Cementing a Path to Sidewalk Improvement

Carina Shiau, Leo Huang, Derek Lim

November 18, 2019

Abstract

We formulate **B) Optimal Contracts** as a clustering problem, in which the goal is to partition work sites into clusters of uniformly small radius, or clusters whose average radius is small—so as to minimize transition cost. To this end, one linear and two quadratic semi-assignment programs were considered. For **C) Optimal Repair Procedures**, we developed a series of three Bellman equations of increasing complexity, the last of which was used to optimally solve the discretized n-slab problem, as parameterized by $\{h_j, \theta_j, \xi_j\}_{i=0}^{n-1}$, the slab height, run slope, and cross slope. The model was validated using random initial configurations generated using Markov chains with Gaussian conditional distributions over three classes of terrains (hill, flat, and incline). For **D) Projecting Future Needs**, we modeled the birth-death cycle of slabs and their probability of breaking in the face of natural and man-made forces: carbonation, freeze-thaw, and rising contract costs. Projections were obtained using Monte Carlo simulation, and a comprehensive budget plan for the next 25 years is put forth.

Contents

1 Letter to Mr. Licitra

108 East Green Street
Ithaca, NY 14850
Dear Mr. Licitra,

As residents of the City of Ithaca, we appreciate your commitment to ensuring the safety, walkability, and long-term health of Ithaca's sidewalks. After studying your Sidewalk Improvement Plan (SIP), we have come up with a selection of models for improving various facets of the program. In particular, we focused on devising solutions for 1) designing optimal contracts, 2) refining repair procedures, and 3) projecting future needs.

Practically speaking, sidewalk repair involves transporting machinery and equipment from one location to another. Costs of transition—expenses incurred in moving from one site to another (faraway) site—should be systematically minimized. It is therefore important to prioritize not only individual blocks, but also clusters of blocks. We propose a method to design optimal contracts with respect to fixed objectives. Our methods deliver

- Smart contracts
- Fair contracts
- Customizable contracts

In particular, we propose a 6-piece contract for the proposed 2020 Sidewalk Improvement District (SID) work plan, where each sub-contract can be granted to a single construction crew, and where each job can be completed in a timely manner—and without racking up $\$1{,}000$s in transition costs, assuming a 1-km radius cluster of sites is easily navigable.

When it comes to repairing a long stretch of sidewalk, it is unclear how to do it cheaply. Which slabs should be elevated, and which slabs should be cut? Given the multitude of sidewalk design regulations, we found that the problem was best tackled *algorithmically*. Our algorithms can be adapted to various slopes and terrains—including Ithaca's own hilly terrain— to produce the cheapest solutions to sidewalk repair. On a test involving 78 slabs, our method achieved a cost of $\$17{,}251.73$, a remarkable improvement over the baseline cost of $\$41{,}184$.

Projecting budget increases is of central importance to the continued success of SIP, because being able to anticipate future needs is the first step in preparing to meet those needs. While many factors threaten to forestall the progress of SIP—including increasing atmospheric CO_2 levels, rising global temperatures, as well as flat revenues and rising costs—we can combat these factors by expanding our budget in a measured way. Through extensive research and simulations, we have come up with a budget plan which increases the budget by degrees from 800k to $1{,}500$k over a period of 25 years starting in 2020.

We hope you enjoy reading our proposal, and ultimately find it to be of use.

Sincerely Yours,

Carina, Derek, and Leo

2 Introduction

In 2014, the newly established City of Ithaca Sidewalk Improvement Program assumed the mantle of maintaining Ithaca's sidewalks. Since then, it has overhauled the sidewalk repair policy — freeing abutting property-owners from sole responsibility and paving the way for a greater-encompassing five-district system. The program has improved well over 100 blocks of sidewalks and over 70,000 linear feet of sidewalks. Its impact on Ithaca's walkability is tangible. Even with its impressive track record, however, there are many avenues for further enhancement, including designing more optimal contracts, optimizing repair procedures, and making more informed and accurate projections of future needs.

When designing contracts for sidewalk repair operations, it pays to prioritize groups of nearby blocks so as to reduce transition costs. We frame the problem of designing optimal contracts as a clustering problem: given a set of top-priority work-sites, how does one partition them into clusters, each of which can be worked on independently? The clusters should contain sites that are in close proximity to each other, so that construction crews do not need to pick up the $1000 transition cost. Using various measures of cluster size/radius, we construct three different mathematical programming problems to perform clustering and analyze their strengths and weaknesses.

Repairing a stretch of sidewalk is no simple task because individual slabs have to satisfy global and local constraints. Neighboring slabs need to satisfy compatibility conditions—the vertical displacement at their interface cannot be too large. The slope of individual slabs need to align closely to the running slope of the road. Cross-slope is constrained to a narrow interval. Broken slabs require replacement, and each repair procedure (cutting, raising, replacing) has its own cost and limitations. It is easy to repair a single slab, but difficult to see which choice of action leads to a globally optimal solution, that is, one which minimizes the total cost of repairing all slabs. To simplify the problem, we discretized the phase-space of each slab and broke the problem down into subproblems, which could be solved using dynamic programming. We formulate and solve three Bellman-esque dynamic programming problems, successively adding layers of complexity—each of which increases either the degree of compliance or dimensionality of the model.

The future of the Sidewalk Improvement Program (SIP) rests squarely upon consistent revenue flows. A budget plan is needed to account for all possible contingencies, including new installation proposals, rising contract costs, and natural aging of concrete. In modeling the aging process of concrete, we researched the effects of rising global temperatures and increasing atmospheric CO_2 levels on the rate of advancement of the carbonation front inside a concrete slab, as well as the number of yearly freeze-thaw cycles. We incorporated these factors into a stochastic model, which was run to obtain the 25-year budget plan.

3 Optimal Contracts

Given a selection of top-priority work sites sprawled across the sidewalk improvement districts, how can one optimally group them into manageable clusters—each of which can be worked on by a single construction crew without incurring expensive transition costs? Of course, if all the assignments are given to a single construction crew, then there is no avoiding transition costs. However, if multiple construction crews can be deployed, or if multiple contractors can be hired, then it is desirable to strategically assign tasks—henceforth defined as clusters of blocks—to different construction crews so as to minimize transition costs. We propose algorithms for grouping nearby blocks into clusters, which can be readily assigned to separate task forces.

3.1 Assumptions

- We assume that the blocks have already been triaged (assigned scores according to their priorities)
- Blocks deemed to have top priority have equal priority
- We assume that local transition costs are negligible, while long-distance transitions should be avoided unless absolutely necessary. It is plausible that construction tasks have an epicenter of activity, in a neighborhood of which tools and equipment are accessible, work zones are delineated, and traffic is blocked off (if necessary). Uprooting such an operation and relocating to a faraway location believably carries a high penalty. We do not assign hard definitions to *local* and *long-distance* for lack of expedient data.
- We assume in an example that in sufficiently small cluster (of intra-cluster distance at most 1 km), transition costs vanish

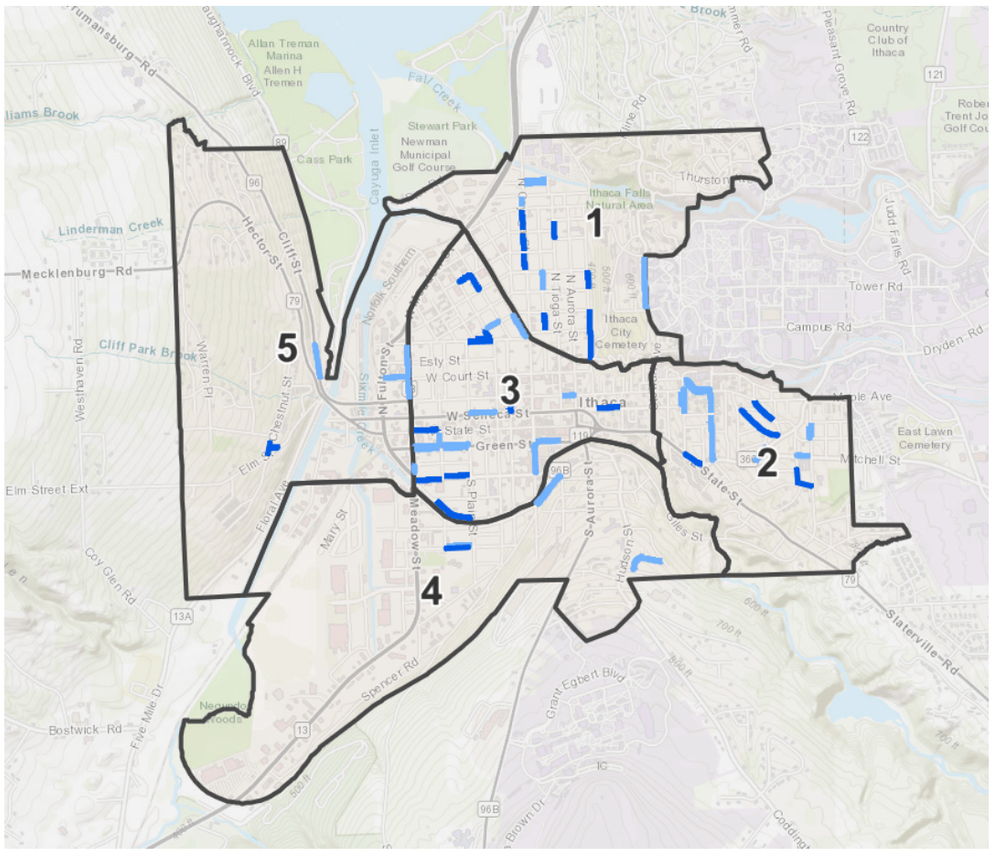

Figure 1: Proposed sidewalk work plan for 2019 (colored dark blue) and 2020 (light blue)

- We assume that multiple construction crews are deployed over some period of time to repair sidewalks. Although most construction work is assigned to a single contractor by means of public bid, "smaller contracts have also been used to do concrete cutting and raising/leveling" [18]. This suggests that multiple contracts are handed out–and as such, multiple construction crews are dispatched at different times.

3.1.1 Notation

Below are the definitions of variables used in the statements of the mathematical programming problems to follow.

N = number of work-sites
M = number of clusters of work-sites
x_{ik} = binary variable expressing whether site i is part of cluster k
d_{ij} = distance between sites i and j
n_k = number of sites per cluster

3.2 First Model: Minimax of Cluster Diameter (LP)

Our first mathematical program aims to minimize the largest within-cluster distance between sites. A formulation due to [1] is reproduced below.

$$\begin{aligned} \min D \\ d_{ij}x_{ik} + d_{ij}x_{jk} - D \leq d_{ij} \\ i \in \{1, 2, ..., N-1\} \\ j \in \{i+1, i+2, ..., N\} \\ k \in \{1, 2, ..., M\} \\ \sum_{k=1}^{M} x_{ik} = 1, i \in \{1, ..., N\} \\ x_{ij} \in \{0, 1\}, D \in \mathbb{R}^{+} \end{aligned}$$

This objective function is motivated by the fact that we want to limit the sizes of the clusters, so that intra-cluster movement is not burdened by transition costs of $1000 or more. Since moving between work sites that are far away from each other costs substantially more than moving between work sites not as far from each other, this minimization of the largest distances aligns well with

our problem of choosing optimal contracts. The variable D above is an upper bound on the largest intra-cluster pairwise distance, which is why the model seeks to minimize it. This objective gives rise to clusters of uniformly small "radius", or maximal intra-cluster distance.

3.3 Second Model: Quadratic Semi-Assignment Problem (QSAP)

This model is a typical quadratic semi-assignment problem – so called because we don't enforce the constraint $\sum_k x_{ik} = 1$. Relaxing this constraint is what makes the problem a clustering problem: the objects indexed by $I = \{1, ..., N\}$ are assigned to clusters $J = \{1, ..., M\}$, where each object is assigned to exactly one site ($\sum_i x_{ik} = 1$). The model aims to reduce the sum of within-group distances. We will see that a trademark of this loss function is that sites are split quite evenly over the clusters; the objective gives rise to clusters of roughly equal size. This could be a desirable property in assigning contractors, since a construction crew is only able to handle a fixed length of sidewalk a week and thus splitting the workload would allow more work to be done in a given time period. The reason that the objective function favors even clusters is because in larger clusters, there are more pairwise distances, so naturally the sum becomes larger.

$$\min \left(\sum_{k=1}^{M} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} d_{ij} x_{ik} x_{jk} \right)$$
$$\sum_{k=1}^{N} x_{ik} = 1 \ \forall \ i \in \{1, ..., N\}$$
$$x_{ik} \in \{0, 1\} \ \forall i, k$$

The quadratic program above is borrowed from [2]. When x_{ij} and x_{jk} are both 1, that means sites i and j are in the same cluster, so the distance between the sites d_{ij} contributes to the loss function. The other constraints are the same as before.

