizes the risk of encountering crime.
05	Plug in the list of collected intersections into Google Directions API to get the final path.

Challenges

Creating the state space

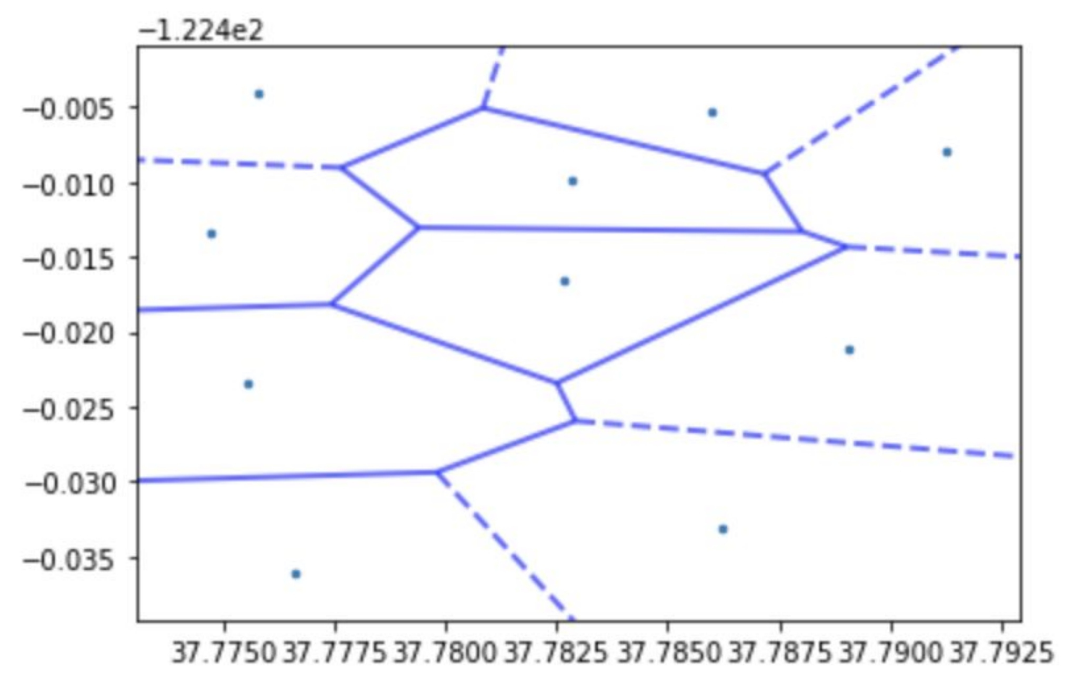
Our search problem required that we had access to the following information for each intersection:

1. The intersection's latitude and longitude coordinates
2. Any neighboring intersection(s) and their respective coordinates

