Ithaca Sidewalks: A Physical, Geometric, Clustering, Algorithmic Approach

Calvin Chen, Dang Pham, William Xu

November 18, 2019

Contents

Abstract

In this manuscript, we use a physical approach to determine the characteristic lifetime of a concrete slab. Using this, the population density of Ithaca in the day and night, and the total number of concrete slabs in a city block, we calculate the priority score of each city block. We then clustered the city blocks by using the proximity of both physical distance and priority score to minimize the transition cost, and therefore, the overall construction costs. Lastly, we used a geometric approach inspired by concepts from special relativity to derive the constraining surfaces on the solution space of the two-slab problem. Generalization to arbitrary number of slabs is then established with proposal of 2 $\mathcal{O}(N^2)$ algorithm to compute the optimal repair strategy.

1 Executive Summary

Dear Mr. John Licitra, City of Ithaca Sidewalk Manager,

Tasked with helping the Ithaca Sidewalk Improvement team, we tackle some very important problems that are crucial in the goal of improving the sidewalks in Ithaca: determine areas requiring attention, grouping areas of concerns for construction crews, and finding solutions to fix sidewalks through raising, cutting, or replacing slabs.

Understanding that the city of Ithaca has been using an ad-hoc solution to determine priority score of areas in Ithaca, we seek to develop a model of priority score that is based on physical laws. We analyze all possible physical phenomena that can affect a sidewalk slab and found that only the population density and number of slabs in each city block affects the priority score. So, we use population data, Ithaca street map, locations of bus stops, government buildings, and schools to develop a priority score map that reflects the real features of Ithaca city layout and the activities of Ithacans. With this map, we determine that there are three hot spot regions that require high frequency maintenance care: Collegetown, the Commons, and the area around Ithaca High School.

For our next task, to assign regions of the city to each construction crew, we used basic machine learning techniques with human-assistance to find optimal contiguous regions for the crews. Here, we find four contiguous, optimal regions to assign to construction crews that turn out to be quite similar to the existing areas defined by the Ithaca Sidewalk Improvement program.

Then, we developed algorithms for determining the least costly set of repair operations to fully fix sidewalks. Again, using motivation from physics, we develop constraints and explore all possible solutions. We find two fast, inexpensive algorithms that come close to finding the best solution.

For the first two tasks, the largest limiting factor on our proposed solutions is by far the lack of data. Having higher resolution population data and having more easily processible data on environmental conditions throughout the city of Ithaca would allow us to broaden and refineour implementation. Furthermore, our clustering of city blocks requires minor human assistance to find the best clustering solution, which implies some degree subjectivity and lack of automation. In addition, the two algorithms we develop for the last task, although fast, are not always guaranteed to be the optimal solution, but only close to optimal.

With better data from the city of Ithaca, we can perform the analyses in this paper again, but with greater resolution. We can draw specific routes for construction crews, create a map of maintenance frequency for each street, and compile a list of repair actions for each street.

Despite the countless improvements that could be made, our modeling and methods provide superb sidewalk-servicing solutions to the city of Ithaca, generating concrete results that bring us ever closer to a sidewalk utopia – where sidewalks are bountiful and smooth.

Sincerely,

Calvin Chen, Dang Pham, & William Xu

2 Problem Statement

We are tasked with addressing the sidewalk catastrophe in Ithaca and proposing solutions on how to best identify and repair damaged sidewalks. Sidewalks can be damaged due to many different causes, ranging from natural and environmental to human. Individual sidewalk slabs must comply with the requirements set by the Americans with Disabilities Act (ADA). Due to the limited budget, noncompliant slabs must be fixed as cost-effectively as possible once they are reported by the public through complaints. Additionally, population-dense areas, schools, bus stops, and government buildings must be prioritized which deciding which sidewalks to repair first. For the first task, we are commissioned to present a prioritization scheme that indicates the order in which blocks of the city of Ithaca should be repaired.

The logistics of construction must also be carefully managed. As it is costly to send construction crews jumping around Ithaca to repair the most urgent sidewalks here and there, we must present regions where construction crews can be deployed to minimize the costs due to moving equipment. This is the second task.

Once a construction crew is deployed to its assigned domain, the specific repair methods must be carefully considered to select the ones fit for the job that also keep costs down. Then, for the third task, we are charged with determining the best course of action (or lack thereof) for each slab along a city block, given all information about the sidewalk slabs along that block.

Concerns with rising costs, flat revenues, and other detrimental circumstances lends the City of Ithaca to also task us with projecting budget increases over the course of the next 25 years. This is the fourth task.

Recognizing that as undergraduates we only have so much free time on our hands, the City of Ithaca requests that we consider three out of the four tasks presented above. In this paper, we discuss and address the first three, hoping to confront and tackle the disaster of devastated sidewalks in the City of Ithaca.

3 Notation

- τ: characteristic lifetime of a concrete slab
- σ: stress exerted on a concrete slab
  - σ_T: thermal stress
  - σ_W: stress from pedestrians' weight
  - σ_S: shear stress of fluid flowing along an inclined surface
  - σ_R: stress due to tree root growth
- E: Young's modulus
- α: coefficient of thermal expansion
- T: temperature

- n: number of people on a slab
- m: mass of a person
- g: gravitational acceleration
- A: area of a slab
- ρ: density of fluid
- d: depth of fluid
- θ: running angle of a slab
- ϕ: cross angle of a slab
- z: vertical height of the center of the surface of the slab perpendicular to and from the surface of the street
- $1/\Gamma$: characteristic time of a destructive tree root growth event
- P: priority score
- N: population density
  - N_{day}: population density in the day
  - N_{night}: population density at night
- n_{slab}: average number of slabs in a city block

4 Assumptions

City blocks: since our implementation requires the population map of Ithaca and we only have that data on a grid of 9×14 cells, we are restricted to a city block at that resolution. Thus, our city block is not the usual definition of a street block, but rather a city cell as shown in Fig. 3b, 3a.

4.1 Sidewalk Slabs

1. We assume that all concrete sidewalk slabs have the same consistency and are made of the same uniform material, i.e. uniform concrete.
2. We assume all sidewalks in Ithaca are one slab wide.
3. We assume that all slabs are 4 ft $\times$ 4 ft, to comply with the requirement (2) of the ADA, from personal experience, and out of simplicity.
4. We assume that all slabs are 5 inches thick, since they in reality vary from $4 - 6$ inches thick [1].

5. We assume that the x and y positions of the slabs are irrelevant, since the stresses we consider largely will not displace slabs in these directions. More importantly, the ADA compliance requirements do not consider relative x and y displacement of slabs in its sidewalk compliance requirements, so we can ignore them. Additionally, ADA requirement (2) is always satisfied, since we assume all sidewalk slabs to be installed as 4 ft $\times$ 4 ft, and various stresses would not cause them to be less than 4 ft wide.

6. We assume that noncompliance with the ADA due to various sources of stress occurs relatively sparsely, such that at any given point in time, the percentage of noncompliant slabs is small. Then, we can model the breakage of slabs as a Poisson process.

7. As per the City of Ithaca Sidewalk Improvement Program [2, 3], Cornell University manages its own sidewalks and walkways, so we do not consider the entire Cornell University campus in our analysis.

4.2 Causes of Stress

We assume there are four primary source of stress exerted on a concrete slab: thermal stress, pedestrian-caused stress, flowing water-induced stress, and tree root growth-related stress [4, 5].

1. We assume that the thermal stress is caused solely by thermal expansion and contraction of the concrete slab itself, and not due to adjacent non-concrete materials expanding and contracting (e.g. water freezing, soil expanding and contracting). Although the other causes of thermal stress are likely not negligible, we make this assumption for clarity and conciseness of our analysis. Additionally, assuming these other materials are randomly and uniformly distributed throughout Ithaca (this is largely the case for water and soil), we find in Sec. 5 that the thermal stress is a constant across all slabs anyway, so this assumption is in practice, irrelevant.

2. We assume that the soil is uniform in composition and water content throughout Ithaca for simplicity and due to time constraints, even though this is not the case [6].

3. We assume that Ithaca is small enough to approximate the spatial temperature distribution as uniform.

4. The average weight of a human in North America is 80kg [7].

5. We assume that excessive weight loads are sufficiently rare occurrences that they have a negligible contribution to the total stress on a sidewalk slab.

6. We define a tree root growth event to be a change in the tree roots underneath a sidewalk slab that would apply some stress on the slab, whether that is an increase in root mass that pushes up on the slab in certain areas, or a root decay that causes a downward force to act on part of the slab.

7. We assume that the density of trees throughout Ithaca is constant. We can see this visually in the City of Ithaca's interactive map of all of the trees in Ithaca [8].

8. We treat tree root growth as a stochastic process, but since we assume a random, uniform distribution, it does not depend on location. Thus, we can model tree root growth with a characteristic time for one occurrence of growth.

4.3 Slab Repair

Throughout Sec. 7, we make the small angle approximation that a 2% slope is equivalent to 1/48.

5 Priority Score Algorithm

First, we identify the contributions to the damage of the physical condition of the slabs. Then, we include population density, which incorporates all of the other factors that play into the priority score. As discussed in Sec. 5.5, the proximity to schools, bus stops, and government building is incorporated into the population density. Additionally, we assume that the number of complaints is directly proportional to the population density, since if there are more people, there are more likely to be more complaints about broken slabs. Thus, population density and physical condition of the slabs are sufficient to describe the priority of a city block.

5.1 Characteristic Lifetime of a Concrete Slab

Experiments have shown that repeated stress on a concrete slab can cause it to break [9]. Therefore, we assume that a concrete slab can endure a fixed amount of stress over its lifetime, or in other words,

$$\int_0^\tau \sigma(t)dt = \text{const.}$$

where τ is the lifetime of a concrete slab and $\sigma(t)$ is the stress exerted on the slab as a function of time. Thus, if we average $\sigma(t)$ over the life time and express that as $\bar{\sigma}$, we can write

$$\tau \propto \frac{1}{\bar{\sigma}}.$$

5.2 Source of Stresses

We identified the four primary sources of stress on a concrete slab as the following:

1. Thermal expansion and contraction
2. Pedestrians' weight
3. Water flowing down a inclined surface (shear stress)
4. Tree root growth

5.2.1 Thermal Stress

Thermal stress is stress caused by a change in temperature of a material. This stress can lead to fracture or plastic deformation. A material will expand or contract depending on the material's thermal expansion coefficient. As long as the material is free to move, it can expand or contract freely without generating stress. Once this material (in our case, the concrete sidewalk slabs) is attached to a rigid body at one end, thermal stress can be produced. The thermal stress σ_T can be calculated from the following formula [10]:

$$\sigma_T(t) = E\alpha\Delta T(t)$$

where E is the Young's modulus of the material, α is the coefficient of thermal expansion of the material, and $\Delta T(t)$ is the temperature difference as a function of time.

The Young's modulus measures the "stiffness" of a material, or how resistant to deformation a material is. The Young's modulus of concrete is typically in the range of 20–30 GPa [11]. The coefficient of thermal expansion measures the change in linear size of a material under heating and cooling. The typical coefficient of thermal expansion of a concrete is around 10^{-5} $^\circ\text{C}^{-1}$ [12]. The averaged annual temperature variation is around 25°C and the averaged diurnal temperature variation is around 10°C in Ithaca [13]. Using these values, we can estimate $\bar{\sigma}_T = E\alpha\Delta\bar{T}$. Estimate $\Delta\bar{T}$ to be around 1°C, then $\bar{\sigma}_T$ is around 2×10^5–3×10^5 Pa (cf. atmospheric pressure, approx. 10^5 Pa).