3.4 Third Model: Sum of Averaged Within Group Squared Distance

In many scenarios it makes more sense to consider the sum of average within-group distances, as opposed to the sum of the sum of within-group distances. The classical formulation of this problem is a fractional quadratic integer program, which is difficult to solve in general [1]. To work around this, one can pre-specify not only the number of clusters, but also the number of sites to be assigned to each cluster. This relaxes the fractional program to a quadratic integer program.

Although specifying the n_k requires additional user-intervention, it could potentially be used to take into account contractor supply – a contractor may be unable to satisfy demands in a timely manner due to limited manpower. The bidding procedure could be modified, so that contractors bid for certain amounts of demand. This allows for greater flexibility and customizability of the model.

$$\min \sum_{k=1}^{M} \frac{1}{n_k} \left(\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} d_{ij}^2 x_{ik} x_{jk} \right)$$
$$\sum_{k=1}^{M} x_{ik} = 1, i \in \{1, 2, ..., N\}$$
$$\sum_{i=1}^{N} x_{ik} = n_k \forall k \in \{1, 2, ..., M\}$$
$$\sum_{k=1}^{M} n_k = N$$
$$x_{ik} \in \{0, 1\} \ \forall i, k$$

In this model, we use distance squared instead of distance, in accordance with [1]. This puts a relatively higher penalty on large clusters.

3.5 Experiments: Contracts for Year 2020

We took the map (see Figure 1) depicting future work plans from the Sidewalk Improvement Program website [19, 20] and assigned pixel-based coordinates to each site. This was accomplished

using **ImageJ**, a high-precision image-processing program. To find the scale of the map, we determined the ratio between the actual distance from Site 1 (211 Cliff Street) and Site 20 (598-500 Mitchel Street) and the pixel distance.

$$\frac{\text{actual distance}}{\text{pixel distance}} = \frac{3.118 \text{ km}}{444.2} = 7.0194 \times 10^{-3}$$

If we assume that transition costs of $1000 or more can only be avoided when the maximum intra-group distance in a cluster is at most 1.00 km, then the first model (minimax cluster diameter) tells us that at least 6 clusters are needed (see Figure 3). This assumption is reasonable, because Ithaca is approximately 4 km wide, and 1 km is roughly the distance from the centers of the five SIDs. Our assumption, then, roughly becomes: a *faraway* site is one in another SID. Of course, sites situated near the border of neighboring SID are very close together, but given the problem of exhaustively clustering all sites, cut-off decisions must be made.

We used `Gurobi MIP` (mixed integer programming) software to solve the linear programming problem in model 1 to obtain a clustering for the 2020 data.

3.6 Results for Year 2020

Table 1 shows how the minimum maximum intra-group distance d^* for clusters decreases with the number of clusters. d^* first falls below 1.00 km when the number of clusters is 6. We elected to display the results for model 1 in this section because Model 1 gives a hard bound on the max within-group distance. Comprehensive results for models 1, 2, and 3 are shown in the appendix.

Table 1: Model 1 Results

Clusters	Distance (km)	clusters	Distance (km)
2	2.35	7	0.81
3	1.63	8	0.80
4	1.18	9	0.74
5	1.11	10	0.55
6	0.90	11	0.50

3.7 Discussion

We found algorithm 1 to be most suitable for the problem of minimizing transition costs, because by increasing M (the number of clusters) one step at a time, we were able to identify the minimum M which guarantees that the optimal clustering has the property that the maximum cluster "radius" is under a fixed threshold (in our case 1 km). The other models, while they were effective in minimizing some fixed objective, could not guarantee such a hard bound.

Regarding **weaknesses** of our model, we only experimented with clustering work sites into tight groups and did not explore assigning priority scores to these clusters of work sites, nor did we take into account the number of contractors that might be available.

In Model 3, it behooves the user to pre-set the number of work sites per cluster. In defense of the framework, however, when N is small, the number of feasible partitions isn't large, and experimenting with various partitions using the code provided is easy. The user can try different partitions until a satisfactory configuration is attained, or if some minimum intra-cluster distance is desired, switch to Model 1. On the flipside, the flexibility to divvy up the sites/blocks into groups of prescribed sizes allows the user to exert greater control over the end result.

On the subject of **validation**, a quick glance at the clustering results (Figures 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27) will ascertain that the results generated by the code are within reason.

We could overcome shortcomings of this model in **future** work. Hierarchical clustering methods would allow for clusterings with different numbers of clusters to be considered at once. This may be useful, as contractor supply and demand forces could be accounted for in the model instead of assuming some fixed number of contractors are available and have to be assigned blocks. Also, our current models naturally allow for different distance metrics between blocks to be used. For instance, it could be more realistic to use driving distance along roads, or it could be more suitable to penalize clustering far-away blocks together with some other metric.

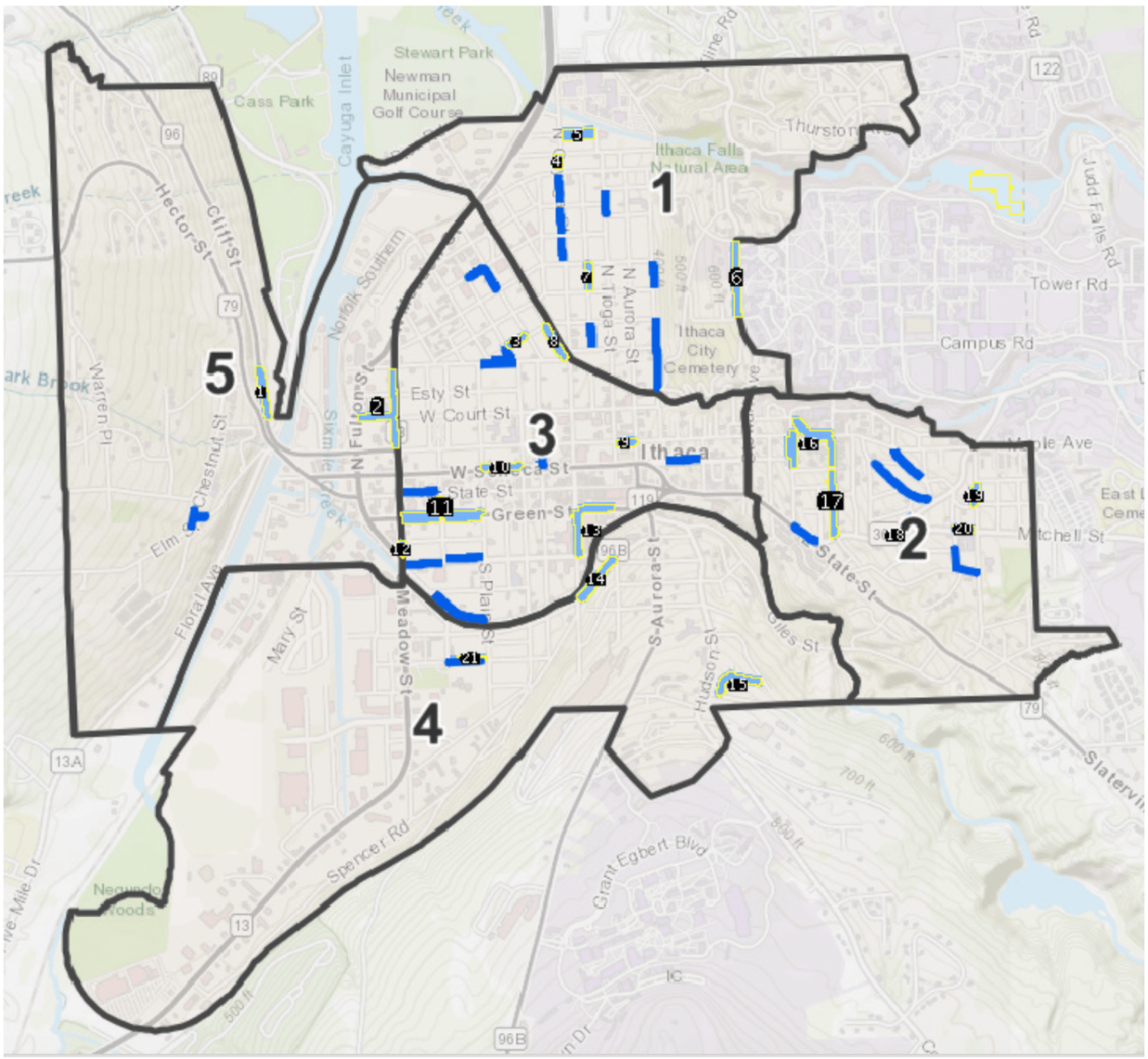

Figure 2: The priority 2020 sidewalk repair sites (colored in light-blue) were singled out and numbered from 1 to 21 using ImageJ software. The dark blue sites were already repaired in 2019, and therefore were excluded from experiments. This map was retrieved from [20] and used for testing all clustering algorithms.

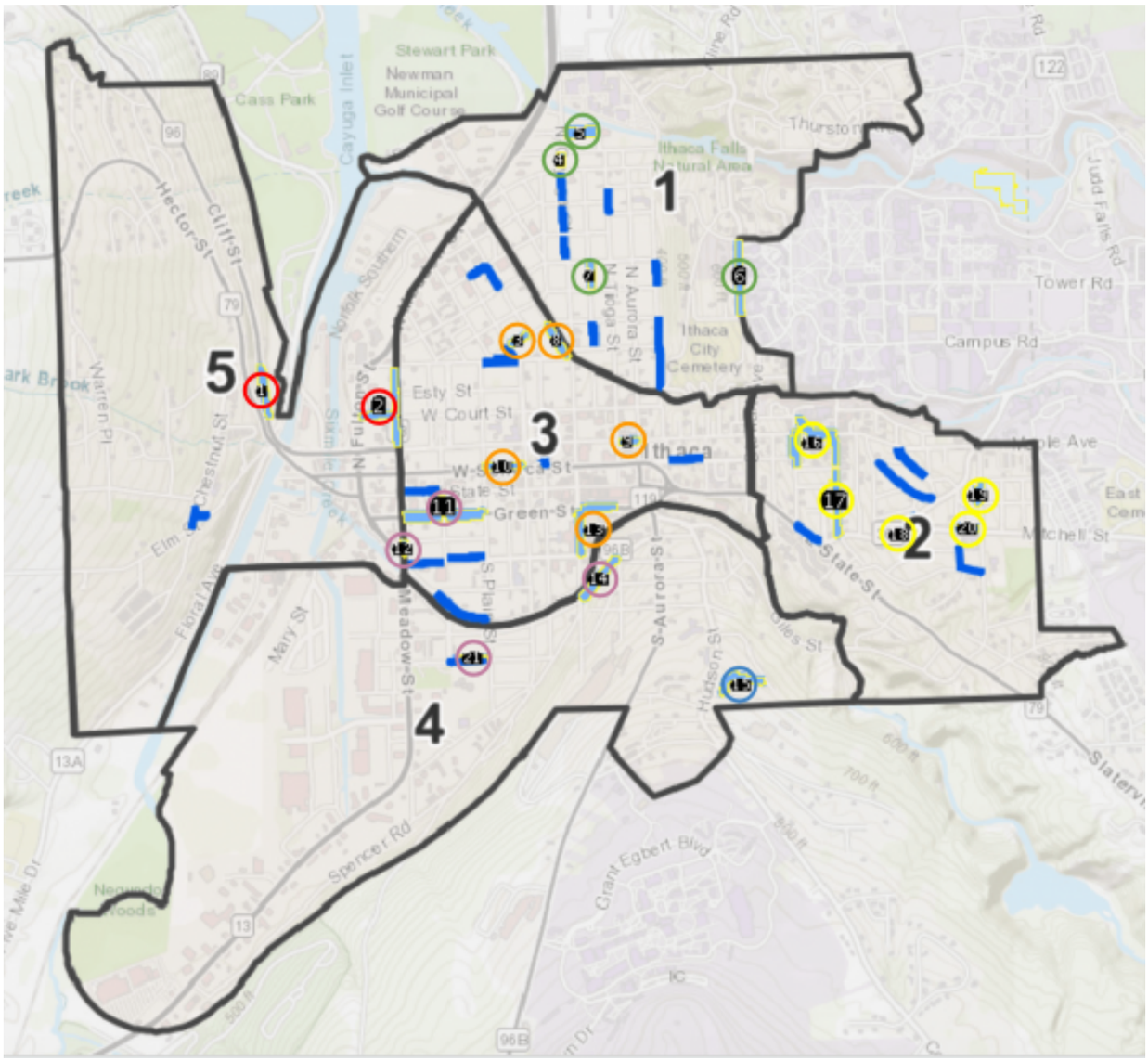

Figure 3: Site 15 (shown in blue) is singly clustered by the model 1 likely because it is geographically isolated. If a single construction crew is dispatched to do repairs at site 15 *in addition* to other sites, then a high transition cost is potentially unavoidable. Note that although $d(15, 14) = 0.79$ km, grouping site 15 with the other purple sites, for example, would raise the minimum maximum within-group distance well above 1.00 km. Other sites are tightly clustered – we observe that the partitions are roughly aligned with district lines. We term this map the "6-piece contract", because it uses 6 clusters to eliminate expensive transition costs.