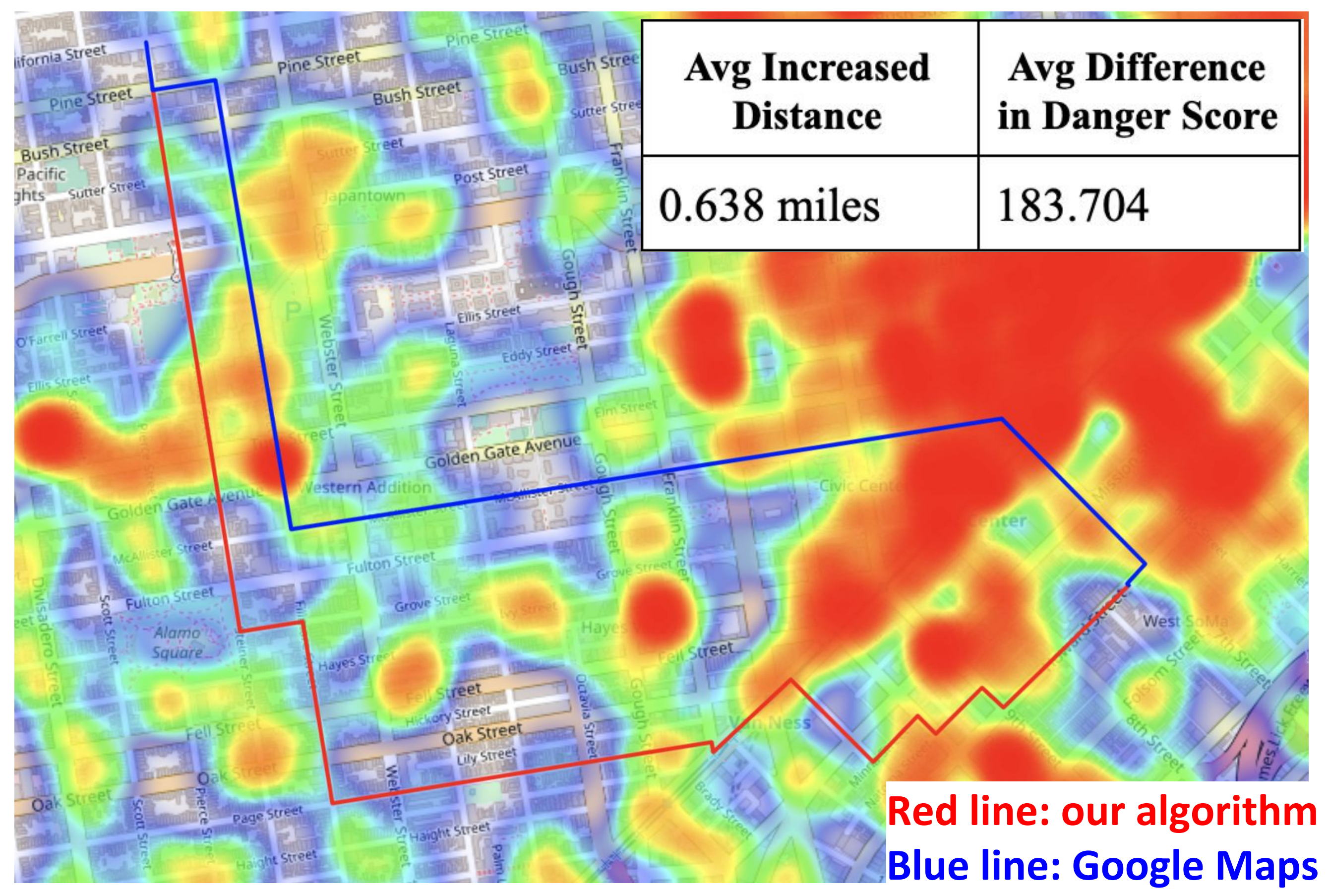
There was no readily available dataset with this information, so we painstakingly created our own graph representation of San Francisco using Google Maps Geocoding API and DataSF.

Using k-means effectively and efficiently

The runtime complexity of our heuristic k-means algorithm is O(*nkdi*). Running k-means on our original crime dataset (*n* = 150,500) took several minutes and the resulting Voronoi cells (see image) often spanned too much of the city to be helpful in the search problem. To fix this, we created a travel boundary so that our k-means would only consider crimes between and near the start and end coordinates. This greatly reduced runtime and the resulting clusters were much more conducive to calculating meaningful danger scores.



Results



Avg Increased Distance	Avg Difference in Danger Score
0.638 miles	183.704

Red line: our algorithm
Blue line: Google Maps

The heat map denotes the frequency of crimes in a given area. As you can see, our algorithm avoids the dangerous red zones nearly perfectly while the Google Maps route goes right through it. Note that while a heat map shows danger zones, the clusters used when calculating the path have a uniform score across a given region, which is why our paths miss some crimeless zones.

Future Work

Given more time, there is a variety of additional features that we would like to implement.

- Beta testing to determine how people react to generated safe paths.
- Travel not limited to starting and ending in intersections.
- Factor in time of day (available in crime data).
- Better stopping heuristic when exploring in the wrong direction.
- Determine better weights for distance vs safety and each crime category.

References

1. Let (φ1, λ1) be the lat-long coordinates of the first point and (φ2, λ2) of the second point. The general Haversine formula uses the following equation to calculate the straight-line distance between 2 points on a sphere:

$$d = 2r\arcsin\left(\sqrt{\operatorname{hav}(\varphi_2-\varphi_1)+\cos(\varphi_1)\cos(\varphi_2)\operatorname{hav}(\lambda_2-\lambda_1)}\right)$$
$$= 2r\arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right)+\cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right)$$

2. Levy, Sharon, et al. SafeRoute: Learning to Navigate Streets Safely in an Urban Environment. 3 Nov. 2018, arxiv.org/pdf/1811.01147.pdf.