5.2.2 Pedestrian-Caused Stress

The stress due to pedestrians' weight, denoted σ_W can be calculated using the following formula:

$$\sigma_W(t) = \frac{n(t)mg}{A}$$

where $n(t)$ is the number of people standing on a slab at a time t, m is the weight of one person, $g \approx 10$ m/s^2 is the gravitational acceleration, and A is the area of a slab. With the average weight of Americans at 80 kg [7], and the slab size ($4' \times 4' \approx 1.5$ m^2), we can estimate $\bar{\sigma}_W = \bar{n}mg/A$. Estimate $\bar{n}$ to be 1 (i.e. assume in the worse case that there is on average 1 person standing on every slab at every point in time). Then, $\bar{\sigma}_W$ is around 500 Pa.

5.2.3 Flowing Water-Induced Stress

The shear stress of a liquid flowing down an inclined surface, denoted σ_S, can be calculated according to the formula in [14] simplified using the free-body diagram in Fig. 1:

$$\sigma_S(t) = \rho g d(t) \sin\theta$$

where ρ is the density of the fluid, $g \approx 10$ m/s^2 is the gravitational acceleration, d is the depth of the fluid, and θ is the inclination. For water ($\rho = 1000$ kg/m^3), we can estimate $\bar{\sigma}_S = \rho g\bar{d}\sin\theta$. Estimate $\bar{d} = 1$ cm, then for a slope of 10%, $\bar{\sigma}_S$ is around 10 Pa.

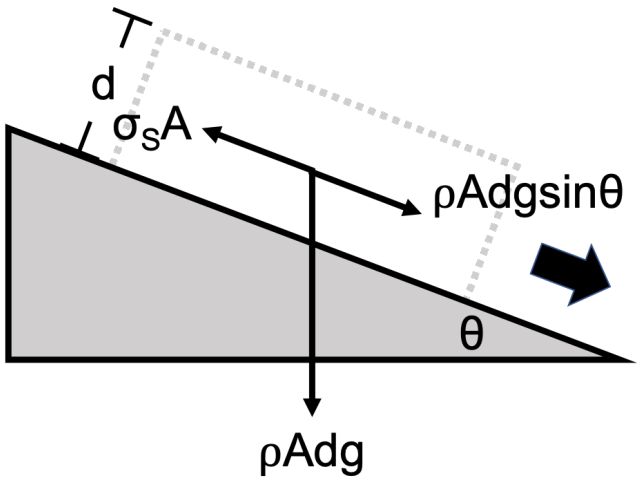


Figure 1: Free-body diagram of a volume of liquid flowing down an inclined surface. The shear stress σ_S can be derived from equating the shear stress with the parallel component of gravity.

5.2.4 Tree Root Growth-Related Stress

Let the characteristic time for one occurrence of destructive tree root growth be $1/\Gamma$. Thus, we can view the tree root growth-related stress, denoted σ_R, as a sum of Dirac delta functions:

$$\sigma_R(t) = c\sum_{k=1}^{\infty}\delta(t-\frac{k}{\Gamma})$$

where c is a constant that measures the total stress released in one tree root growth event. Therefore, the average stress $\bar{\sigma}_R$ is $c\Gamma$.

5.2.5 All Together Now

Thus, combining all four of the stresses, we arrive at the formula for calculating the characteristic lifetime of a concrete slab:

$$\tau \propto \frac{1}{\bar{\sigma}_T + \bar{\sigma}_W + \bar{\sigma}_S + \bar{\sigma}_R}.$$

5.3 First-Order Approximation

In principle, $\bar{\sigma}_T(x, y)$ will be a function of position if the temperature is not uniform across Ithaca. Additionally, $\bar{\sigma}_W(N)$ will be a function of population density N, $\bar{\sigma}_S(\theta)$ will be a function of the slope of the slab, and $\bar{\sigma}_R(x, y)$ will be a function of tree density across Ithaca. However, since we assume that Ithaca is small enough to approximate the spatial temperature distribution as uniform, $\bar{\sigma}_T$ is then a constant. Moreover, because we assume that the tree density is constant across Ithaca, $\bar{\sigma}_R$ is also a constant.

From Sec. 5.2, we can see that the typical value of $\bar{\sigma}_T$ is at least 2 orders of magnitude larger than that of $\bar{\sigma}_W$ and is at least 4 orders of magnitude larger than that of $\bar{\sigma}_S$. Therefore, we can approximate

τ as the following:

$$\tau(N,\theta) \propto \frac{1}{\bar{\sigma}_T + \bar{\sigma}_R} + \mathcal{O}\left(\frac{\bar{\sigma}_W}{\bar{\sigma}_T + \bar{\sigma}_R}\right) = \text{const.} + \mathcal{O}\left(\frac{\bar{\sigma}_W}{\bar{\sigma}_T + \bar{\sigma}_R}\right)$$

Thus, to the first order, the characteristic lifetime of a concrete slab is just a constant across Ithaca.

5.4 Formalism

We assert that the priority score should be proportional to the probability that one of the slabs in the city blocks fails to comply with the ADA. We assume that this failure is a Poisson process (see Sec. 4), so the probability of an individual slab failing is simply the inverse of the lifetime $1/\tau$ of a slab. Therefore, the probability of any one of the slabs failing in a city block will be the sum of this probability over all slabs in the block.

We also assume that the number of complaints the Ithaca government will receive is directly proportional to the population density in the block, since the more people there are, the more likely a person will discover slab failures in a block. Therefore, the priority score will also be proportional to the population density of the city block.

Thus, we define the priority score P of a city block as

$$P = N \sum_{\text{slabs}} \frac{1}{\tau}$$

Using our approximation that τ is a constant from Sec. 5.3 and our assumption that the population density is equal to the average of day-time population density and night-time population density $N = (N_{\text{day}} + N_{\text{night}})/2$. Denote the total number of concrete slabs in a city block as n_{slabs}, we can rewrite

$$P \propto (N_{\text{day}} + N_{\text{night}}) n_{\text{slabs}}.$$

Note that the proportionality constant does not matter, since it is the same for all city blocks, and the priority score is a relative value for comparison between city blocks.

5.5 Data and Implementation

Roads of Ithaca are created using the Python library `osmnx` using road data from OpenStreetMap. With this, we create the map of Ithaca streets (shown in Fig. 2 below), calculate the length of the road, and subsequently how many slabs of sidewalk per street (using the fact that a slab is 4 ft in length). The Ithaca population map is a 9×14 grid of data taken from CensusViewer [15] and Pham et al [16], shown in Fig. 3 below. All of these maps are restricted within the rectangular region bounded by latitude, $42.4613 < x < 42.4278$, and longitude, $-76.5216 < y < -76.47$.

For the day population N_{day}, we use schools, bus stops, and government buildings. The number of each type of building is obtained from OpenStreetMap via the Python library `osmnx`. The population values for each are as follows:

- Each school has approximately 400 students, except for Ithaca high school, which has 1360 students [17, 18].

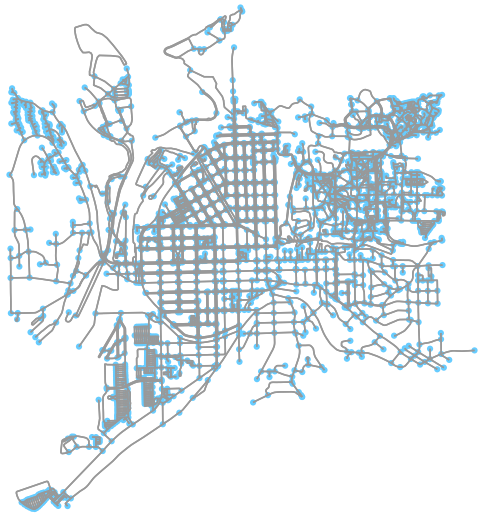


Figure 2: Street map of Ithaca. Blue dots are nodes where streets start and end.

- A bus stop serves on average 30 people per day (this is an estimate).
- Ithaca has 303 government employees [19]. Dividing this over all the government buildings gives us about 34 employees per government building.

The Ithaca day population for each cell at location (i, j) in the 9×14 grid is:

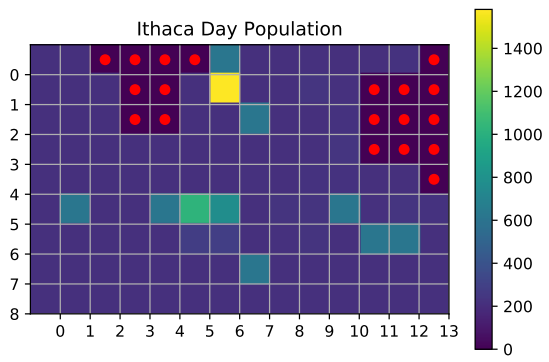
$$N_{\text{day},i,j} = 400N_{\text{school},i,j} + 30N_{\text{bus stop},i,j} + 34N_{\text{government},i,j}$$

where $N_{\text{school},i,j}$, $N_{\text{bus stop},i,j}$, and $N_{\text{government},i,j}$ are the number of schools, bus stops, and government buildings in a cell. Note aside from Ithaca high school, we consider there to be 400 students per school. We count post offices, the town hall, the courthouse, police stations, and fire stations as government buildings. Note that we only find these buildings in cells that do not contain Cornell. Thus, this day population map excludes Cornell.

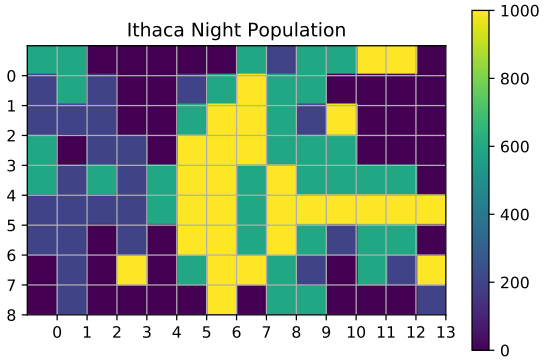
For the night population N_{night}, we use the Ithaca population map. This population includes Cornell students living in Collegetown (where the city *does* maintain sidewalk conditions).

The software implementation is as follows:

1. Retrieve the coordinates of all roads in Ithaca, and calculate the number of slabs each road has using its length (Fig. 2).
2. Discretize the Ithaca road map into a 9×14 grid of cells. Calculate the number of slabs in each cell to get $n_{\text{slabs},i,j}$.
3. Retrieve the location and population of all of Ithaca's schools, bus stops, and government buildings. Calculate $N_{\text{day},i,j}$ in each cell (Fig. 3a).
4. Use data from the Ithaca population map to get $N_{\text{night},i,j}$ (Fig. 3b).
5. Calculate the priority score for each cell using the formalism developed in the previous section. This gives us the priority score matrix, as desired. See Fig. 4.



(a) Ithaca day population N_{day}. The one yellow cell corresponds to Ithaca High School, which has the highest daytime population.



(b) Ithaca night population N_{night}. The yellow regions correspond to the Commons and Collegetown.

Figure 3: Ithaca population during the day and night. The color scale is the number of people in that cell. The red dots are located in cells where population calculations were excluded (Cornell and Cayuga Lake are ignored). Comparing these two plots with Fig. 2, we can see how each cell corresponds to a region in Ithaca.

5.6 Results

The priority score map is shown in Fig. 4. Each cell represents a city block, as defined in Sec. 4. There are a few interesting observations to notice. First, per our definition, the priority score is in units of population/time. Thus, the priority score is a direct metric of the frequency with which a region requires sidewalk maintenance, i.e. the priority map depicts regions based off of how often it needs attention. Second, by comparing with Fig. 2, we notice that there are two main regions requiring attention. The small yellow region that requires the most attention corresponds to Collegetown, and the slightly larger green area that requires the second most attention corresponds to the Commons. This makes sense because these are areas with greater population density. Third, the deep purple areas are either Cornell, Cayuga Lake, or very low population places. Again, this makes sense since these are either areas that do not require regular maintenance due to lack of population, or areas where the City of Ithaca does not manage sidewalks.