4 Optimal Repair Procedures

We first give a basic summary of our three-tiered approach. Each of the models uses dynamic programming to determine a choice of actions of minimal cost that repairs all slabs in a block and places them in a valid configuration. The allowable actions are leaving a slab as is, raising a slab, cutting a slab, and replacing the whole slab. In the first model, we only consider the vertical positions of each slab. As such, we found it convenient to model the slabs as rectangles in the plane, each with their long side parallel to the x-axis. The DP algorithm associated with the first model runs exceptionally fast, as it does not consider run or cross slopes.

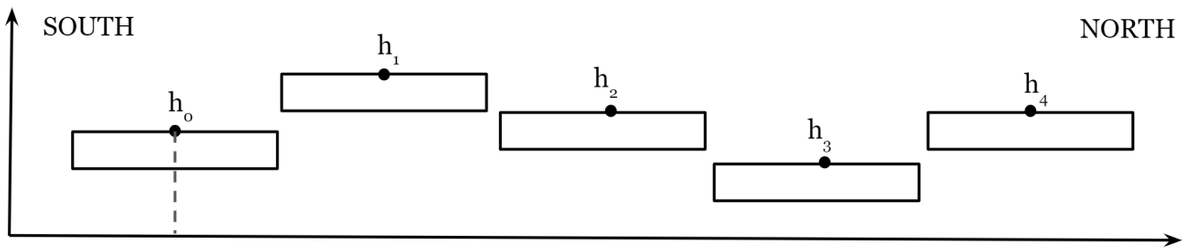

Figure 4: In model 1, each slab only has one positional coordinate: the height h_j. We ignore cross-slope and running slope for the time being, focusing instead on the underlying DP problem. For ease of reference, we assume the slabs run from south to north and are ordered as such.

We augment the first model by incorporating run slopes. Then we can no longer assume the slabs are all parallel. This means that any valid configuration has each slab's south endpoint compatible with the previous slab's north endpoint, and each slab's north endpoint compatible with the next slab's south endpoint. Moreover, each slab run slope must be close to the run slope of the segment of road next to that slab.

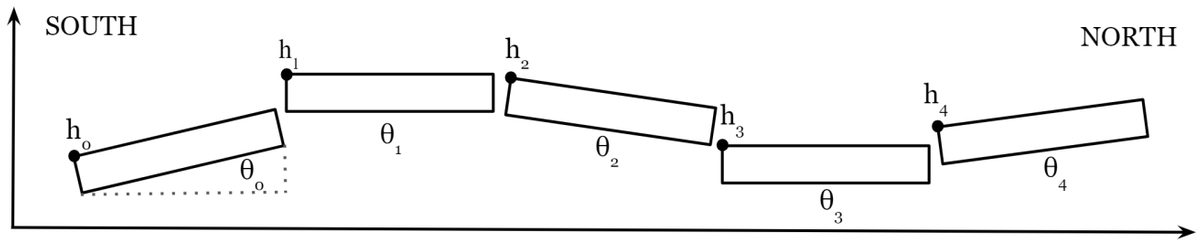

Figure 5: In the second model, slabs are now rectangles of varying slopes in the plane. The heights h_j are measured from the south endpoint of each slab.

Our last model takes the most information into account—cross slopes are considered in this model. Due to this extra degree of freedom, we cannot simply model slabs as objects in the plane, and must instead consider them in the $3D$ world that they naturally reside. Now each slab has four height-related compatibility conditions—one for each endpoint. Also, each slab still has the run slope condition, as well as a cross slope condition that restricts the cross slope to a small range allowed by regulations. The associated DP algorithm requires the most time to execute, but still runs in time polynomial in the values of the inputs while accounting for all relevant slab properties.

4.1 Assumptions

- Each block is about 467.3 linear feet in length. This is the average length of the 156 blocks improved since 2014 as per [20].
- We assume homogeneity of slabs. Each slab is 6 feet long and 4 feet wide. The rationale behind this assumption is that the city of Ithaca requires "plain concrete slabs" to have sides no more than 6 feet long [11]. Also, the ADA requires slabs to be at least 4 feet wide [22].
- Each block has about 78 slabs. This is the number of 6 feet slabs required to cover 467.3 linear feet (the assumed length of each block).
- Each slab has a thickness of over 2 inches, so that we may always cut 2 inches off of any slab. We will fix the thickness at 4 inches, as this is the specified minimum thickness for sidewalks in the city of Ithaca [11]

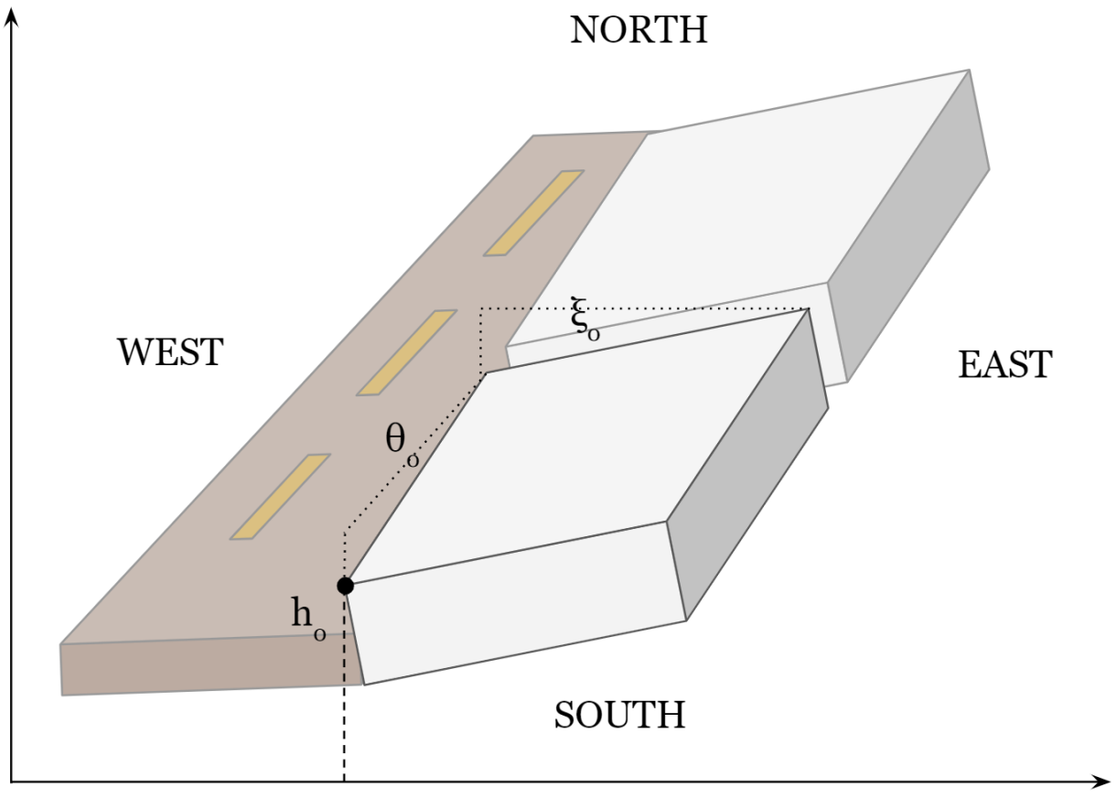

Figure 6: In the third model, all properties of the slabs are considered. Each slab's position is parameterized by the triple (h_j, θ_j, ξ_j), which represents the height of the slab (as measured from the southwest corner), the slope of the slab as measured in the direction parallel to the road, and the slope of the slab as measured in the direction perpendicular to the road. Again, we fix cardinal directions to make reference simpler.

- Raising a slab costs \$5.13 per square foot of the area of the whole slab, cutting costs \$16 per linear foot of the length of the whole slab, replacing costs 22 per square foot of the area of the whole slab being replaced, and leaving a slab alone costs \$0 and does not change its properties.
  - Since the width of each slab is at minimum 4 feet to meet ADA standards [22], raising a slab costs at least \$20.52 per linear foot of the length of the whole slab, so cutting is always cheaper than raising.
- We assume that possible values for heights, running slopes, and cross slopes are taken from finite sets S_H, S_Θ, and S_Ξ respectively. We specify these finite sets when describing our models. This is reasonable since the precision of measurement is limited and the range of possible heights or slopes that a construction crew can attempt to fit a slab to is also finite.
- The running slope of the road is constant along each slab. Also, the slabs have level surfaces, so that each slab only has one running slope and one cross slope.
- About 5% of slabs in a block are broken, and thus must be replaced. If there are n slabs in a block then there are $n/20 + \epsilon$ broken slabs, where ϵ is a Gaussian random variable.
- Raising can only be done when the target height for each endpoint of the slab is at least the initial height of that corresponding endpoint of the slab.

4.1.1 Notation

Below are the definitions of variables used in the dynamic programming problems to follow.
s_j = slab j
h = height of a slab
θ = run slope of a slab
ξ = cross slope of a slab
W = width of a slab
L = length of a slab

θ = run slope of a slab
S_H = set of admissible heights
S_Θ = set of admissible run slopes
S_Ξ = set of admissible cross slopes
h_j^* = initial height of slab
α = maximal allowed difference in height between adjacent slabs
r_j^* = initial run slope of segment of road adjacent to slab j
β = maximal allowed difference in run slope between road and slab
ξ_j^* = initial cross slope of a slab

4.2 First Model: High How are You

For our first model, we assume that the running slope and cross slope of slabs are fixed and constant. This means that we need not consider the width of any slab. We can then model the slabs as rectangles parallel to the x-axis of the plane, with the position on the y-axis being the **height** of each slab (see Figure 4). Note that each point on a single slab has the same height. Thus, raising and cutting only changes the height, and a valid configuration of slabs requires only that the heights of each slab are compatible with the heights of its adjacent slabs.