6 Optimal Contracts and Clustering

We can approach problem (b) – optimal contracts – by utilizing the priority score matrix from above with a clustering algorithm. Then, the problem becomes unsupervised clustering on 3-space with training data (i, j, wP_{ij}) where P_{ij} the priority score of each city block, (i, j) the location of the city block, and w the weight of the priority score (discussed below).

It is given in the problem statement that the transition cost adds extra time and money. Thus, the goal is to minimize the number of transitions by creating large connected regions of city blocks in

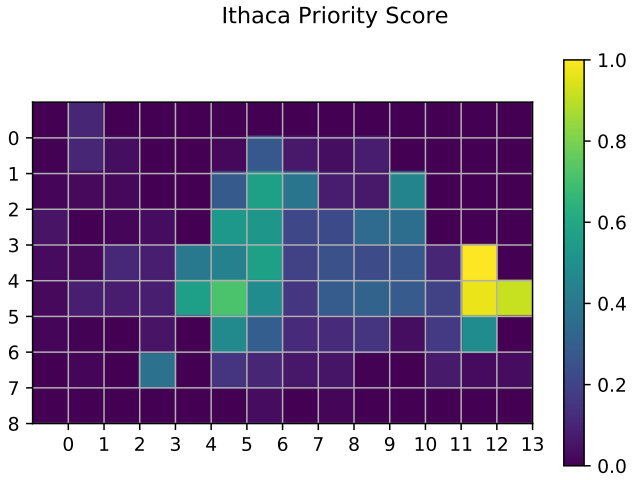


Figure 4: Priority score map of Ithaca. Color scale is based on the normalized priority score, ranging between 0 and 1.

(i, j)-space that are weighted by the priority score $w \cdot P_{ij}$. We consider a region to be connected if all cells are adjacent to at least one other cell of the same region. Two cells are considered adjacent if their sides or corners are touching (i.e. two cells can be diagonally connected, since it is fairly quick to pass along roads from one cell to another along the diagonal point).

Here, w is the factor of how much we want to weigh the priority score versus the spatial coordinates (i, j). A weight of $w = 0$ implies that only spatial coordinates are important, and clustering should be based solely on position. For very large w, instead, priority score is considered more important, and clustering is based primarily off of priority score. We chose $w = 100$, since it gives us the best clustering. (See Sec. 8.1 for the effects of w on the clustering results.)

Thus, the training data (i, j, wP_{ij}) allows us to create clusters based on the need of a region. As a result, a given city block is grouped not only by how close it is to another block, but also by the number of noncompliant slabs. This has an interesting consequence: each construction crew is associated with a different frequency of noncompliance and therefore repair. For example, if we have 4 construction crews (i.e. 4 clusters), then we will have one "high-intensity" crew, which services a small area that has a large number of noncompliant slabs per unit area; two "medium-intensity" crews that service a medium-sized area with a medium number of noncompliant slabs per unit area; and one "low-intensity" crew that services a large area with a small number of noncompliant slabs per unit area.

The (human-assisted) algorithm that we are proposing to group city blocks is then:

1. Choose how many clusters we want on the map. This corresponds to the number of construction crews.

2. Create the priority score matrix with elements P_{ij}. Create the training dataset for clustering with each datapoint (i, j, wP_{ij}).

3. Cluster the dataset using an unsupervised clustering algorithm.

4. Manually alter the clustering if anything is obviously not well-clustered.

The last step is necessary, since we have an unsupervised clustering algorithm, so some abnormal behaviors might appear. Note that this change must not be excessive (no more than a few cells) or we might as well just do this all manually!

6.1 *k*-means

The simplest approach is a centroid-based clustering algorithm: the algorithm initializes k centroids and k clusters associated with these centroids, and then finds:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \mu_i\|^2$$

where $\mu_\mathbf{i}$ are centroids, and $\mathbf{x}$ is a data point in cluster S_i. This is known as the k-means clustering algorithm.

The argument in the equation is also known as the inertia or the within-cluster sum-of-squares. Note that as a result of the way the k-means algorithm is constructed, it requires only one parameter, the number of clusters k. We use the implemented `k-means++` initialization for faster convergence, and the `scikit-learn`'s k-means[1] implementation.

It is important to note that in our (i, j, P_{ij}) parameter space, P_{ij} (units of population/time) does not have the same units as (i, j) (unit of distance). As a result, k-means is not the best clustering technique to use because we cannot weigh these parameters with different units on equal footing.

Running the k-means clustering algorithm on our data, we obtain the following map of Ithaca:

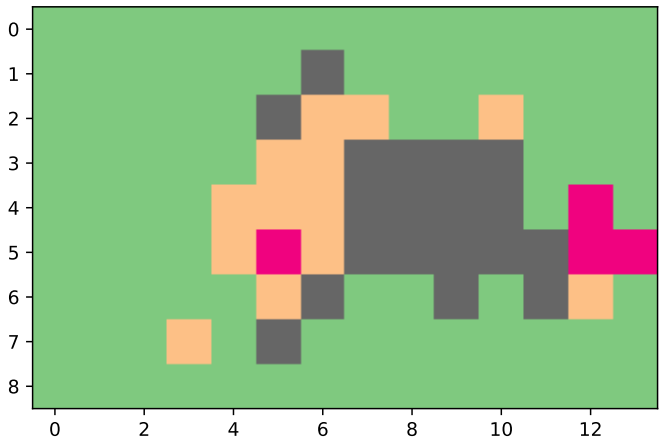


Figure 5: k-means clustering for (i, j, P_{ij}) space with $k = 4$ (i.e. four construction crews).

[1] `sklearn.cluster.KMeans`

6.2 Gaussian Mixture

The Gaussian mixture model[2] is a Gaussian, probability-based clustering algorithm. Data points are generated from a mixture of Gaussian models. Each data point is given a probability of which cluster (i.e. mixture) it belongs to. This method is a soft clustering algorithm (in contrast to the hard clustering method, k-means, described above), since it gives a probability of each point belonging to each cluster instead of definitively assigning each point one cluster. Each data point is then assigned to the cluster with the highest probability.

First, let us consider a dataset $\mathbf{X}$ with N data points, with each data point $\mathbf{x}_i,\ \forall i \in [1, N]$ having dimension d. Here, we are working in d-space. For a multivariate Gaussian distribution, we have:

$$\mathcal{N}(\mathbf{x}_i|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d|\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mu)^T\Sigma^{-1}(\mathbf{x}_i - \mu\right)$$

where $\mathbf{x}_i \in \mathbb{R}^d$ is a data point vector, $\mu \in \mathbb{R}^d$ is the mean, $\Sigma \in \mathbb{R}^{d\times d}$ is the $d \times d$ covariance matrix, and $|\Sigma| \equiv \det(\Sigma)$ is the determinant of the covariance matrix.

For a mixture of K Gaussians, we take the superposition:

$$f(\mathbf{x}_i) = \sum_{k=1}^{k=K} w_k\mathcal{N}(\mathbf{x}_i|\mu_k, \Sigma_k)$$

where w_k is the weight of the kth Gaussian, and K is both the number of Gaussians and the number of components.

We require the following conditions for the weights:

$$0 \geq w_k \geq 1,\ \sum_{k=1}^{k=K} w_k = 1$$

The likelihood function is defined as the combined probability of all data points $\mathbf{x}_i,\ \forall i \in [1, N]$:

$$\mathcal{L}(\mathbf{X}) = \prod_{n=1}^{N} f(\mathbf{x}_i) = \prod_{i=1}^{N}\sum_{k=1}^{K} w_k\mathcal{N}(\mathbf{x}_i|\mu_k, \Sigma_k)$$

$$\implies \ln\mathcal{L}(\mathbf{X}) = \sum_{i=1}^{N} \ln\left(\sum_{k=1}^{K} w_k\mathcal{N}(\mathbf{x}_i|\mu_k, \Sigma_k)\right)$$

The goal is to maximize this likelihood function, i.e. use it as the objective function:

$$\underset{w_k,\,\mu_k,\,\Sigma_k}{\arg\max}\,\mathcal{L}(\mathbf{X})$$

With the log-likelihood given, one possible way to do this is Maximum Likelihood method (finding parameters such that the likelihood function is optimized). To do this, we take the derivative with respect to the variables μ and Σ, set that equal to zero, and then find μ and Σ. However, this can be

[2] `sklearn.mixture.GaussianMixture`

very difficult given the form of the log-likelihood that we have. Another approach is the Expectation-Maximization (EM) method, which is an iterative algorithm. The derivation of the algorithm is fairly involved, so we will just present the algorithm itself. Further information on the theorem derivation can be found in Bishop [20]. The EM algorithm is:

1. Initialize μ_k, Σ_k, and w_k (randomly). Find the log-likelihood with the initial parameter values.

2. Compute the posterior probabilities $\gamma_{Z_i}(\mu_k, \Sigma_k)$ using μ_k, Σ_k, and w_k with:

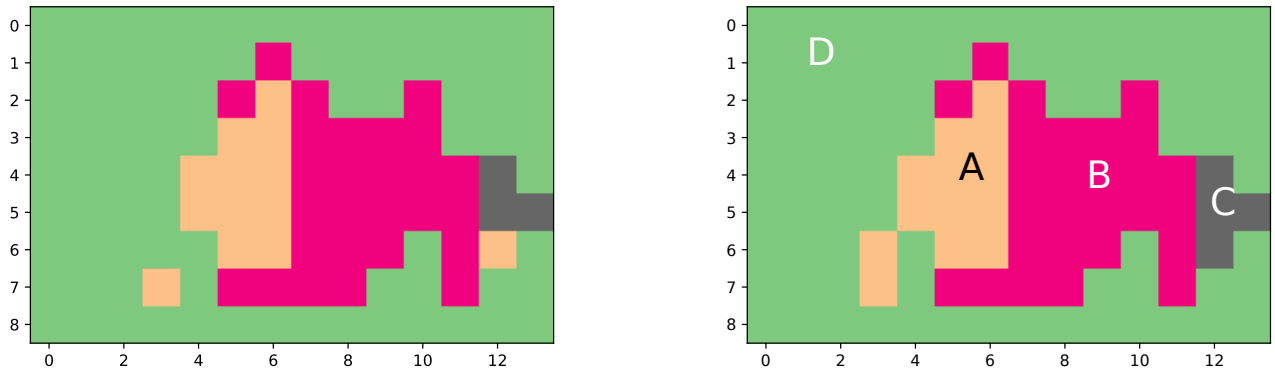
$$P(Z_i = k|x_i) = \gamma_{Z_i}(\mu_k, \Sigma_k) = \frac{P(x_i|Z_i = k)P(Z_i = k)}{P(x_i)} = \frac{w_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{k=1}^{K} w_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}$$

3. Find the new estimates μ'_k, Σ'_k, and w'_k with:

$$\mu'_k = \frac{\sum_{i=1}^{N} \gamma_{Z_i}(\mu_k, \Sigma_k) x_i}{\sum_{i=1}^{N} \gamma_{Z_i}(\mu_k, \Sigma_k)}$$
$$\Sigma'_k = \frac{\sum_{i=1}^{N} \gamma_{Z_i}(\mu_k, \Sigma_k)(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{N} \gamma_{Z_i}(\mu_k, \Sigma_k)}$$
$$w_k = \frac{1}{N} \sum_{i=1}^{N} \gamma_{Z_i}(\mu_k, \Sigma_k)$$

4. Using the newly found μ'_k, Σ'_k, and w'_k, repeat steps 2 and 3 until a set of parameters that is acceptable within some predetermined ϵ precision is found.

Running the Gaussian mixture algorithm on our data, we obtain the following maps of Ithaca:



(a) The "raw" Gaussian mixture.

(b) The human-assisted Gaussian mixture.

Figure 6: Gaussian mixtures for the (i, j, P_{ij}) space with $k = 4$ components. On the left, two cells are unconnected with the rest of their region. On the right, after minor manual edits, all regions are connected now.

6.3 Results

We experimentally determine $k = 4$ to be the optimal number of clusters, hence, the plots depict k-means and Gaussian mixture results for $k = 4$.

The k-means result in Fig. 5 shows some promising clustering. In particular, the algorithm correctly identifies the "hot spot" of Collegetown and the center of the Commons. Furthermore, the regions are fairly contiguous, almost satisfying one of our initial condition that the city blocks for one contractor should be connected. However, it is not good enough for us to go in and correct the disconnected cells manually (we will have to change about 8-10 blocks, which is a lot).

The Gaussian mixture clustering in Fig. 6 shows an obvious improvement over that of k-means. We can clearly see that the cluster regions in Fig. 6a are "almost" contiguous. Here, we can use human assistance to change only two blocks to make the regions completely connected. Fig. 6b shows the results after this manual step. Thus, with $k = 4$ components, we clearly have four regions for the contractors.

The results in Fig. 6b is very interesting. The first construction crew working in area C is the aforementioned "high-intensity" crew. They need to repair more sidewalks per cell per unit time, but cover a much smaller region. The contractors in area A and B come in second and third in terms of work frequency, respectively. They are the "medium-intensity" crews, covering larger range, but also having to repair fewer sidewalks per cell per unit time. Lastly, area D is the region where the sidewalks per cell per unit time that need to be repaired is very low. That is, for area D, the City of Ithaca only needs to deploy a construction crew to each cell *very rarely*. Thus, they can cover a larger range over a longer time scale.

It is important to note that the areas that we are proposing here are *very* similar to the cluster of areas that the Ithaca Sidewalk Improvement program are currently having [2, 3]. The areas proposed by the Ithaca Sidewalk is reproduced from [2] in Fig. 7. Area 5 corresponds to region D, area 3 and 1 correspond to A and B, and 2 corresponds to C.

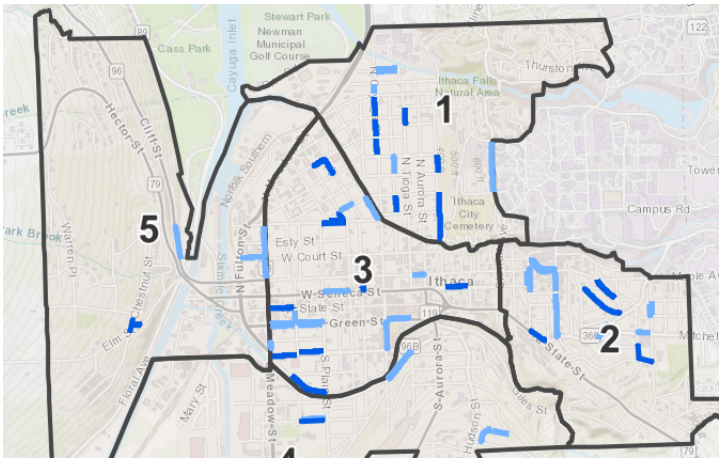


Figure 7: The Ithaca Sidewalk Improvement Program area cluster.

7 Optimizing Slab Repair Costs

A slab can be uniquely defined by three variables: the elevation of the center of the slab z, the angle of the running slope of the slab θ, and the angle of the cross slope of the slab ϕ. Then, any slab can be represented by a point in $z - \theta - \phi$ space. We call the $z - \theta - \phi$ space point representation of a given slab that slab's parameter-point. We want to find an optimal algorithm that most cost-effectively employs the minimum changes in z, θ, and ϕ of all slabs on a given street so that it meets the ADA requirements.

7.1 Clarifications & Interpretations

For the raising repair method, we interpret "raising" to mean changing to soil levels beneath a given slab such that we can change the z, θ, and ϕ values of the slab to be whatever we want (i.e. "raising" means we can both physically raise and lower the slab).

For the cutting repair method, we interpret "linear foot" to be along the length dimension of a slab, parallel to the street. We also interpret "at most 2 inches" to refer to the depth of the slab. Since a slab is 4 ft $\times$ 4 ft, the cost of cutting a slab is $\$16 \cdot 4 = \$64$ on average, while the cost of raising a slab will be $\$5.13 \cdot 16 = \$82.08$. Additionally, replacing a slab costs $\$22 \cdot 16 = \$352$. Thus, it is always cheaper to cut a slab than raise a slab, and it is always cheaper to raise a slab than replace a slab.

7.2 Setup

Now, we just need to determine the conditions under which each repair method is employed. Clearly, if a slab is cracked, it cannot be fixed by cutting or raising, so it must be replaced. However, if a slab is not cracked, it can always be fixed by either cutting or raising, since both cutting and raising are capable of adjusting both angle and elevation (note that, as per our assumptions, elevation, i.e. the z-direction, is the only aspect of position that we need to consider). So, we need to determine the conditions (of an uncracked slab) under which we are able to fix with cutting, and the conditions under which no repairs are necessary at all.

Multiple ADA compliance violations can be fixed at once with one carefully planned cut, so we can consider ADA requirements (3), (4), and (5) together[3]. Any slab that cannot be fixed with one cut must be raised. So, we must determine the possible z, θ, and ϕ values of a given slab that can be fixed with one cut. To do this, we find the set of points in $z-\theta-\phi$ space that are considered "good" (based off of the ADA requirements), in that they do not need any repairs. Then, we find the set of points that are considered "cuttable", in that given the constraints on cutting, and the relationships between z, θ, and ϕ, we can move any point in that set into the "good" region. The remaining set of points in $z - \theta - \phi$ space must be raised.

We first consider the simplest case of a street with only two slabs.

[3]Note that requirement (1) only applies for cracks, which require slab replacement. We assume the slabs are all 4 ft $\times$ 4 ft so ADA requirement (2) is automatically satisfied.

7.3 Two-Slab Case

Let the first slab have its parameter-point located at the origin of $z-\theta-\phi$ space. We want to find the region in which the second slabs lies where no repair action is required, and the setup satisfies ADA requirements.

First, we can characterize the cutting into two categories: cuts that only adjust the height z of a slab, and the cuts that adjust the angles θ and ϕ. Since we are only allowed to cut at most 2 inches into the slab, we know that the change in height for a pure z correction has to be between 0 and -2 inches. Similarly, an angle correction has to have $\sin\theta, \sin\phi \in [-2/48, 2/48]$, which corresponds to a change in slope of -4% to 4%.

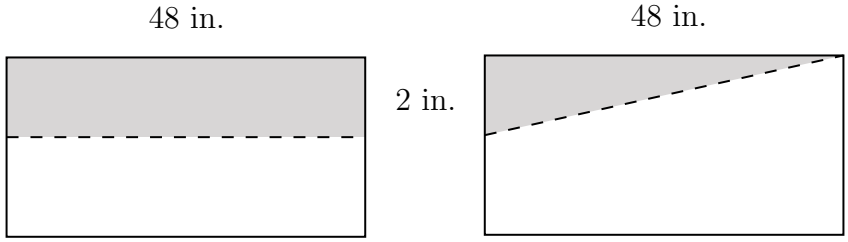


Figure 8: Two different possible cuts. **Left:** a cut that only corrects for z. **Right:** a cut that corrects for θ and/or ϕ. We can see that if we want to correct for θ, ϕ, z must change as a result.

However, from Fig. 8 we can see that in order to correct for θ and/or ϕ, we must simultaneously change z. Thus, we can draw out a region in the $z-\theta-\phi$ space that represents the possible parameter-points reachable by cutting a slab with parameter-point at the origin. Similarly, we can also draw out a region which represents the possible original parameter-points of a slab that has already been cut and now has its parameter-point at the origin. Fig. 9 depicts these two regions, with the first one in blue, and the second one in red. We can see that the shape of these regions when projected onto the $z-\theta$ plane and $z-\phi$ plane are rhombuses. It is not hard to show that in the full 3D $z-\theta-\phi$ space, the shape will be an octahedron. Taking inspiration from the future/past light cones in special relativity, we will call these regions the future-cuttable octahedron (FCO) and the past-cuttable octahedron (PCO), respectively.

With this in mind, we can construct the repair decision map, i.e. a map that shows the region in which each repair method is optimal (no repair required, cutting, and raising). First, let's consider the $z-\theta$ plane. Given the ADA requirements (3)-(5), it is easy to find the region where, if the parameter-point of the second slab is located in that region, the second slab does not require any repairs. This region is depicted by the shaded parallelogram in Fig. 10. Call this region the "good" region, denoted by Ω. Call the region where, if the parameter-point of the second slab is located in that region, the second slab can be repaired by cutting, the "cuttable" region. Then, to find the cuttable region, we can exploit a common trick in special relativity for causality: find the union of all of the PCO on the boundary of the good region. Or in mathematical terms: for all $x_i \in \partial\Omega$, the region is given by

$$\bigcup_{x_i \in \partial\Omega} \mathrm{PCO}(x_i).$$

Fig. 10 shows the repair decision map with the shaded good region and cuttable region.

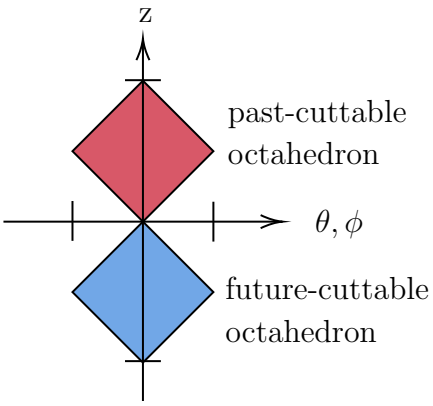


Figure 9: The region spanned by the parameter-points reachable by cutting a slab with parameter-point at the origin is marked in blue and is called the future-cuttable octahedron (FCO). The region spanned by the possible original parameter-points of a slab that has already been cut and now has its parameter-point at the origin is marked in red and is called the past-cuttable octahedron (PCO).

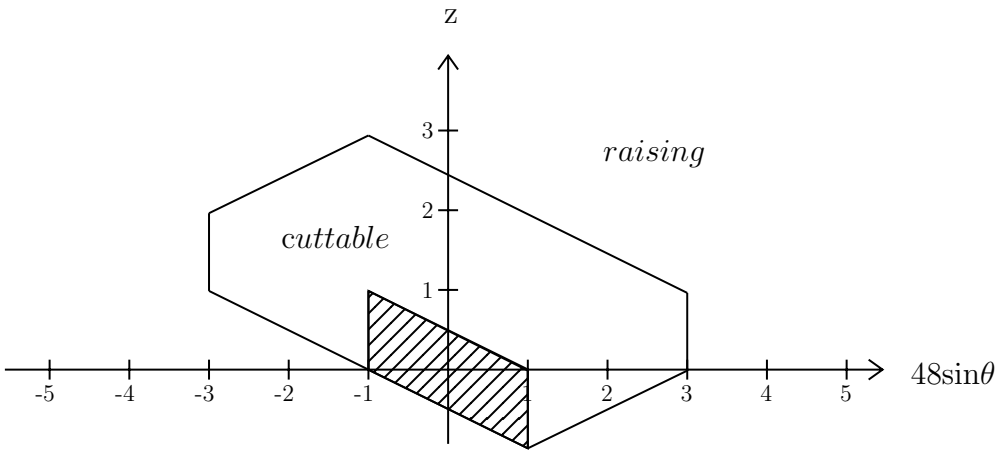


Figure 10: Repair decision map on the $z-\theta$ plane. The shaded region is the good region. The cuttable region is also shown. This area can be obtained by taking the union of all PCO that starts from the boundary of the shaded region.