We assume that we are given n slabs $s_0, \ldots, s_{n-1}$ ordered from south to north, and their heights $h_0^*, \ldots, h_{n-1}^*$. Our goal is to compute a sequence of operations of raising, cutting, replacing, or leaving a block alone for each slab, such that the final configuration of slabs is valid and the sequence of operations is of minimum cost. The corresponding Bellman equation is as follows.

$$\begin{aligned} V(h_0) &= \min_{h \in [\min_k h_k^*,\, \max_k h_k^*]} \text{cost}(h, s_0) + V(h_1|h) \\ V(h_j|h_{j-1}) &= \min_{h \in [h_{j-1}-\alpha,\, h_{j-1}+\alpha]} \text{cost}(h, s_j) + V(h_{j+1}|h) \qquad\qquad j = 1,\ \ldots,\ n-1 \end{aligned}$$

In which $V(h_n|h) = 0$. α is the maximal difference in height between two adjacent slabs in a valid configuration (1/2 inch). $V(h_0)$ is the minimum cost over all valid configurations of the slabs. For $j = 1, \ldots, n-1$, $V(h_j|h_{j-1})$ is the minimum cost for forming a valid configuration of $s_j, \ldots, s_{n-1}$, given that height h_j must be compatible with the height of the previous slab, h_{j-1}. The $\text{cost}(h, s_j)$ is the cost of making slab s_j to have height h_j in the cheapest way possible. It takes on the values (note that the conditions overlap, so it takes on the topmost value for which it meets the condition)

$$\text{cost}(h, s_j) = \begin{cases} 22 \cdot W_j \cdot L_j & \text{if } s_j \text{ is broken} \quad \text{(replace cost)} \\ 0 & \text{if } h = h_j^* \quad \text{(no cost)} \\ 5.13 \cdot L_j & \text{if } h \in [h_j^* - 2,\ h_j^*) \quad \text{(cut cost)} \\ 16 \cdot L_j \cdot W_j & \text{if } h > h_j^* \quad \text{(raise cost)} \\ 22 \cdot W_j \cdot L_j & \text{otherwise} \quad \text{(replace cost)} \end{cases}$$

In which W_j is the width and L_j is the length of s_j. Note that the minimization in computing $V(h_0)$ only considers heights for slab s_0 that are within the minimum and maximum of the original heights of the slabs. This is because any configuration where the first slab is outside of this range can be achieved by a configuration with a first slab within this range. Moreover, the minimization for computing $V(h_j|h_{j-1})$ only considers heights h_j that are compatible with the previous slab's height, meaning those heights within α of h_{j-1}.

We assumed that the set of heights that the slabs can take on is finite. Here, we will specifically assume that the original heights h_k^* of the slabs are measured up to $1/6^{\text{th}}$ of an inch [21]. Thus, the set of heights S_H that we may place slabs at is all heights in a mesh of precision $1/6^{\text{th}}$ inch between $\min_k h_k^*$ and $\max_k h_k^*$. We implement this in a bottom-up algorithm with time complexity $\mathcal{O}(n\alpha|S_H|)$.

4.2.1 Varying Height

To test this model, we compute the costs of optimal configurations of n slabs with randomized initial heights in some range. For a fixed number of slabs n and fixed range of heights $S_H := [0, h_{max}]$, we generate n slabs with heights drawn from a uniform distribution on $S_H = [0, h_{max}]$, and then compute the mean cost over 30 simulations.

However, the heights of slabs in a block are likely not actually uniformly distributed. The height of a slab can generally be expected to be close to the height of its neighboring slabs. Thus, we develop another method to generate slab heights. We can fix the first slab height h_0^*, then model the sequence of remaining slab heights $h_1^*, \ldots, h_{n-1}^*$ as a Markov chain, where we assume a relation

$$h_j^* \sim \mathcal{N}(h_{j-1}, \sigma^2) \quad j = 1, \ldots, n-1$$

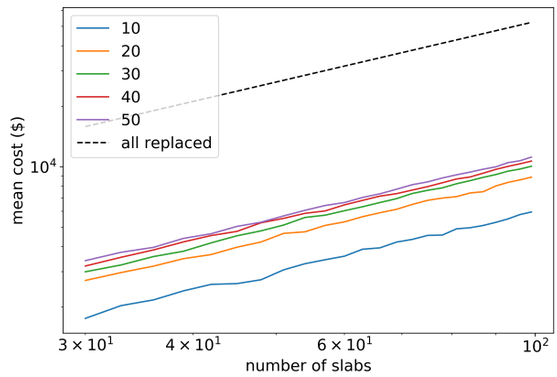

Figure 7: This figure plots the number of slabs versus the mean cost in dollars for repair procedures generated by model 1. x and y are in logscale. The lower 5 runs used an h_{max} of $10, 20, 30, 40, 50$, respectively. The dotted line serves as a baseline, and represents the cost associated with replacing all slabs, as opposed to fixing them.

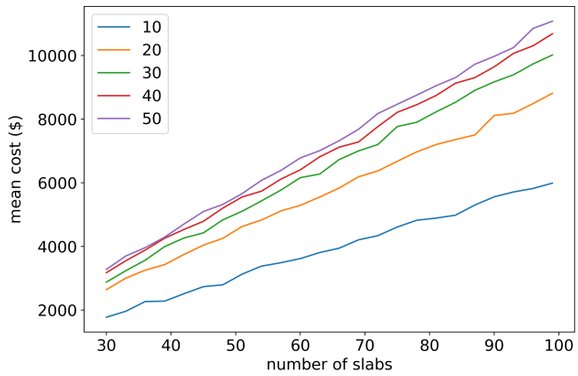

Figure 8: Number of slabs and mean cost in dollars for model 1. Again, legend denotes range of heights which we draw uniformly from. The only difference between this figure and 7 is that the axes are no longer in logscale, and the baseline run has been removed. Compare this figure with 7, which employs Markov chains.

for some variance σ^2. This means that the height of slab s_j is normally distributed about the height of its previous slab s_{j-1}, which is reasonable if adjacent slabs tend to have similar heights. Also, to maintain the assumption that the heights are within a finite range, we round each h_j^* to the nearest height in some height range S_H. We will refer to this as the **neighbor-dependent slab height model**.

4.3 Second Model: Run Rabbit Run

We add to our first model by accounting for running slopes. Say we are given the running slopes $r_0^*, \ldots, r_{n-1}^*$ of the road at the position of each slab (recall that we assume the road has constant running slope along each individual slab) along with the original running slopes $\theta_0^*, \ldots, \theta_{n-1}^*$ of the slabs. Since we do not consider the cross slopes, the slab heights and running slopes can be assumed constant along the width of the slab, so we view the slabs as rectangles ordered from south to north in the plane, which may now be tilted due to the running slopes (see Figure 5).

The points on the surface of a slab may not have constant height now due to the running slopes. Given the height at any point and the running slope of the slab, we may compute the height of any point of the slab. Thus, we consider the heights at a specific point of each slab—for this model we define the **height** of a slab at the height of its south endpoint. Hence, we assume $h_0^*, \ldots, h_{n-1}^*$ are the original heights of the south endpoints of the slabs, and that we are minimizing a function over choices of heights of south endpoints $h_0, \ldots, h_{n-1}$ and run slopes $\theta_0, \ldots, \theta_{n-1}$. Our Bellman equation is of the form

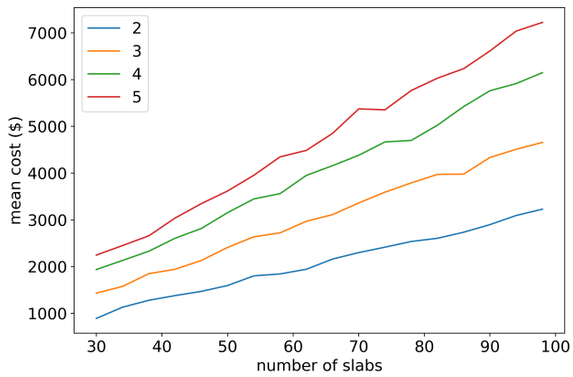

Figure 9: Number of slabs vs. mean costs with neighbor-dependent slab heights in the height range $S_H = [0, 30]$. Different lines are different variances σ^2. Note the lower cost for repairing these slabs compared to that of repairing slabs with heights drawn from the uniform distribution on $[0, 30]$ (the green line in Figure 8).

$$V(h_0, \theta_0) = \min_{\theta \in [r_j^* - \beta,\, r_j^* + \beta]} \; \min_{\min_k h_k^*, \max_k h_k^*} \text{cost}(h, \theta, s_j) + V(h_1, \theta_1 |\; h + \theta L_0)$$

$$V(h_j, \theta_j | h_{j-1}) = \min_{\theta \in [r_j - \beta,\, r_j + \beta]} \; \min_{h \in [h_{j-1} - \alpha,\, h_{j-1} + \alpha]} \text{cost}(h, \theta, s_j) + V(h_{j+1}, \theta_{j+1} |\; h + \theta L_j)$$

$$j = 1, \ldots, n-1$$

Observe that $V(h_j, \theta_j | h_{j-1})$, the optimal choice of $h_j, \ldots, h_{n-1}$ and $\theta_j, \ldots, \theta_{n-1}$, does not depend on the previous slab's running slope θ_{j-1}. This is because compatibility of the j^{th} running slope θ_j only depends on the running slope of the road r_j^*, and compatibility of the j^{th} south endpoint height h_j only depends on the height of the previous slab's north endpoint. Also, when computing the future costs of $s_{j+1}, \ldots, s_{n-1}$ given a height h and run slope θ of s_j, we need to align the south endpoint of s_{j+1} with the north endpoint of s_j. This is captured in the $V(h_{j+1}, \theta_{j+1} | h + \theta L_j)$ term, $h + \theta L_j$ is the height of the north endpoint of s_j if its south endpoint is h.

The cost function $\text{cost}(h, \theta, s_j)$ is modified from the cost in Model 1. Now, to cut, we must also satisfy a condition on θ since the run slope can only be changed so much from a cut of at most 2 inches. Given a south endpoint h, any run slope achieved by a cut lies within the interval

$$\theta_{\text{cut}} \in \Big[\frac{(h_j^* - 2) - h}{L_j},\; \frac{h_j^* - h}{L_j}\Big]$$

To raise, the target south endpoint must be higher than the initial south endpoint. This is analogous to Model 1's raising condition. Also, the north endpoint must now be higher than the initial north endpoint. Thus, the cost is of the form

$$\text{cost}(h, \theta, s_j) = \begin{cases} 22 \cdot W_j \cdot L_j & \text{if } s_j \text{ is broken} & \text{(replace cost)} \\ 0 & \text{if } h = h_j^* \text{ and } \theta = \theta_j^* & \text{(no cost)} \\ 5.13 \cdot L_j & \text{if } h \in [h_j^* - 2,\; h_j^*) \text{ and } \theta \in [\frac{(h_j^* - 2) - h}{L_j},\, \frac{h_j^* - h}{L_j}] & \text{(cut cost)} \\ 16 \cdot L_j \cdot W_j & \text{if } h \geq h_j^* \text{ and } h + \theta L_j \geq h_j^* + \theta_j^* L_j & \text{(raise cost)} \\ 22 \cdot W_j \cdot L_j & \text{otherwise} & \text{(replace cost)} \end{cases}$$

Just as we did with the height, we specify a minimal precision for the run to fix units. We assume that runs are measured in 1/3%. We can once again solve this DP problem with a bottom-up implementation, now with cost $\mathcal{O}(n\alpha\beta|S_H||S_\Theta|)$.

4.3.1 Varying Run

Now, we fix the number of slabs per block to $n = 78$ (our assumed average slabs per block), since we see from section 4.2.1 that for a fixed set of slab parameters the cost grows in n in a predictable manner. Moreover, we will generate slabs with neighbor-dependent slab heights at variance $\sigma^2 = 3$. However, now the height of a slab is defined as the height of its south endpoint, so h_j^* should not depend on h_{j-1}^* but instead the north endpoint of the previous slab. Thus, we redefine the neighbor-dependent slab height model to take the form

$$h_j^* \sim \mathcal{N}(h_{j-1} + \theta_{j-1} L_{j-1},\; \sigma^2)$$

Besides this, we need to specify the run of the road r_j^* along each slab, and the initial run of each slab θ_j^*. We experiment with 3 different types of terrains, which determine different road runs.

- The **flat** terrain has constant run slopes $r_j^* = 0$.
- The **hill** terrain has run slopes that start at zero, increase up to 3 at the midpoint, and then decrease down to zero.
- The **incline** terrain has run slopes that start at zero and then increase up to 7.

For choosing the run θ_j^* of slab s_j, we take from a uniform distribution on a range $[r_j^* - \delta, r_j^* \delta]$ centered at r_j^*, the running slope of the road next to s_j. Once again, drawing these parameters

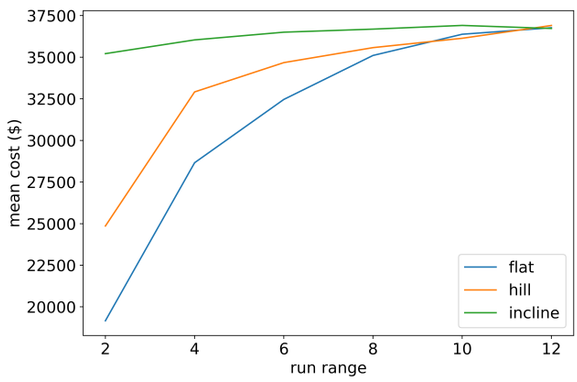

Figure 10: Run slopes drawn uniformly around the road's run slopes.

uniformly from some set is not necessarily the most realistic model. We assume that the slabs s_j are most likely to have run slopes close to the run slopes of the piece of the road they are next to. This can be modelled as

$$\theta_j^* \sim \mathcal{N}(r_j^*, \sigma^2)$$

for some variance σ^2. Note that unlike the neighbor-dependent slab heights model, the run slope of a slab s_j does not depend on the run slope of its adjacent slabs. In practice, we use this model constrained within the range of admissible run slopes $S_\Theta = [-8, 8]$. We will call this the **road-dependent slab runs model** of generating run slopes.

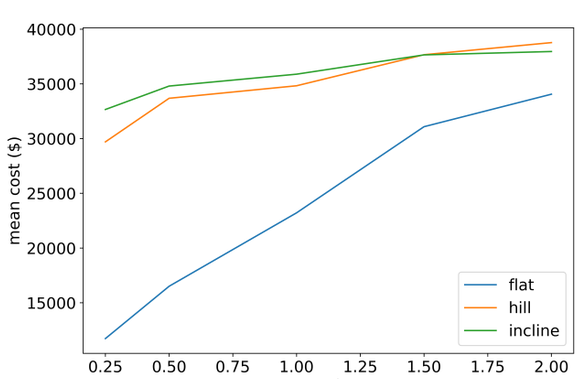

Figure 11: Runs drawn from road-dependent slab runs model. $S_\Theta = [-8, 8]$ so these simulations require lower costs than the analogous uniform run slope ones.

4.4 Third Model: Rock 'n Roll

For our final model, we add to Model 2 by including cross slopes ξ_j. We assume that the slabs have initial cross slopes ξ_j^*. We now have to account for four endpoints of each slab, so model is no longer in the plane and we require more compatibility conditions to be satisfied in a valid configuration (see Figure 6). Our heights $h_0, \ldots, h_{n-1}$ are now the heights of the southwest corner of each slab.

Given the northwest height and cross of the previous slab, h_{j-1} and ξ_{j-1}, for this slab s_j to be compatible we need that the southwest and southeast heights are compatible with the northwest

and northeast heights of the previous slab. If we have chosen a compatible southwest height h of this slab, then southeast endpoint compatibility requires that a compatible cross must satisfy

$$h + \xi W_j \in [h_{j-1} + \xi_{j-1} - \alpha,\ h_{j-1} + \xi_{j-1} + \alpha]$$

Meaning that the southeast heights have to be compatible. This implies that

$$\xi \in \big[\frac{h_{j-1} - h + \xi_{j-1} W_{j-1} - \alpha}{W_j},\ \frac{h_{j-1} - h + \xi_{j-1} W_{j-1} + \alpha}{W_j}\big]$$

So that combining this with our restriction that cross slopes are between 1 and 2%, any compatible cross slope satisfies

$$\xi \in \big[\frac{h_{j-1} - h + \xi_{j-1} W_{j-1} - \alpha}{W_j},\ \frac{h_{j-1} - h + \xi_{j-1} W_{j-1} + \alpha}{W_j}\big] \cap [1, 2]$$

We call this set $I_\Xi(h, h_{j-1}, W_{j-1}, W_j, \alpha)$. Hence, our Bellman equation is of the form

$$V(h_0, \theta_0, \xi_0) = \min_{\theta \in [r_0 - \beta, r_0 + \beta]} \min_{h \in [\min_k h_k^*, \max_k h_k^*]} \min_{\xi \in [1,2]} \text{cost}(h, \theta, \xi, s_0) + V(h_1, \theta_1, \xi_1 | h + \theta L_j, \xi)$$

$$V(h_j, \theta_j, \xi_j | h_{j-1}, \xi_{j-1}) = \min_{\theta \in [r_j - \beta, r_j + \beta]} \min_{h \in [h_{j-1} - \alpha, h_{j-1} + \alpha]} \min_{\xi \in I_\Xi(h, h_{j-1}, W_{j-1}, W_j, \alpha)} \text{cost}(h, \theta, \xi, s_j) + V(h_{j+1}, \theta_{j+1}, \xi_{j+1} | h + \theta L_j, \xi)$$
$$j = 1, \ldots, n-1$$

Compatible cross slopes ξ must be within the range $[1, 2]$ and also must satisfy that the back far side height is within 1/2 inches of the front far side height of the previous slab.

The cost function $\text{cost}(h, \theta, \xi, s_j)$ is a small extension of the cost function for Model 2. We now require $\xi = \xi_j^*$ for there to be no cost of modification. For a cut, the ξ must belong to a certain range. This is given by a range analogous to that of run slopes θ achievable by a cut:

$$\xi_{\text{cut}} = \big[\frac{(h_j^* - 2) - h}{W_j},\ \frac{h_j^* - h}{W_j}\big]$$

Furthermore, for raising we must now consider whether each of the 4 target endpoint heights are at least as high as the matching initial endpoint heights. Again, this is analogous to the further restriction on raising that including run slopes in Model 2 required.

We assume the cross slopes, just like runs, are measured in 1/3%. Our bottom-up implementation for this model runs in $\mathcal{O}(n\alpha\beta|S_H||S_\Theta||S_\Xi|)$. Note that this is still polynomial in the values of the inputs of the problem. Importantly, this scales linearly in the number of slabs, and is thus feasible to use for computing optimal repair plans for all blocks that need to be repaired in Ithaca. The other factors in this runtime complexity are bounded by the precision of measurements and physical constraints of the problem.

4.4.1 Varying Cross

We find that varying cross slopes does not change cost by much. This is reasonable, since the cross slopes do not have compatibility conditions directly restricting them and are bounded within a small range of 1% to 2%. The only compatibility conditions associated with cross slopes are the matching of the east endpoints of adjacent slabs. Thus, we just take cross slopes uniformly within a small range S_Ξ.

Also, while we can still compute optimal repair procedures fairly quickly with this model, it is still slower than the other two models and thus cannot as easily compute large simulations. Assuming that the cross slopes do not change cost much, we can still derive useful information from computing optimal repair procedures of random configurations of slabs. To do this, we fix reasonable choice of parameters as tested with out earlier models.

In particular, we have $n = 78$ slabs, the same three terrains, a height range of $S_H = [0, 30]$, neighbor-dependent slab heights of variance 3, road-dependent slab runs of variance .5, $n/20 + \epsilon$ broken slabs per block (recall ϵ is Gaussian noise), and $S_\Theta = [-8, 8]$. With this, we have that the cost of replacing all slabs is 41184. Our algorithm determines repair procedures with average cost \$17251.73 on the flat terrain, \$32414.44 on the hill terrain, and \$34755.41 on the incline terrain. This is a major cost decrease from the all-replacement scheme in the flat terrain, and still significant albeit not as impressive for the more complex terrains.

4.5 Discussion

Our models for the repair problem benefit from various **strengths**. They build up on each other in an intuitive and interpretable manner, so that each one is more complex and takes into account more information than the previous yet still follows the same optimization principle of dynamic programming. The simpler Model 1 has a fast runtime, and can be used on large simulations to obtain useful data. Model 2 takes the run slopes into account, which allows various important information like road terrain and different endpoints of the slabs to be accounted for. Model 3 takes all slab parameters into account, so it has the most power in modelling real scenarios. Assuming our Bellman equations are valid in modelling the real optimal choices of repair procedures, our algorithms indeed find the optimal choices. The cost functions can be directly changed as needed without changing the underlying equation and algorithmic solutions.

As with any models, there are **weaknesses** in our approaches to the repair problem. We make multiple assumptions, and while we try to justify each, they may still introduce inaccuracies in our models. Although our model allows us to vary length and width of slabs, we have only run simulations on slabs of length and width that are homogenous within each block, even though Ithaca's sidewalk blocks have slabs of varying dimensions. Also, we have assumed that the cost of a cut is \$5.13 per linear feet of slab cut, but in reality a cut should be penalized further because in the future a block that has been cut before may not have enough concrete left to cut again.

These models can be improved in the **future** with several considerations. If all else stays constant and the costs of each repair operation changes, then our models still work if the $\text{cost}(h, \theta, \xi, s_j)$ does not depend on the parameters of other slabs. Also, since repairing is not a one-time procedure, one could incorporate effects of the different repair operations on future repairs of the slab. As mentioned above, a cut could be penalized for not allowing a potential future cut. Likewise, a raise could be weighted to cost less since perhaps a raised block could be cut more or lowered in some different manner in the future.

5 Projecting Future Needs

Rising costs and flat revenues are threatening the effectiveness of the Sidewalk Improvement Program. To further complicate this issue, the city wants to install more sidewalks each year, and climate change hastens the deterioration of cement. We propose a new budget plan for the next 25 years to account for these changes and support the regular maintenance and construction of new sidewalks for Ithaca.

5.1 Assumptions

- We assume that all sidewalks in Ithaca are reinforced. Steel reinforcement is placed 4 inches below the surface of the concrete slab. Recall that we assume thickness of each slab is 4 inches for our repair problem model [11].
- There are currently 700 blocks of sidewalk in Ithaca [23]. The lifespan of a concrete slab is 34.3 years [24], so we initialize the ages of each concrete slab as a randomly distributed chi-squared variable with mean equal to 12.
- We assume that the city of Ithaca immediately replaces a cracked concrete slab.
- For simplicity, we assume that the percent increase in contract prices is constant although the data from past years appears to be unpredictable [18].

5.2 Factors to Consider

To formulate a comprehensive budget plan, we consider the following factors:

1. Chance of breaking: environmental stressors can cause cement to be more fragile. This increases its chance of cracking and its need for repair.
    (a) Carbonation of cement: the number of years before the carbonation front reaches the steel reinforcement is related to the atmospheric CO_2 concentration, which varies with time.
    (b) Freeze-thaw cycles: the number of freeze-thaw days that a concrete slab can withstand is related to the cement strength grade, applied stress ratio, and rising global temperatures, which varies with time.

(c) Random chance: we include a random chance probability that an ordinary concrete slab may crack every year after the age threshold of 30 years, since this nears the average lifespan of a concrete slab [24].

2. Number of new installations: new sidewalk installation plans need to be accounted for in the budget.

3. Rising contract costs: each year, the cost of repair per square foot increases as a multiple of the previous year's cost.

5.2.1 Carbonation

Carbon dioxide can be released into the atmosphere through natural processes, such as cellular respiration and volcanic eruptions, as well as through human activities, such as deforestation and the burning of fossil fuels [5]. In 2018, the global average atmospheric carbon dioxide was 407.4 parts per million [6]. This value is growing each year as we continue to emit greenhouse gases into the atmosphere. According to climate Scenario IS92a, which was developed by the Intergovernmental Panel on Climate Change (IPCC), atmospheric CO_2 concentration changes can be predicted by the following equation [4]:

$$m = 0.0061t - 11.51 \tag{1}$$

where m is the concentration of CO_2 in 10^{-3} kg/m^3 and t is the year. This equation is plotted in Figure 12.

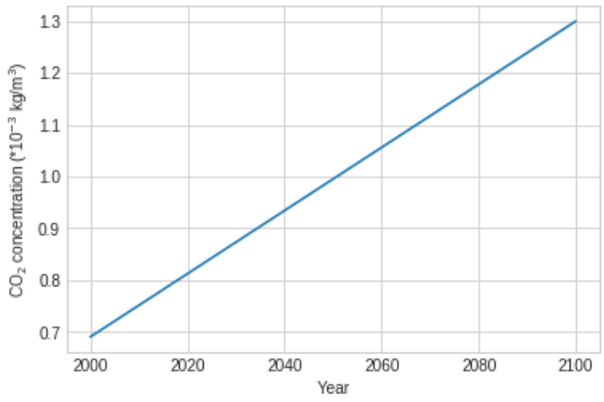

Figure 12: Concentration of CO_2 from the year 2000 to 2100

Increasing carbon emissions impact the rate of carbonation in concrete. When concrete is first poured, it has an alkalinity of pH 13 [3]. This high pH forms a thin oxide layer on the steel reinforcement, known as the passive layer. This protects the steel from reacting with oxygen and water in the air. The passive layer reduces the corrosion rate to 0.1μm/yr as opposed to 0.1 mm/yr.