Similarly, we can calculate the repair decision map on the $z-\phi$ plane shown in Fig. 11. Here, the good region is a hexagon. From the $z-\theta$ plane and $z-\phi$ plane repair maps, we can generate the full repair decision map in 3D as shown in Fig. 12. The good region is a slanted hexagonal prism as shown on the left side of Fig. 12. The cuttable region is the football-shape on the right side of Fig. 12.

Thus, the optimal strategy for fixing a two-slab street is outlined below:

1. Check for any cracked slabs. If a slab is cracked, replace it.
2. Plot the parameter-point of both slabs in $z-\theta-\phi$ space.
3. Draw out the repair decision map around both of the parameter-points.
4. If the parameter-point of one slab lies in the good region of the other slab, then no repair is

necessary.

5. If the parameter-point of one slab lies in the cuttable region of the other slab, then we cut the first slab.

6. If neither of parameter-points of the two slabs lies in each other's cuttable region, then we raise one of the slabs.

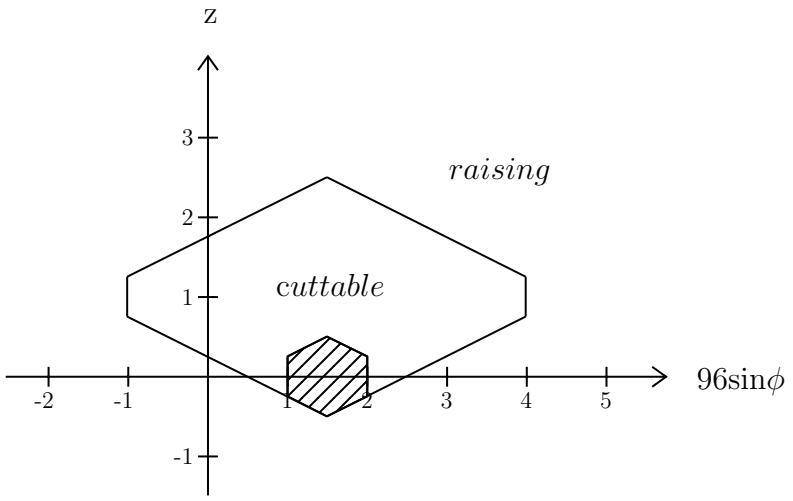


Figure 11: Repair decision map on the $z - \phi$ plane. The shaded region is where no repair is required. The region which can be fixed by cutting is shown. This area can be obtained by taking the union of all PCO that starts from the boundary of the shaded region.

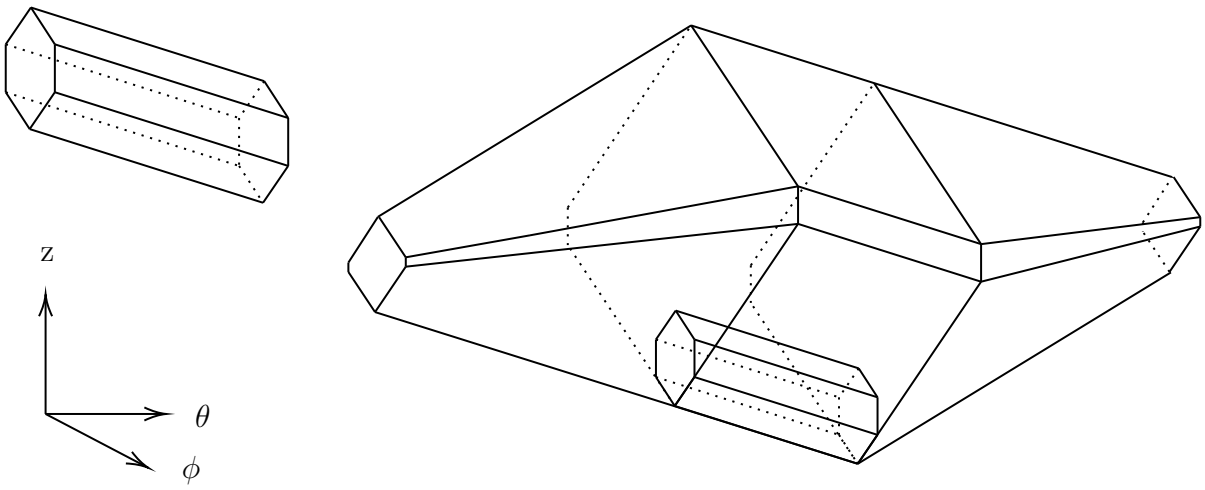


Figure 12: The full 3D repair decision map. **Left:** the good region. **Right:** the cuttable region. This region can be obtained by taking the union of all PCO that starts from the boundary of the embedded good region.

The pseudo-code for this algorithm is shown in Algo. 1. This algorithm will always allow us to use

the cheapest way to repair the slabs. Using this algorithm as a basis, we can generalize to the case of an arbitrary number of slabs.

Algorithm 1: Algorithm to determine the repair action for a slab i, given the good and cuttable regions of another slab j, assuming the slab is *not cracked*. If it is cracked, the only available action is to replace it with cost = 352. Cost is in US$ for one full slab.

```
function action (P_i, G_j, Ω_j);
  Input  : point P_i = (z_i, θ_i, φ_j), G_j the good region of point j, and Ω_j the cuttable region of
           point j
  Output: cost, action
cost := 0
action := None
if P_i ∈ G_j then
    cost := 0
    action := No action for block i
else if P_i ∈ Ω_j then
    cost := 64
    action := Cut block i so that P_i ∈ G_j
else
    cost := 82.08
    action := Raise block i so that P_i ∈ G_j
return cost, action
```

7.4 Generalized Algorithms for Arbitrary Slabs

Using the simplest-case algorithm established in Sec. 7.3, we propose two algorithms with complexity $\mathcal{O}(n^2)$ that will return the repair action, and its associated cost, on each slab on a street with an arbitrary number of slabs.

The first algorithm (Algo. 2) uses a sequential reference slab to generate the good region and the cuttable region. For all neighboring slabs, we use Algo. 1 to find the optimal action for the two slabs and add up the cost of a full iteration over all slabs on the street. Then, we loop through all possible starting points of our iterations to give the minimum cost.

The second algorithm (Algo. 3) uses an absolute fixed reference point to generate the good region and the cuttable region. For all slabs, we use Algo. 1 to find the optimal action for the slab with respect to the absolute reference slab and add up the cost of a full iteration over all slabs on the street. Then, we loop through all possible starting points of our iterations to give the minimum cost.

Algorithm 2: First algorithm to find the minimal solution for fixing slabs that are not cracked, in a sequential order. If a slab is cracked, replace it. G_j the good region of point j, and Ω_j is the cuttable region of point j. These can be calculated from point P_j through the process described in section 7.3

```
function sequential_referencing (P);
  Input  : P - list of slab points (z_i, θ_i, φ_i) that are not cracked
  Output: cost, sequence of action A
min_cost := ∞
min_action := { }
N := |P|
for i ∈ [0, N − 1] do
    cost := 0
    action_sequence := { }
    j := i
    while ∃ unchecked slabs do
        cost_temp, action_temp := action(P_{j+1}, G_j, Ω_j)
        cost := cost + cost_temp
        action_sequence := action ∪ action_temp
        j := next unchecked slab
    end
    if cost < min_cost then
        min_cost := cost
        min_action := action_sequence
    end
end
return min_cost, min_action
```

Algorithm 3: Second algorithm to find the minimal solution for fixing slabs that are not cracked by comparing it to a fixed, reference slab. If a slab is cracked, replace it. G_j the good region of point j, and Ω_j the cuttable region of point j. These can be calculated from point P_j through the process described in section 7.3.

```
function fixed_referencing (P);
  Input  : P - list of slab points (z_i, θ_i, φ_i) that are not cracked
  Output: cost, sequence of action A
min_cost := ∞
min_action := { }
N := |P|
for i ∈ [0, N − 1] do
    cost := 0
    action_sequence := { }
    for j ∈ [0, N − 1] AND j ≠ i do
        cost_temp, action_temp := action(P_j, G_i, Ω_i)
        cost := cost + cost_temp
        action_sequence = action ∪ action_temp
    end
    if cost < min_cost then
        min_cost := cost
        min_action := action_sequence
    end
end
return min_cost, min_action
```

8 Validity & Robustness Testing

8.1 Priority Score Perturbation

Since the result for clustering in Sec. 6 depends heavily on the priority score, we introduce a random perturbation of the priority score to see its effect on the clustering for optimal contracts. From our definition of priority in Sec. 5, these perturbations can be interpreted as follows:

1. An incorrect estimation of the population of each city block.
2. An incorrect estimation of the total number of slabs in each city blocks.
3. Any sort of incorrect estimation in determining the stress of slabs in a given city block, including perturbations from the second order effects (e.g. pedestrian-caused stress and flowing water-induced stress) and incorrect assumptions that the stress is uniform across the city.

The perturbation is introduced by the following formula:

$$P'_{ij} = P_{ij}(1 + \epsilon_{ij})$$

where P'_{ij} is the perturbed priority score at cell (i, j), P_{ij} is the original priority score, and ϵ_{ij} is a random variable drawn from a normal distribution with mean μ and standard deviation σ. We use three different values for σ: $\sigma = 0.01$, 0.05, 0.1. The perturbed priority score maps are shown in Fig. 13. We can see that as σ increases, the priority score map differs more from the unperturbed ($\sigma = 0$) map.

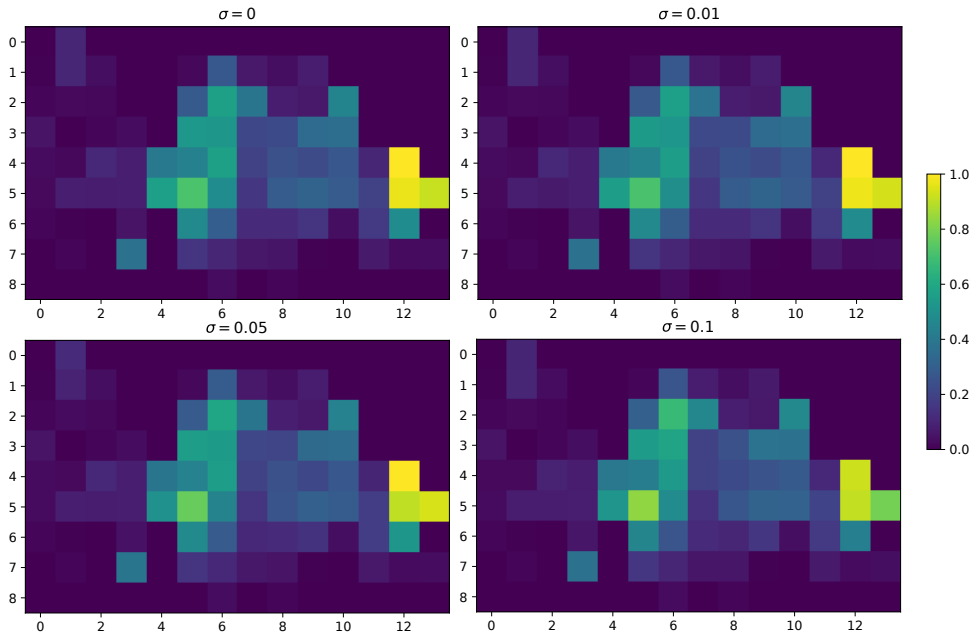


Figure 13: The perturbed priority score map with Gaussian noise of $\sigma = 0$ (unperturbed), 0.01, 0.05, and 0.1. We can see that as σ increases, the priority maps differs more from the unperturbed map.

We run the Gaussian mixture algorithm for clustering using these perturbed priority score maps. The result is shown in Fig. 14. We can see that the clustering is not affected by the given perturbation. This result shows that our clustering algorithm is a robust algorithm that can withstand perturbation up to $\sigma = 0.1$. This also justifies our first order approximation of the lifetime of a concrete slab in Sec. 5. We stated that the typical value of the second order effect of stresses is at least 2 orders of magnitude less than the primary source of stress (thermal stress). Our result of robustness analysis shows that the clustering result stays the same up to a perturbation of $\sigma = 0.1$. This indicates that even if we underestimate these second order effect by an order of magnitude, it is still safe to ignore these secondary effects.

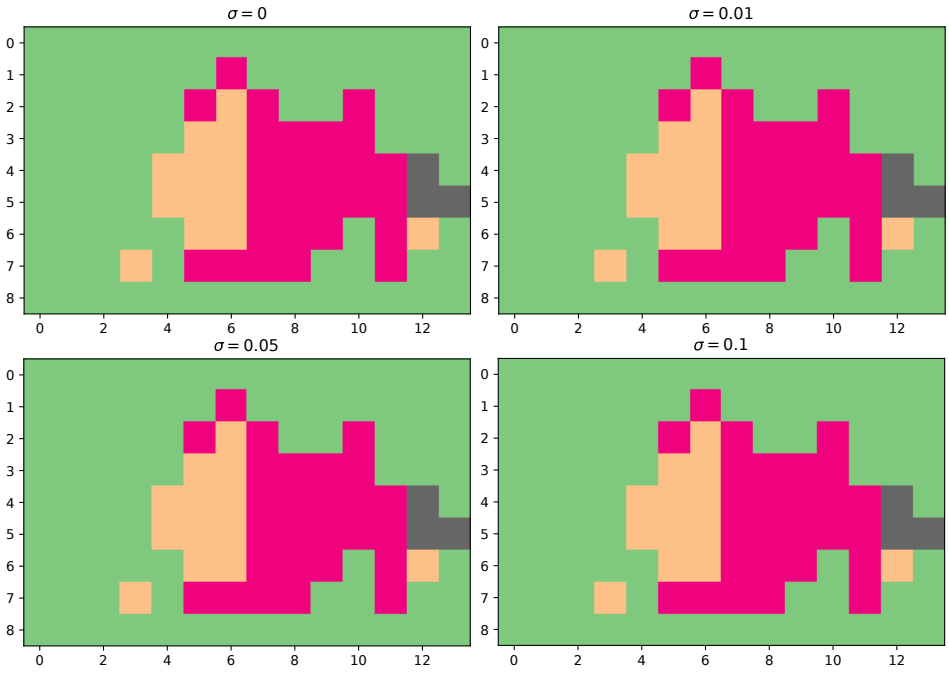


Figure 14: Result of clustering using Gaussian mixture with the perturbed priority score map. We can see that the clustering result is invariant under a perturbation of σ up to 0.1.

8.2 Weight of Clustering

As we stated in Sec. 6, we cluster city blocks in the $(i, j, P_{i,j})$ space. However, the units for (i, j) and P_{ij} are different. The units for i and j are distance, while the units for P_{ij} are arbitrary. If the units of the parameter space are not the same for clustering algorithm, then there is an additional degree of freedom which corresponds to the weights which we can assign to each parameter. In mathematical terms, we can choose any metric for the parameter space. Therefore, to study the effect of this additional degree of freedom on the clustering result, we alter the weight w of the priority score when the data set is passed into the clustering algorithm. We define the weight as the following: for any two points x, y in

the parameter space, the distance between x and y is given by:

$$D(x,y) = \sqrt{(i_x - i_y)^2 + (j_x - j_y)^2 + w^2(P_x - P_y)^2}$$

In other words, the weight w changes the importance of the priority score. Clustering with a larger weight value places more importance on the priority score.

Fig. 15 shows the resulting clustering for weights $w = 0, 10, 50, 100, 1000$. As expected, when $w = 0$, the clustering algorithm groups city blocks based solely on the physical distance. As w increases, the clustering becomes more and more dominated by the priority score. When $w = 100$, the clustering result is dictated primarily by the priority score.

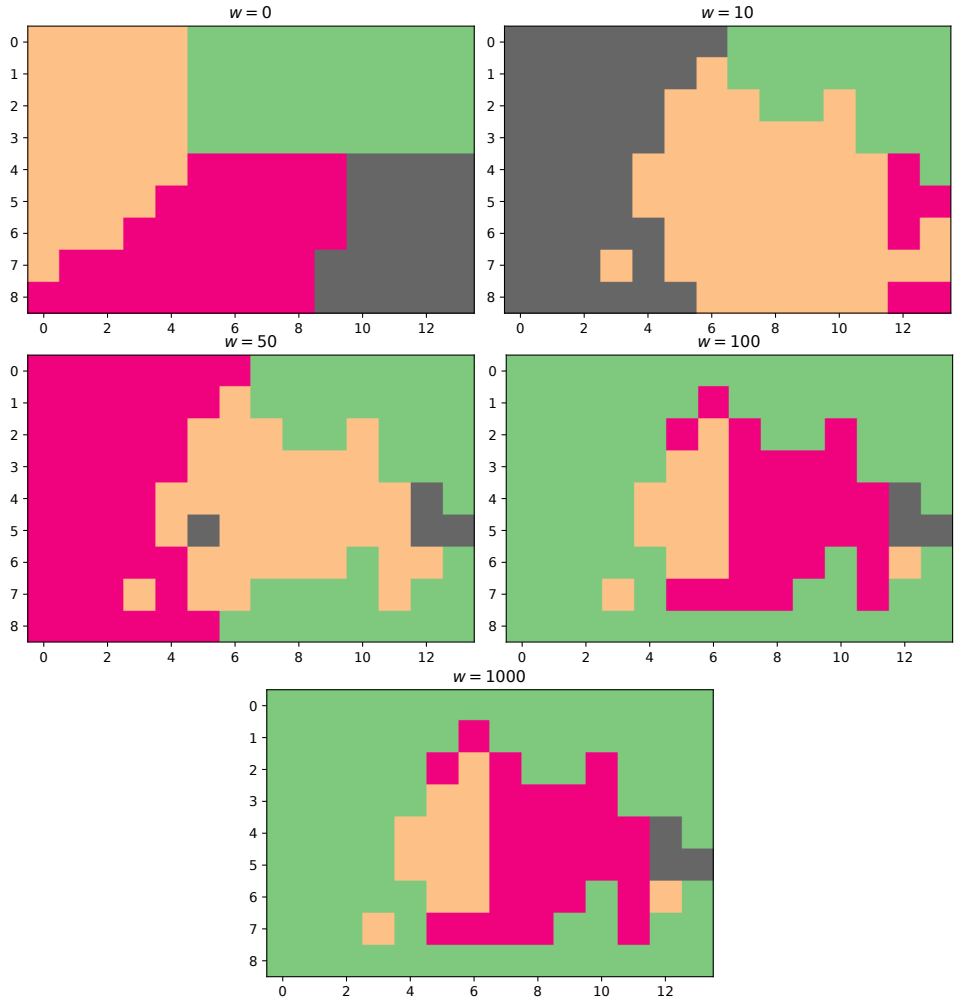


Figure 15: Gaussian mixture cluster results for different priority score weights.

8.3 Optimizing Slab Repair Strategy

The optimality of both Algo. 2 and Algo. 3 can be easily tested by considering all possible sequence of actions. For N slabs on a street, both algorithms only test N possible sequences and find the minimum

of case of those N sequences. However, there are 3 possible actions for all slabs: no repair, cutting, and raising. In addition, the action of raising can give any possible value in the $z-\theta-\phi$ space, and the action of cutting can give any value in the FCO, which is a subset of the $z-\theta-\phi$ space. Therefore, each of the 3^N sequence of actions are embedded in a 3^ND space. The naive way to search for the optimal solution requires a computation time of $3^N \cdot 3^{N/\delta}$ where δ is the resolution of the $z-\theta-\phi$ space. If we want a solution within a smaller tolerance of the true minimum, then we will need a higher resolution. However, the higher the resolution, the more computationally expensive the algorithm becomes.

On the other hand, our proposed algorithms are computationally cheap, even if not guaranteed to give the global minimum solution. Nevertheless, from our rigorous analysis in Sec. 7.3, our algorithms provide some degree of optimality.

9 Strengths & Weaknesses

9.1 Priority Score

The primary strength of our priority score analysis is that we use a physics-based analysis that has its basis in few assumptions about the various potential causes of slab stress. Through this physics-based analysis, we rigorously demonstrate our formalism and validity of our implementation. The validity of our implementation and analysis is further reinforced in Sec. 8.1.

Our priority score analysis does not have many weaknesses, but it has some limitations. The primary limitation is the failure to directly apply much of the data that was available and could have been used to more accurately determine the slab stresses across Ithaca. For example, we were able to find very detailed tree [8] and soil [6] data that, with more time, we could have incorporated into our model to produce a more accurate priority score map. However, these other effects must be very large in order to change our priority score map, as Sec. 8.1 shows.

9.2 Optimal Contracts and Clustering

The first and most important strength of using unsupervised clustering on the (i, j, wP_{ij})-space data is the semi-automation of the process. In this sense, we are able to combine the power of unsupervised clustering with the human ability to recognize clustering patterns. By using the (i, j, wP_{ij}) parameter space, we are balancing the weight of spatial closeness and priority. Thus, implicitly we are dividing work equally for the different construction crews. A "high-intensity" crew only has to cover a small region, whereas a "low-intensity" crew has to cover a larger area. Through testing two different methods, k-means and Gaussian mixture, we are able to select the unsupervised clustering method that gives the better result: Gaussian mixture. Through perturbation analysis, we see that a $\sigma = 0.1$ perturbation in priority score does not change the results of the Gaussian mixture clustering. Therefore, this method is robust, at least to our perturbation tests. It also achieves the goal of contiguous regions, thus minimizing the cost of construction.

The first weakness of our method of clustering is also its strength: using human modifications at the end. Thus, this algorithm's flow is not entirely automated. However, one can argue that this step is necessary to achieve the optimal solution. Another weakness of this method is its sensitive dependence

on the weight w. As seen above, a change in w can change the clustering results drastically. Thus, we had to determined w experimentally. Again, this is another weakness because the process for finding w is subjective and not automated.

9.3 Optimizing Slab Repair Costs

The most important strength of our analysis of the optimal strategy of slab repair is the rigorous geometric argument for finding the good region and cuttable region. Using the result from the simplest case of two slabs, we generalize the problem into an arbitrary number of slabs and propose two solution-finding algorithms that have complexity $\mathcal{O}(N^2)$.

The main weakness in the generalization of the two-slab case is the limited set of possible solutions that are probed in our algorithms when compared to the full set of possible solutions. Thus, our algorithms are not guaranteed to give the optimal solution, although it is significantly faster than the naive approach which tests every single possible sequence of actions.

10 Conclusions & Future Work

In this work, we present solutions to several important problems that are central to improving Ithaca sidewalks. First, we determine the factors affecting a sidewalk slab through physically motivated models. We arrive at simplifying results to determine that the characteristic time scale of the slab is a constant, with physical effects come in at the second order. From here, we determine the priority score map of Ithaca. This map is the goal we sought for the first problem.

Then, we seek to cluster the data on a three-dimensional space, balancing between spatial coordinates and priority score. We compare two methods of unsupervised clustering, and find that Gaussian mixture gives the best results. We use a human-assisted approach: human validation at the last step is necessary to optimize. Furthermore, we find contiguous regions for construction crews to travel. This help minimize the cost since there is no need to charge the extra $1000 for long-distance moving.