However, when calcium hydroxide (CO_2) in concrete interacts with carbon dioxide pollutants in the atmosphere, calcium carbonate ($CaCO_2$) is formed, which reduces the alkalinity of concrete [3]. The affected layer of concrete is termed the carbonated layer, while the boundary between it and the alkaline concrete is termed the carbonation front. By Fick's law of diffusion, the carbonation depth is linearly proportional to the square root of the product of carbon dioxide concentration and elapsed time [4]. According to Yoon et al. (2007), the constant of proportionality is approximately equal to 4.4:

$$d = 4.4 \times \sqrt{[CO_2] \times t} \tag{2}$$

where d is the carbonation depth in mm and t is the number of days elapsed (see Figure 13).

Advancement of the carbonation front towards the steel reinforcement at the base of the concrete results in eventual deterioration of the passive layer. Once carbon dioxide reaches the steel reinforcement, an electrochemical reaction creates rust. Rust has a lower density than steel, so the expansion exerts an internal pressure that cracks and damages the concrete [3, 7]. We will assume that, once the carbonation front reaches the steel reinforcement, noticeable damage is not instantaneous. Instead, there is a variable probability r each year before cracks reach the surface and the slab needs to be repaired.

5.2.2 Freeze-Thaw Cycle

In addition, increased carbon emissions have contributed to the global warming trend observed since the mid-twentieth century. These greenhouse gases accumulate in the atmosphere and block

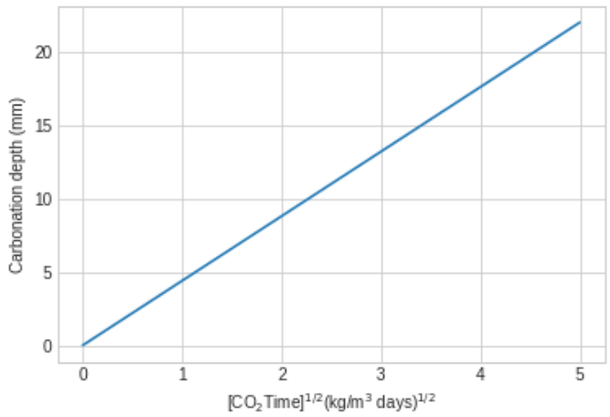

Figure 13: Carbonation depth versus CO_2 concentration and time

heat on Earth from escaping. The gradual heating of Earth will impact the number of freeze-thaw cycles in Ithaca each year.

Ice has a lower density than water, so H_2O expands by about 9% as it freezes [8]. When water in moist concrete is frozen, it produces pressure in the concrete pores. If the pressure exerted exceeds the tensile strength of concrete, the cavity will rupture. Progressive expansion of cement paste from reiterated freeze-thaw cycles will ultimately cause damage and cracking of concrete [8, 9].

The tensile and comprehensive strength of cement affects how many freeze-thaw cycles a slab of concrete can withstand [10]. Other environmental stressors, such as cyclic loading (i.e. people walking), also play a role. According to Sun et al. (1999), if the concrete has a strength grade of C80 and is subject to a stress ratio of 0.5, the concrete can endure at least 500 freeze-thaw cycles.

The city requires that the strength grade of cement in Ithaca to be C150 [11]. Thus, we assume that, after 500 freeze-thaw days have passed, there is a r probability each year that cracks will surface and the city will begin repair.

Moreover, the number of freeze-thaw cycles each year is variable, especially with respect to climate change. Back in the 1980's, Ithaca had approximately 100 freeze-thaw cycles [12]. Rising global temperatures and decreases in temperature variability in the winter has diminished this number to nearly 80 freeze-thaw cycles in 2018 [16, 15, 17].

This past year, Ithaca had an average temperature of $8.1°C$ [13]. To relate the average temperature in Ithaca to the number of freeze-thaw days in a year, we use the equation:

$$d = -1.0104T^2 - 1.848T + 157.2 \tag{3}$$

where d is the number of freeze-thaw days in a year and T is the average temperature of the year [12].

It is projected that the average temperature of the United States will rise by $6°C$ by the end of the 21st century [14]. Because it is also projected that the variance of temperatures in Ithaca winters will diminish by 5 to $10°C$ by the end of the century [15], we do not consider temperature variability to be a factor in increasing the number of freeze-thaw cycles per year. Thus, we use Equation 3 to obtain an upper bound on the number of cycles in a given year.

5.2.3 Rising Contract Costs and New Installation Plans

In 2019, the cost to repair a square foot of concrete is $22. However, according to Tim Logue and John Licitra's "Proposal to Increase Sidewalk Improvement District (SID) Fees" [18], the SID contract prices have been steadily increasing. There is no linear trend in price increase, and the percent change is not monotonically increasing either. Therefore, we predict the percent change by considering that there was an overall 29% change in price per square foot of sidewalk from 2015 to 2018. This corresponds to a 9% increase in price each year.

We do not know the number of blocks of sidewalks that the city plans to install. In past years, the number of blocks have ranged from 1 to 6 [20]. Accounting for varying numbers of new installations, we propose a range of budget plans.

5.3 Stochastic Simulations

There are many parameters that influence the budget:

1. r: the chance that a concrete slab will crack, either due to damage from carbonation, damage from repetitive freeze-thaw cycles, or random chance.

2. i: the percent increase in contract costs each year.

3. ft: the number of freeze-thaw cycles that a concrete slab can withstand before becoming compromising its durability.

4. b: the number of new sidewalk blocks that the city plans to install each year.

As a baseline, we set $r = 2\%$, $i = 2\%$, $ft = 500$, and $b = 6$. We vary a single parameter at a time and ran 10 Monte Carlo simulations to obtain different budget plans for the next 25 years. These simulations demonstrate that our model is relatively robust to perturbations. These trends are displayed in Figure 14.

When increasing the projected number of sidewalk block installations b, the budget also increases. Similarly, increasing the probability of breakage r, the budget understandably increases as well. Interestingly, when $r = 0.05$, the budget is approximately 2.25 million at year 2020, but then it decreases until year 2027 before increasing again. The chi-squared distributed cement ages and high chance of breakage force the city to repair many blocks simultaneously until the sidewalks are replaced with younger (and thus, more durable) cement.

Moreover, when increasing the percent increase in contract costs i, the budget begins to grow exponentially. This is especially noticeable at $i = 9\%$, which was the average price increase in the past three years. This figure serves as a reminder that contractors cannot raise their prices without the citizens of Ithaca paying the six-times the amount of taxes for sidewalk improvement and maintenance.

As we increase the number of freeze-thaw cycles ft that cement can endure before compromising its durability, the budget decreases because the concrete can last longer. In Yoon et al. (2007), cement with strength grade C80 subject to 0.5 stress ratios can withstand 500 freeze-thaw cycles before breakage. Considering that Ithaca requires the strength grade to be C150 [10], we can safely assume that the budget trend will more closely resemble the line for $ft = 1000$.

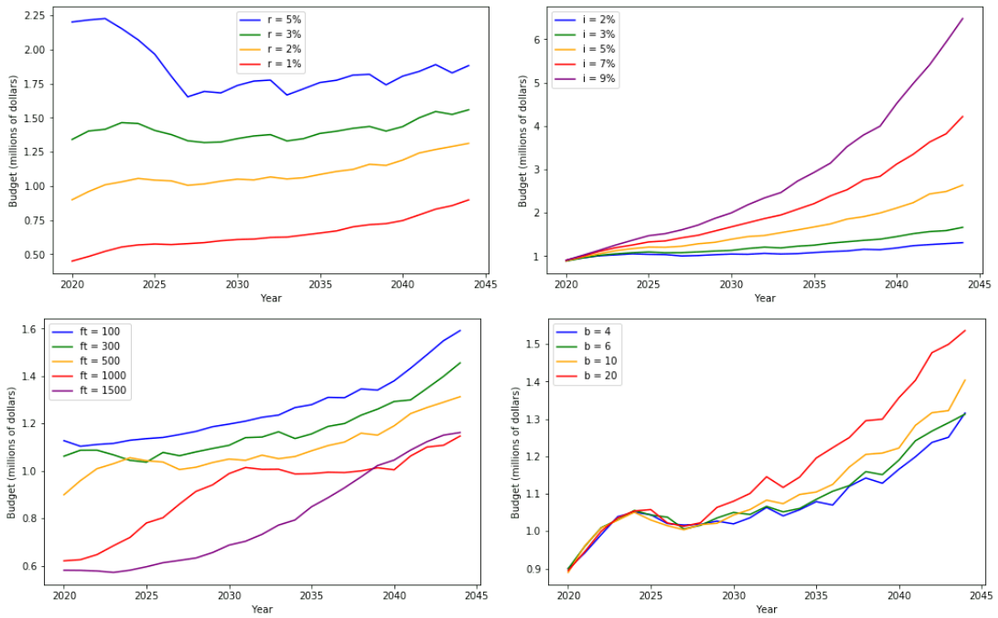

Figure 14: Budget per year with varying r, i, ft, and b values

Using these outputs, we propose the budget outlined in Table 2. For each year of this budget, we assume that Ithaca plans to install 10 new sidewalk blocks; a slab fully exposed to carbonation has a 0.02 probability of cracking; there is a 0.02 probability that a slab will crack after 1000 freeze-thaw cycles; there is a 0.02 random chance of cracking; and contract prices will increase by 3%. This budget predicts that Ithaca will spend \$823,521.60 in 2020, which is approximately equal to the current budget. By 2044, this budget will have to increase to \$1,471,193.27 in order to maintain walkable sidewalk conditions.

Table 2: Budget Plan for 2020-44

Year	Budget($)	Year	Budget($)
2020	823,521.60	2033	1,110,817.70
2021	862,802.16	2034	1,126,492.12
2022	921,007.18	2035	1,126,066.44
2023	960,984.34	2036	1,187,639.38
2024	996,945.09	2037	1,208,694.42
2025	1,038,238.44	2038	1,251,696.88
2026	1,058,541.69	2039	1,323,226.53
2027	1,089,258.94	2040	1,329,164.94
2028	1,085,952.33	2041	1,385,345.00
2029	1,123,904.48	2042	1,430,850.99
2030	1,116,181.69	2043	1,444,911.63
2031	1,107,203.27	2044	1,471,193.27
2032	1,126,116.14		

5.4 Discussion

This model of future budget requirements has **strengths** arising from its flexibility in accounting for a variety of parameters. Carbonation and freeze-thaw cycles, the main external contributors to sidewalk decay in our model, have been widely studied and have significant scientific literature describing them. In fact, our model leverages specific information about the freeze-thaw cycles in Ithaca itself.

Despite its strengths, this model has significant **weaknesses** as well. Although there is much research on climate change, there are still large degrees of uncertainty in properties of the future climate of the Earth. Since our model directly depends on properties like concentration of CO_2 and variance of temperature within each year, the accuracy of the model is uncertain. Also, our assumption that costs increase by a fixed percentage each year may be far from the truth, as can be seen in the varying cost increases of the last few years.

For the **future**, our model can be updated with further information to compute more accurate estimates. As science progresses and humankind can better predict the future of Earth's climate, our model would become more accurate as well. There are also various other parameters and factors that can be included in our model. The primary utility of the sidewalk system is to serve citizens, so it would be improved by adding projects of walking traffic, which depends on parameters like future population, population density distribution, and activity level of Ithaca's citizens.

6 Conclusion

We have developed a suite of models to address the problems of determining optimal contracts, optimal repair procedures, and future budget requirements for the city of Ithaca's sidewalk system. In designing optimal contracts, we used mathematical programming to cluster work sites into manageable groups. We found that the third model gives the user a great deal of flexibility, that the second model is good for even splitting, and that the first model is effective in guaranteeing a uniformly small cluster radius.

The optimal repair problem was able to be formulated in three different dynamic programming problems of varying complexities. Repair procedures computed for varying configurations of slabs are found to cost significantly less than the base policy of replacing every slab in a block.

Incorporating the effects of climate change, rising costs, and new installation plans into our model, we proposed a 25-year budget plan for the city of Ithaca. We predict that revenue must increase from $823,521.60 in 2020 to $1,471,193.27 in 2045. Ithaca's sidewalks provide a fundamental service to the community; we hope that our models and considerations may help to solve the core problems that sidewalk improvement faces.

7 Bibliography

References

[1] Rao, M.R. (1971). Cluster Analysis and Mathematical Programming. *Journal of the American Statistical Association, 66*(335), 622-626. doi: 10.2307/2283542.

[2] Pardalos, P.M., Rendl, F., & Wolkowicz, H. The Quadratic Assignment Problem: A Survey and Recent Developments. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 00*(0000). doi: 10.1090/dimacs/016/01.

[3] Ware, T. (2013). Diagnosing and repairing carbonation in concrete structures. *Journal of Building Survey, Appraisal & Valuation, 4*(1), 338–344.

[4] Yoon, I.S., Copuroglu, O., Park, & K.B. (2007). Effect of global climatic change on carbonation progress of concrete. *Atmospheric Environment, 41*(34), 7274-7285. doi: 10.1016/j.atmosenv.2007.05.028.

[5] Galan, I., Andrade, C., Mora, P., & Sanjuan M.A. (2010). Sequestration of CO_2 by Concrete Carbonation. *Environmental Science & Technology, 44*(4), 3181–3186. doi: 10.1021/es903581d.

[6] Lindsey, R. (2019). Climate Change: Atmospheric Carbon Dioxide. Retrieved from `https://www.climate.gov/news-features/understanding-climate/climate-change-atmospheric-carbon-dioxide`.

[7] Pepin, R. (2019). Understanding the Process of Corrosion in Reinforced Concrete. Retrieved from `https://www.giatecscientific.com/education/understanding-concrete-corrosion/`.

[8] Portland Cement Association. (2019). Freeze-Thaw Resistance. Retrieved from `https://www.cement.org/Learn/concrete-technology/durability/freeze-thaw-resistance`.

[9] Molero, M., Aparicio, S., Al-Assadi, G., Casati, M.J., Hernandez, M.G., & Anaya, J.J. (2012). Evaluation of freeze–thaw damage in concrete by ultrasonic imaging. *NDT & E International, 52*, 86-94. doi: 10.1016/j.ndteint.2012.05.004.

[10] Eilers Construction & Consulting. (2018). What are the Different Grades of Cement? Retrieved from `http://ecc-crete.com/what-are-the-different-grades-of-cement/`.

[11] City of Ithaca, Tompkins County. (2019). Article VIII: Concrete Sidewalk Specifications. Retrieved from `https://ecode360.com/IT1348`.

[12] Schmidlin, T.W., Dethier, B.E., & Eggleston, K.L. (1986). Freeze-Thaw Days in the Northeastern United States. *Journal of Climate and Applied Meterology, 26*. doi: 10.1175/1520-0450(1987)026<0142:FTDITN>2.0.CO;2.

[13] U.S. Climate Data. (2019). Climate Ithaca - New York. Retrieved from `https://www.usclimatedata.com/climate/ithaca/new-york/united-states/usny0717/2019/1`.

[14] Public Health (2019). Climate Change. Retrieved from `https://www.publichealth.org/public-awareness/climate-change/`.

[15] Bathiany, S., Dakos, V., Scheffer, M., & Lenton, T.M. (2018). Climate models predict increasing temperature variability in poor countries. *Science Advances, 4*(5). doi: 10.1126/sciadv.aar5809.

[16] Peng, X., Frauenfeld, O.W., Cao, B., Wang, K., Wang, H., Su, H., Huang, Z., Yue, D., & Zhang, T. (2016). Response of changes in seasonal soil freeze/thaw state to climate change from 1950 to 2010 across china. *Journal of Geophysical Research: Earth Surface, 121*(11). doi: 10.1002/2016JF003876.

[17] Ho, E., & Gough, W.A. (2005). Freeze thaw cycles in Toronto, Canada in a changing climate. *Theoretical and Applied Climatology, 83*(1-4), 203-2010. doi: `https://doi.org/10.1007/s00704-005-0167-7`.

[18] Logue, T., & Licitra, J. (2019). A Proposal to Increase Sidewalk Improvement District (SID) Fees. Retrieved from `https://e.math.cornell.edu/sites/mcm/old_problems/SID_memo_2019_update.pdf`.

[19] 2020 Proposed Sidewalk Work Plan and Budget. Retrieved from https://www.cityofithaca.org/DocumentCenter/View/10417/Draft-2020-SID-Work-Plan.

[20] Sidewalk Improvement Program. Retrieved from https://storymaps.arcgis.com/stories/4df2b76ade6a4171a5ccbed80bf47ceb.

[21] Schwolsky, R. (2007). Cutting Concrete: How a Pro Breaks Through. Retrieved from https://www.toolsofthetrade.net/hand-tools/cutting-tools/cutting-concrete-how-a-pro-breaks-through_o.

[22] Engineering Policy Guide. 642.1: Sidewalk Design. Retrieved from http://epg.modot.org/index.php/642.1_Sidewalk_Design_Criteria.

[23] City of Ithaca with Street Block Number - 2018 [PDF file]. Retrieved from https://cityofithaca.org/DocumentCenter/View/1434/City-of-Ithaca-with-Street-Block-Number-PDF?bidId=.

[24] Maintenance Issues. Retrieved from http://walkbikecny.org/wp-content/uploads/2014/06/SSM_ch7_Maintenance_Issues.pdf.

8 Appendix

Appendix A1: Clustering with Model 1 (minimax radius)