In the last problem, we constrain the solution space to the problem of raising and cutting slabs through a geometrically-based model inspired by concepts from special relativity. With this, we are able to determine the possible ways to raise or cut a slab in relation to another slab while also complying with the ADA. Armed with this solution space, we further present two $\mathcal{O}(n^2)$ algorithms to find approximated solutions to fixing sidewalks made up of N slabs.

Robustness analysis through perturbations of parameters, optimality of solutions, and discussion of strengths and weaknesses are also included. Here, we show that our models are fairly robust: when the priority map is perturbed up to $\sigma = 0.01$, the clustering result does not change. Furthermore, the algorithms presented to find solutions to fix an entire sidewalk are of polynomial-order time complexity, in contrast to the brute force time complexity of $\mathcal{O}(3^n)$.

There are many venues for potential improvement of our models. In the first problem, having data of trees around Ithaca in a more easily accessible format could help us model the effects of trees on specific slabs and specific streets. In addition, having data of soil quality and type on each street in a more easily accessible format would allow us to calculate the effects of stress due to the soil (e.g. differential thermal expansion of the soil, effects on tree root growth) and consequently their effects on the lifetime

of the sidewalk slabs. Given more time, we could have searched for a better solution-finding algorithm that guarantees a stronger minimality of cost and better computational complexity. Furthermore, a fully automated clustering algorithm that does not require manual alterations would be ideal for a more refined, automated streamline workflow. Lastly, having more data in general (e.g. average number of people at a bus stop over the course of a day, or greater resolution of Ithaca population data, etc.) would allow us to calculate solutions with better resolution. With these improvements, we could accurately cluster city blocks, and even propose the specific routes each construction crew should take!

References

[1] The Quikrete Companies. *Project: Concrete Sidewalks and Small Slabs*. [Online; accessed 17-November-2019].

[2] City of Ithaca. *Sidewalk Improvement Program*. [Online; accessed 17-November-2019]. URL: https://storymaps.arcgis.com/stories/4df2b76ade6a4171a5ccbed80bf47ceb.

[3] City of Ithaca. *Sidewalk Policy*. [Online; accessed 17-November-2019]. URL: https://www.cityofithaca.org/219/Sidewalk-Policy.

[4] Bald Eagle Construction & Permit Services. *Why Concrete Cracks (And What To Do About It)*. [Online; accessed 17-November-2019]. URL: https://baldeagleconstruction.com/why-concrete-cracks.

[5] E. Gillespie. *What Causes a Crack in a Sidewalk?* [Online; accessed 17-November-2019]. Sept. 2017. URL: https://www.gardenguides.com/info_12080018_causes-crack-sidewalk.html.

[6] United States Department Of Agriculture (USDA) and Cornell University Agricultural Experiment Station. *Soil Survey, Tompkins County NY, 1965*. [Online; accessed 17-November-2019]. 1965. URL: https://cugir.library.cornell.edu/catalog/cugir-007498.

[7] Centers for Disease Control and Prevention. *Anthropometric Reference Data for Children and Adults: United States, 2011–2014*. [Online; accessed 16-November-2019]. URL: https://www.cdc.gov/nchs/data/series/sr_03/sr03_039.pdf.

[8] City of Ithaca. *Ithaca Blooms*. [Online; accessed 17-November-2019]. URL: http://geo2.tompkins-co.org/html/?viewer=ithacablooms.

[9] W.F. Cassie. "The Fatigue of Concrete". In: *Journal of the Institution of Civil Engineers* 11.4 (1939), pp. 165–167.

[10] G.F. Carter and D.E. Paul. *Materials science & engineering*. ASM International, 1991.

[11] I. Yoshitake et al. "A Prediction Method of Tensile Young's Modulus of Concrete at Early Age". In: *Advances in Civil Engineering* 2012 (2012).

[12] N.H. Tran, K.D. Hall, and M. Jameson. "Coefficient of Thermal Expansion of Concrete Materials". In: *Journal of the Transportation Research Board* 2087.1 (2008), pp. 51–56.

[13] Western Regional Climate Center. *ITHACA CORNELL UNIV, NEW YORK – Climate Summary*. [Online; accessed 16-November-2019]. URL: https://wrcc.dri.edu/cgi-bin/cliMAIN.pl?ny4174.

[14] J. Guo and P.Y. Julien. "Shear Stress in Smooth Rectangular Open-Channel Flows". In: *JOURNAL OF HYDRAULIC ENGINEERING* 131.1 (2005), pp. 30–37.

[15] OpenStreetMap. *CensusViewer*. [Online; accessed 17-November-2019]. URL: http://newyork.us.censusviewer.com/client.

[16] D. Pham, C. Shiau, and K. Wadell. "The Flu Is Coming: A Case Study in Ithaca". In: *Vladimirsky's Math 3610 Fall 2019 Project 1* (Oct. 2019).

[17] *Ithaca City School District*. [Online; accessed 17-November-2019]. URL: https://www.ithacacityschools.org/.

[18] *National Center for Education Statistics: Ithaca Senior High School*. [Online; accessed 17-November-2019]. URL: https://nces.ed.gov/ccd/schoolsearch/school_detail.asp?Search=1&DistrictID=3615570&ID=361557001340.

[19] M. Smith. *Ithaca, Tompkins municipal employees highest paid in region*. [Online; accessed 17-November-2019]. Sept. 2016. URL: https://ithacavoice.com/2016/09/ithaca-tompkins-municipal-employees-highest-paid-region/.

[20] C. Bishop. *Pattern Recognition and Machine Learning*. 1st ed. Springer-Verlag New York, 2006.

A Python Code

```
# Dang Pham, Calvin Chen, William Xu
# CMCM 2019, 18 November 2019
# this code creates ALL the plots shown in the paper

import networkx as nx
import numpy as np
import osmnx as ox
import pandas as pd
from numpy import *
from matplotlib.pyplot import *
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture

ox.config(log_console=True, use_cache=True)
gmapkey = 'AIzaSyDlOlzIRJx-3Ihn4omh-CY2SDOlahSn0Es'

### DATA ###
#ithaca population map
ithaca_population_map = [
    [ 600,  600,    0,    0,    0,    0,    0,  600,  200,  600,  600, 1000, 1000,
      0],
    [ 200,  600,  200,    0,    0,  200,  600, 1000,  600,  600,    0,    0,    0,
      0],
    [ 200,  200,  200,    0,    0,  600, 1000, 1000,  600,  200, 1000,    0,    0,
      0],
    [ 600,    0,  200,  200,    0, 1000, 1000, 1000,  600,  600,  600,    0,    0,
      0],
    [ 600,  200,  600,  200,  600, 1000, 1000,  600, 1000,  600,  600,  600,  600,
      0],
    [ 200,  200,  200,  200,  600, 1000, 1000,  600, 1000, 1000, 1000, 1000, 1000,
   1000],
    [ 200,  200,    0,  200,    0, 1000, 1000,  600, 1000,  600,  200,  600,  600,
      0],
```

```
    [   0,  200,    0, 1000,    0,  600, 1000, 1000,  600,  200,    0,  600,  200, 1000],
    [   0,  200,    0,    0,    0,    0, 1000,    0,  600,  600,    0,    0,    0,  200]
]
TOTAL_GOV_EMPLOYEES = 303
TOTAL_GOV_BUILDINGS = 0

ITHACA_HIGH_SCHOOL_POP = 1360
ALL_OTHER_SCHOOLS_POP = 400

PER_BUS_STOP_POPULATION = 30
SLAB_LENGTH = 4 #ft
FT_TO_METER = 0.3048 #ft/meter

ithaca_map_x = 9
ithaca_map_y = 14

ithaca_map_r0 = (42.4613, -76.5216)
ithaca_map_r1 = (42.4278, -76.47)

### END OF DATA ###

### BEGIN MAIN CODE ###
# get the street network for Ithaca
place = 'Ithaca'
place_query = {'city':'Ithaca', 'state':'New York', 'country':'USA'}
G = ox.graph_from_place(place_query, network_type='walk', simplify=True)

# add elevation, calculate grade
G = ox.add_node_elevations(G, api_key=gmapkey)
G = ox.add_edge_grades(G)
edge_grades = [data['grade_abs'] for u, v, k, data in ox.get_undirected(G).edges(keys=True, data=True)]
G_proj = ox.project_graph(G)

# first plot, plot of ithaca
fig,ax = ox.plot_graph(G_proj)
fig.savefig('ithaca_streets.pdf')


# get all the buildings of Ithaca
government_buildings = ['townhall', 'courthouse', 'police', 'post_office', 'fire_station']
school_buildings = ['school']

amenities_list = government_buildings + school_buildings

tags = {
    'amenity' : amenities_list,
```

```
    'highway' : 'bus_stop'
}
place_name = 'Ithaca, New York, USA'
gdf = ox.pois_from_place(place=place_name, tags=tags)

for gov_building in government_buildings:
    TOTAL_GOV_BUILDINGS += len(gdf.loc[gdf['amenity'] == gov_building])
PER_GOV_POP = ceil(TOTAL_GOV_EMPLOYEES/TOTAL_GOV_BUILDINGS)

# get gov buildings
gov_df = []
for building in government_buildings:
    temp_df = gdf.loc[gdf['amenity'] == building]

    for index, row in temp_df.iterrows():
        if row['geometry'].type == 'Point':
            y = row['geometry'].x #Google maps convention, lat = y, lon = x,
   opposite of OSM convention
            x = row['geometry'].y
        elif row['geometry'].type == 'Polygon':
            y = mean(row['geometry'].exterior.coords.xy[0])
            x = mean(row['geometry'].exterior.coords.xy[1])
        else:
            raise ValueError
        gov_df.append({'type': building, 'name': None, 'x': x, 'y': y})
gov_df = pd.DataFrame(gov_df)

# get schools
school_df = []
for index, row in gdf.loc[gdf['amenity'] == 'school'].iterrows():
    if row['geometry'].type == 'Point':
        y = row['geometry'].x #Google maps convention, lat = y, lon = x, opposite
   of OSM convention
        x = row['geometry'].y
    elif row['geometry'].type == 'Polygon':
        y = mean(row['geometry'].exterior.coords.xy[0])
        x = mean(row['geometry'].exterior.coords.xy[1])
    elif row['geometry'].type == 'MultiPolygon' and row['name'] == 'Ithaca High
   School':
        x =  42.455992
        y = -76.498355
    else:
        raise ValueError
    school_df.append({'type': 'school', 'name': row['name'], 'x': x, 'y': y})
school_df = pd.DataFrame(school_df)

# get bus stops
bus_df = []
for index, row in gdf.loc[gdf['highway'] == 'bus_stop'].iterrows():
    if row['geometry'].type == 'Point':
```

```
        y = row['geometry'].x #Google maps convention, lat = y, lon = x, opposite of OSM convention
        x = row['geometry'].y
    elif row['geometry'].type == 'Polygon':
        y = mean(row['geometry'].exterior.coords.xy[0])
        x = mean(row['geometry'].exterior.coords.xy[1])
    else:
        raise ValueError
    bus_df.append({'type': 'bus_stop', 'name': None, 'x': x, 'y': y})
bus_df = pd.DataFrame(bus_df)

all_dfs = {'gov': gov_df, 'school': school_df, 'bus_stop':bus_df}
PER_BUS_STOP_POPULATION = 5
# Cell class for each Ithaca cell
class Cell:
    def __init__(self, r0, r1, night_population = 0, bus_stops = None, gov = None, schools = None):

        self.r0 = r0 #upper left corner coordinate r0 = (x0,y0)
        self.r1 = r1 #lower right corner coordinate


        self.night_population = night_population

        self.bus_stops = bus_stops
        self.gov = gov
        self.schools = schools

        if self.bus_stops != None and self.gov != None and self.schools != None:
            self.day_population = self.calculate_day_population()

    def calculate_day_population(self):

        total_school_population = 0

        for index, row in self.schools.iterrows():
            if row['name'] == 'Ithaca High School':
                total_school_population += ITHACA_HIGH_SCHOOL_POP
            else:
                total_school_population += ALL_OTHER_SCHOOLS_POP

        total_gov_population = len(self.gov)*PER_GOV_POP
        total_bus_stops_population = len(self.bus_stops)*PER_BUS_STOP_POPULATION

        self.day_population = total_school_population+total_gov_population+total_bus_stops_population

# make ithaca map and populate it with buildings, bus stops, and populations
ithaca_map = []
ithaca_delta_x = abs(ithaca_map_r1[0] - ithaca_map_r0[0])/ithaca_map_x
```

```
ithaca_delta_y = abs(ithaca_map_r1[1] - ithaca_map_r0[1])/ithaca_map_y

for i in range(ithaca_map_x):
    ithaca_map.append([])
    for j in range(ithaca_map_y):

        cell_r0 = (ithaca_map_r0[0] - i*ithaca_delta_x, ithaca_map_r0[1] + j*ithaca_delta_y) #upper left corner of the cell
        cell_r1 = (ithaca_map_r0[0] - (i+1)*ithaca_delta_x, ithaca_map_r0[1] + (j+1)*ithaca_delta_y) #lower right corner

        ithaca_map[i].append(Cell(night_population=ithaca_population_map[i][j], r0=cell_r0, r1=cell_r1))
ithaca_map = array(ithaca_map)
ithaca_day_pop = zeros((ithaca_map_x, ithaca_map_y))
for i in range(ithaca_map_x):
    for j in range(ithaca_map_y):
        cell_ij = ithaca_map[i,j]

        cell_ij_x0 = cell_ij.r0[0]
        cell_ij_x1 = cell_ij.r1[0]

        cell_ij_y0 = cell_ij.r0[1]
        cell_ij_y1 = cell_ij.r1[1]

        no_of_bus = 0
        no_of_schools = 0
        no_of_gov = 0
        for key, df in all_dfs.items():
            temp_df = []
            for _, row in df.iterrows():
                r = (row['x'], row['y'])
                if cell_ij.r1[0] < r[0] < cell_ij.r0[0]:
                    if cell_ij.r0[1] < r[1] < cell_ij.r1[1]:
                        temp_df.append({'type': row['type'], 'name': row['name'], 'x': row['x'], 'y': row['y']})
            temp_df = pd.DataFrame(temp_df)
            if key == 'bus_stop':
                no_of_bus = temp_df
            elif key == 'school':
                no_of_schools = temp_df
            elif key == 'gov':
                no_of_gov = temp_df

        cell_ij.bus_stops = no_of_bus
        cell_ij.schools = no_of_schools
        cell_ij.gov = no_of_gov
        cell_ij.calculate_day_population()
        ithaca_day_pop[i,j]=cell_ij.day_population

```

```
# plot the ithaca day population
fig, ax = subplots()
ignore_points = [(2,0),(3,0),(4,0),(5,0),(3,1),(4,1),(3,2),(4,2),(13,0),(13,1)
    ,(13,2),(13,3),(13,4),(12,1),(12,2),(12,3),(11,1),(11,2),(11,3)]
added_pop = (30000 - sum(ithaca_day_pop))/(ithaca_map_x*ithaca_map_y - len(
    ignore_points))
ithaca_day_pop = ithaca_day_pop + added_pop

for point in ignore_points:
    cell_ij = ithaca_map[point[1], point[0]]
    cell_ij.day_population = 0
    ithaca_day_pop[point[1], point[0]] = cell_ij.day_population
for point in ignore_points:
    scatter(point[0], point[1], c='r')

cbar = ax.imshow(ithaca_day_pop,vmin=0)
colorbar(cbar)
ax.set_xticks([0.5+i for i in range(14)])
ax.set_yticks([0.5+i for i in range(9)])
ax.set_xticklabels([i for i in range(14)])
ax.set_yticklabels([i for i in range(9)])
grid()
title('Ithaca Day Population')
savefig('day_population.pdf')
show()

# plot the ithaca night population
fig, ax = subplots()
cbar = ax.imshow(ithaca_population_map,vmin=0)
colorbar(cbar)
ax.set_xticks([0.5+i for i in range(14)])
ax.set_yticks([0.5+i for i in range(9)])
ax.set_xticklabels([i for i in range(14)])
ax.set_yticklabels([i for i in range(9)])
grid()
title('Ithaca Night Population')
savefig('night_population.pdf')
show()


#calculate the number of slabs in a cell
for cell in ithaca_map.flatten():
    cell.slabs = 0

lat_0 = ithaca_map[0,0].r0[0]
long_0 = ithaca_map[0,0].r0[1]
lat_1 = ithaca_map[8,13].r1[0]
long_1 = ithaca_map[8,13].r1[1]
lat_sep = (lat_0 - lat_1)/9
long_sep = (long_1 - long_0)/14
```

```
261
262 total_length = 0
263 all_slab_list = []
264
265 for u, v, k, data in ox.get_undirected(G).edges(keys=True, data=True):
266
267     lon1 = G.nodes[data['from']]['x']
268     lat1 = G.nodes[data['from']]['y']
269
270     lon2 = G.nodes[data['to']]['x']
271     lat2 = G.nodes[data['to']]['y']
272
273     grade_abs = data['grade_abs']
274
275     orig_node = data['from']
276     target_node = data['to']
277
278     length_in_meter = data['length']
279     length_in_feet = length_in_meter / FT_TO_METER
280
281     total_length += length_in_feet
282
283     no_of_slabs = length_in_feet/SLAB_LENGTH
284     all_slab_list.append(no_of_slabs)
285
286     coordinate_1 = (lat1, lon1)
287     coordinate_2 = (lat2, lon2)
288
289     ind_x1 = (lat_0 - lat1)//lat_sep
290     ind_y1 = (lon1 - long_0)//long_sep
291     ind_x2 = (lat_0 - lat2)//lat_sep
292     ind_y2 = (lon2 - long_0)//long_sep
293
294     same_cell = np.array([ind_x1-ind_x2,ind_y1-ind_y2])
295     if ind_x1 <= 8 and ind_x2 <= 8 and ind_y1 <= 13 and ind_y2 <= 13:
296         if np.all(same_cell == 0):
297             ithaca_map[int(ind_x1),int(ind_y1)].slabs += no_of_slabs
298         elif sum(abs(same_cell)) == 1:
299             if abs(same_cell[0]) ==1:
300                 max_x = max(ind_x1,ind_x2)
301                 sep = ithaca_map[int(max_x),int(ind_y1)].r0[0]
302                 ithaca_map[int(ind_x1),int(ind_y1)].slabs += no_of_slabs*abs((lat1
    - sep)/(lat1-lat2))
303                 ithaca_map[int(ind_x2),int(ind_y2)].slabs += no_of_slabs*abs((lat2
    - sep)/(lat1-lat2))
304             elif abs(same_cell[1]) ==1:
305                 max_y = max(ind_y1,ind_y2)
306                 sep = ithaca_map[int(ind_x1),int(max_y)].r0[1]
307                 ithaca_map[int(ind_x1),int(ind_y1)].slabs += no_of_slabs*abs((lon1
    - sep)/(lon1-lon2))
```

```
                ithaca_map[int(ind_x2),int(ind_y2)].slabs += no_of_slabs*abs((lon2 - sep)/(lon1-lon2))

ithaca_day = zeros((ithaca_map_x, ithaca_map_y))
ithaca_night = zeros((ithaca_map_x, ithaca_map_y))
ithaca_slabs = zeros((ithaca_map_x, ithaca_map_y))

for i in range(ithaca_map_x):
    for j in range(ithaca_map_y):
        ithaca_slabs[i,j] = ithaca_map[i,j].slabs
        ithaca_day[i,j] = ithaca_map[i,j].day_population
        ithaca_night[i,j] = ithaca_map[i,j].night_population

ithaca_priority = ithaca_slabs*(ithaca_day + ithaca_night)
# plot the ithaca priority score map
fig, ax = subplots()
cbar=ax.imshow(ithaca_priority/np.max(ithaca_priority),vmin=0,vmax=1)
colorbar(cbar)
ax.set_xticks([0.5+i for i in range(14)])
ax.set_yticks([0.5+i for i in range(9)])
ax.set_xticklabels([i for i in range(14)])
ax.set_yticklabels([i for i in range(9)])
grid()

fig.suptitle('Ithaca Priority Score')
fig.savefig('ithaca_priority.pdf')


# perform clustering
# make training data for clustering
training_data = []
for i in range(ithaca_map_x):
    for j in range(ithaca_map_y):
        training_data.append((i,j,100*ithaca_priority[i,j]))

# k-means first
kmeans = KMeans(n_clusters=4, random_state = 10).fit(training_data)
imshow(reshape(kmeans.labels_, (ithaca_map_x,ithaca_map_y)),cmap='Accent')
savefig('kmeans.pdf')

# now gmm
gmm = GaussianMixture(n_components=4, random_state=10).fit_predict(training_data)
imshow(reshape(gmm, (ithaca_map_x,ithaca_map_y)),cmap='Accent')
savefig('gmm_before.pdf')

# gmm with human editing
gmm = GaussianMixture(n_components=4, random_state=10).fit_predict(training_data)
gmm = reshape(gmm, (ithaca_map_x,ithaca_map_y))
gmm[6,12] = gmm[5,12]
gmm[6,3] = gmm[7,3]
```

```
imshow(gmm,cmap='Accent')
savefig('gmm_after.pdf')

# perturbations of results

# plot the priority map for noise perturbations
for snr in [0,0.01,0.05,0.1]:
    ithaca_priority_noise = ithaca_slabs*(ithaca_day + ithaca_night)*(1+random.normal(loc=0, scale=snr, size=(ithaca_map_x, ithaca_map_y)))
    imshow(ithaca_priority_noise,vmin=np.min(ithaca_priority), vmax=np.max(ithaca_priority))
    title(r'$\sigma=' + str(snr) + '$')
    savefig('priority_map_snr_{}.pdf'.format(snr))
    show()

# gmm clustering for noise perturbations in the priority score map
for snr in [0,0.01,0.05,0.1]:
    training_data_noise = []
    ithaca_priority_noise = ithaca_slabs*(ithaca_day + ithaca_night)*(1+random.normal(loc=0, scale=0, size=(ithaca_map_x, ithaca_map_y)))
    ithaca_priority_noise = ithaca_priority_noise/np.max(ithaca_priority_noise)
    for i in range(ithaca_map_x):
        for j in range(ithaca_map_y):
            training_data_noise.append((i,j,100*ithaca_priority_noise[i,j]))

    gmm_noise = GaussianMixture(n_components=4, random_state=10).fit_predict(training_data_noise)
    gmm_noise = reshape(gmm_noise, (ithaca_map_x,ithaca_map_y))
    imshow(gmm_noise,cmap='Accent')
    title(r'$\sigma=' + str(snr) + '$')
    savefig('gmm_snr_{}.pdf'.format(snr))
    show()

# changes in weights of priority map
ithaca_priority_scaled = ithaca_priority/np.max(ithaca_priority)
for w in [0, 10, 50, 100, 1000]:
    training_data_weights = []
    for i in range(ithaca_map_x):
        for j in range(ithaca_map_y):
            training_data_weights.append((i,j,w*ithaca_priority_scaled[i,j]))
    gmm_weights = GaussianMixture(n_components=4, random_state=10).fit_predict(training_data_weights)
    gmm_weights = reshape(gmm_weights, (ithaca_map_x,ithaca_map_y))
    title(r'$w={}$'.format(w))
    imshow(gmm_weights, cmap='Accent')
    savefig('gmm_w_{}.pdf'.format(w))
    show()


#### END ######
```