Model 1: Minimax Linear Program, 3 clusters

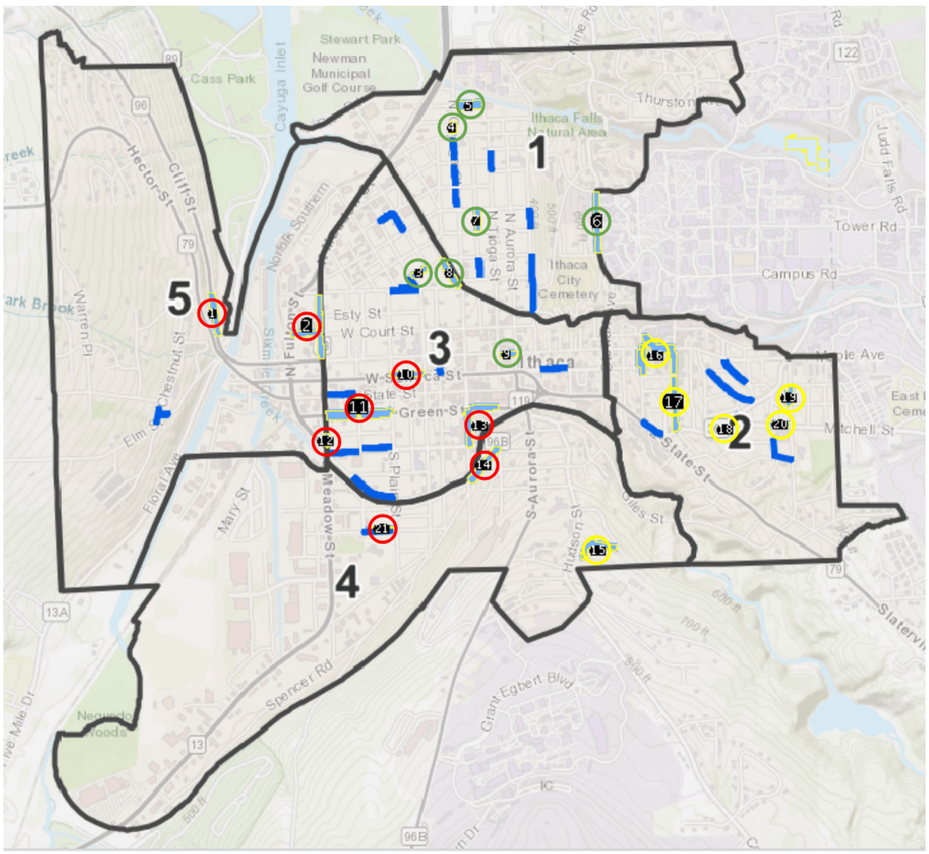

Figure 15

Model 1: Minimax Linear Program, 4 clusters

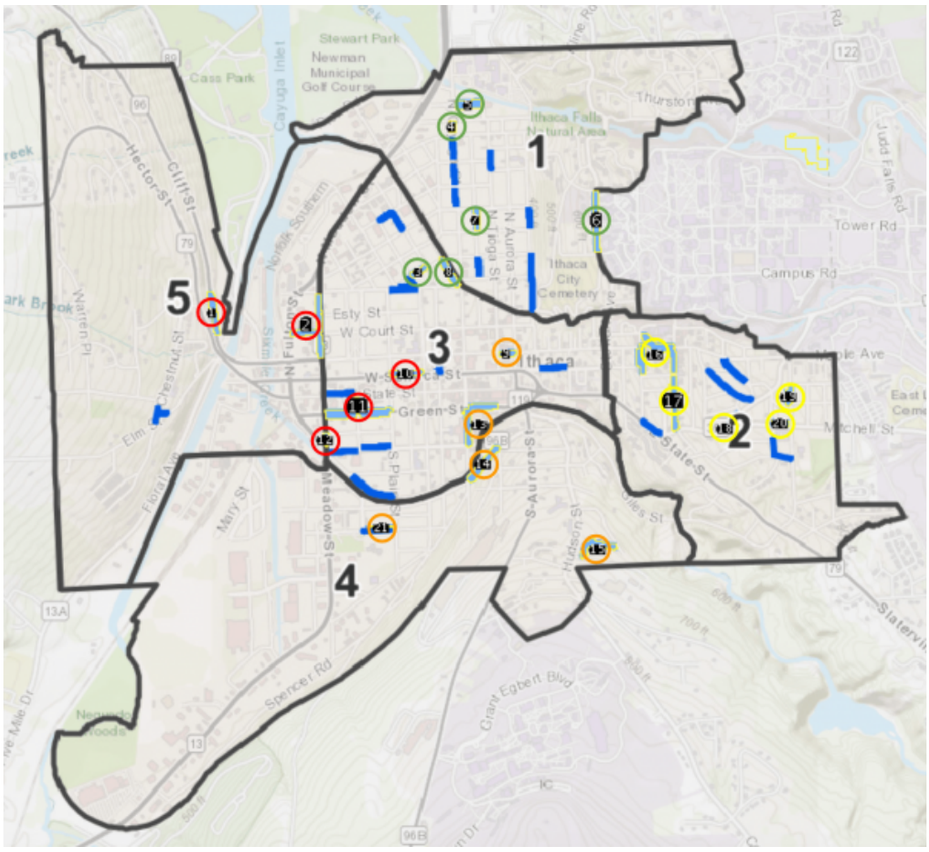

Figure 16

Model 1: Minimax Linear Program, 5 clusters

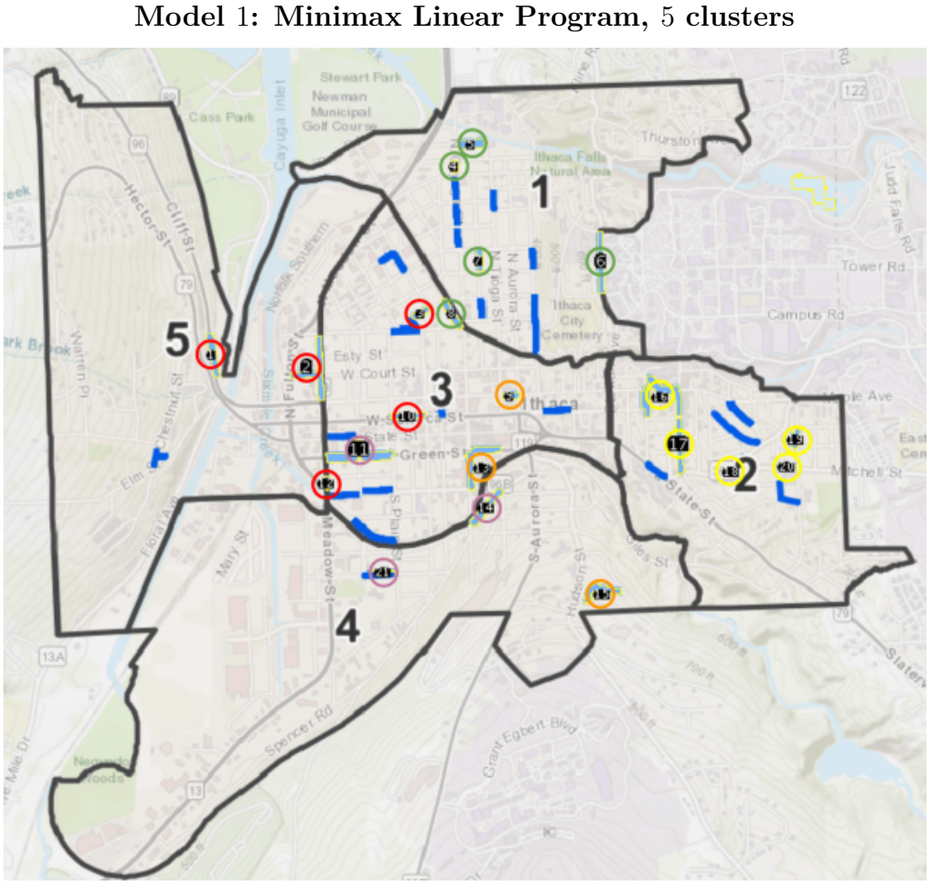

Figure 17

Model 1: Minimax Linear Program, 6 clusters

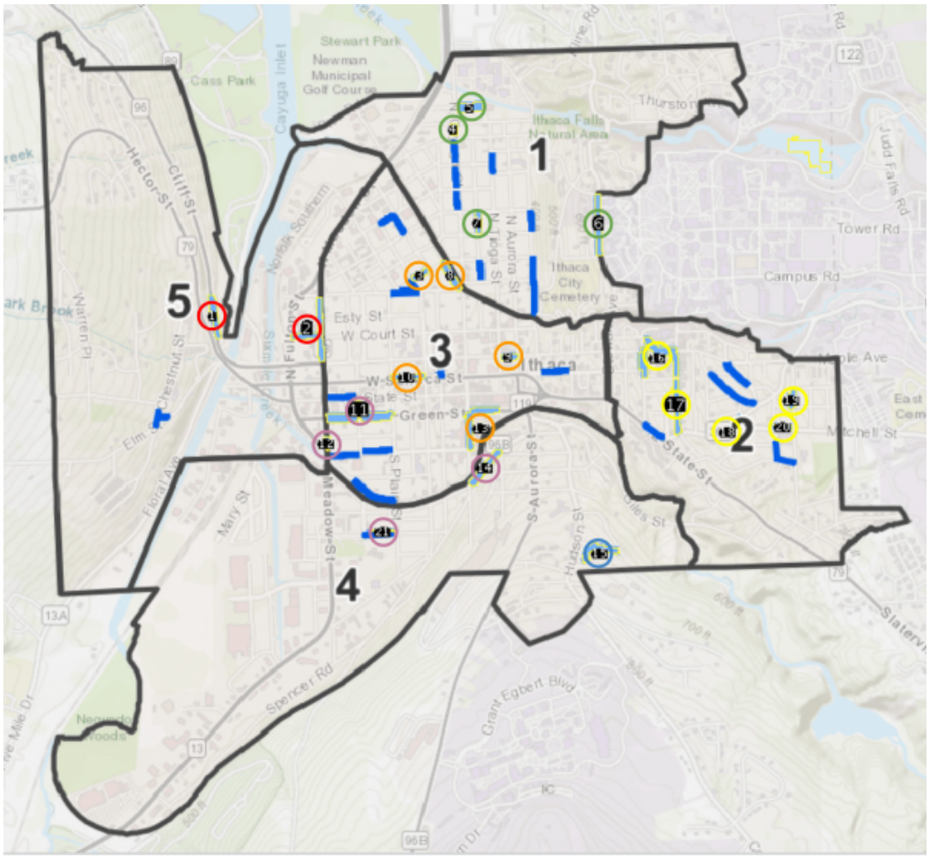

Figure 18

Appendix A2: Clustering with Model 2 (quadratic semi-assignment problem)

Model 2: Quadratic Semi-Assignment Problem, 2 clusters

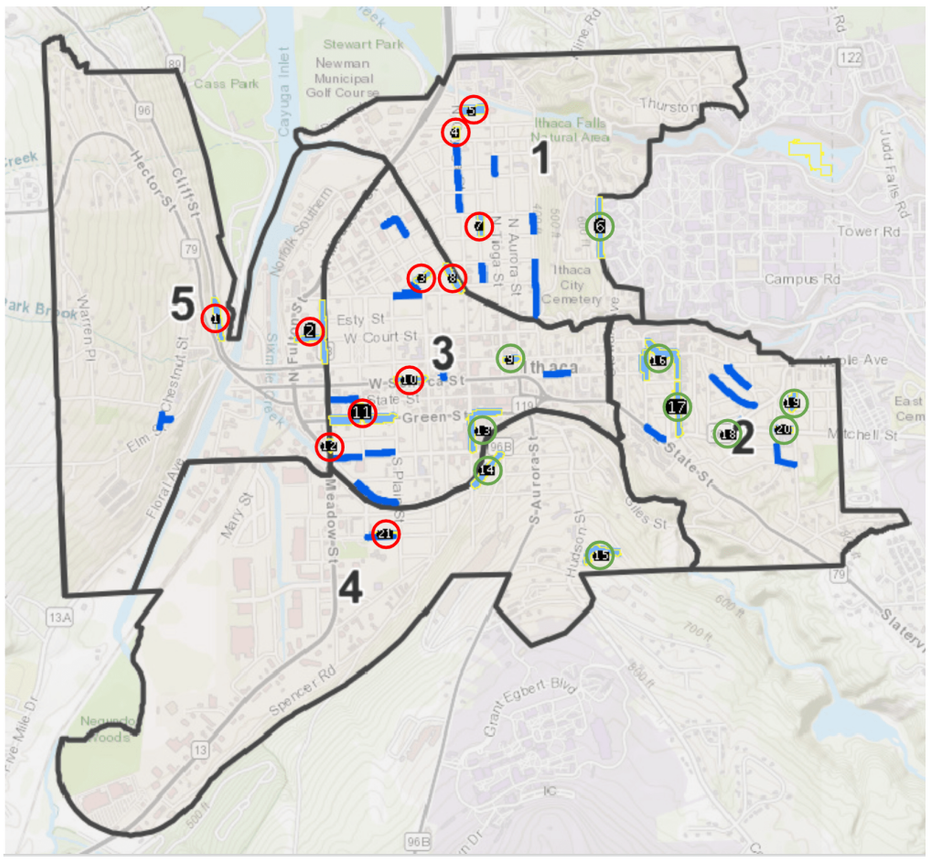

Figure 19: Within the red cluster, the maximal intra-cluster distance is $d(5, 21) = 1.89$ km. As such, it is unlikely that a construction crew would be able to avoid high transition costs while working within this cluster. The minimum sum of intra-cluster pairwise distances is 104.67 km

Model 2: Quadratic Semi-Assignment Problem, 3 clusters

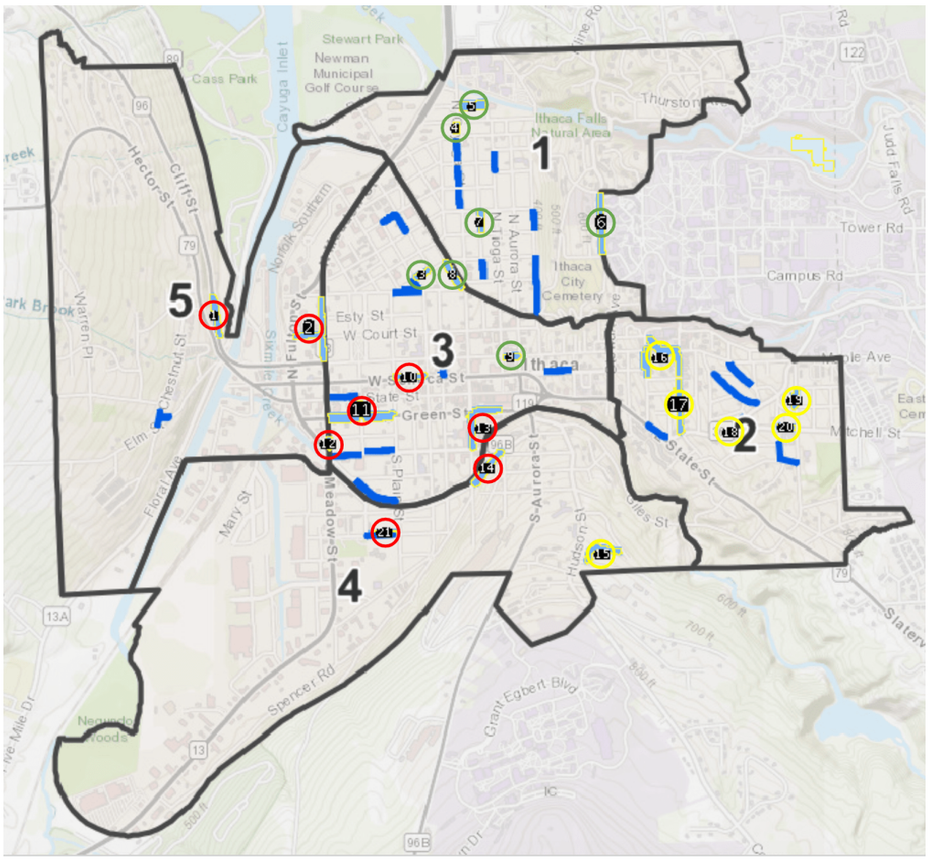

Figure 20: When there are 3 clusters, the cluster sizes are more reasonable. The minimum sum of intra-cluster pairwise distances is 49.21 km. However, the largest intra-cluster distance is $d(1, 21) = 1.54$ km, which is still somewhat large.

Quadratic Semi-Assignment Problem, 4 clusters

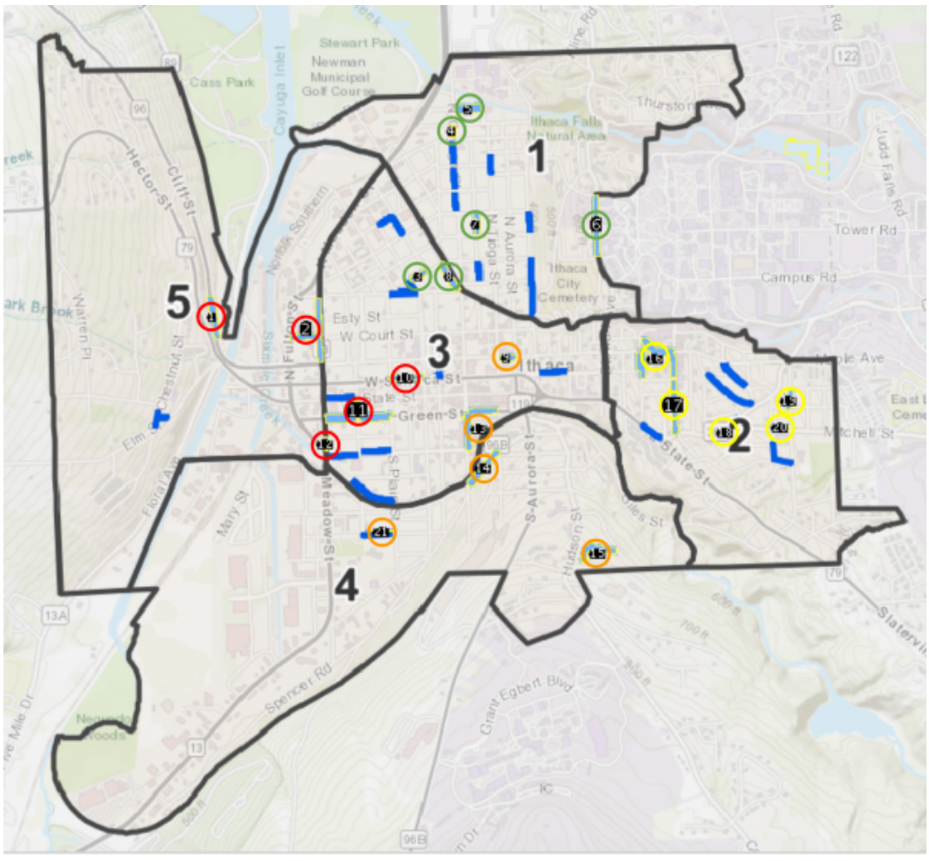

Figure 21

Model 2: Quadratic Semi-Assignment Problem, 5 clusters

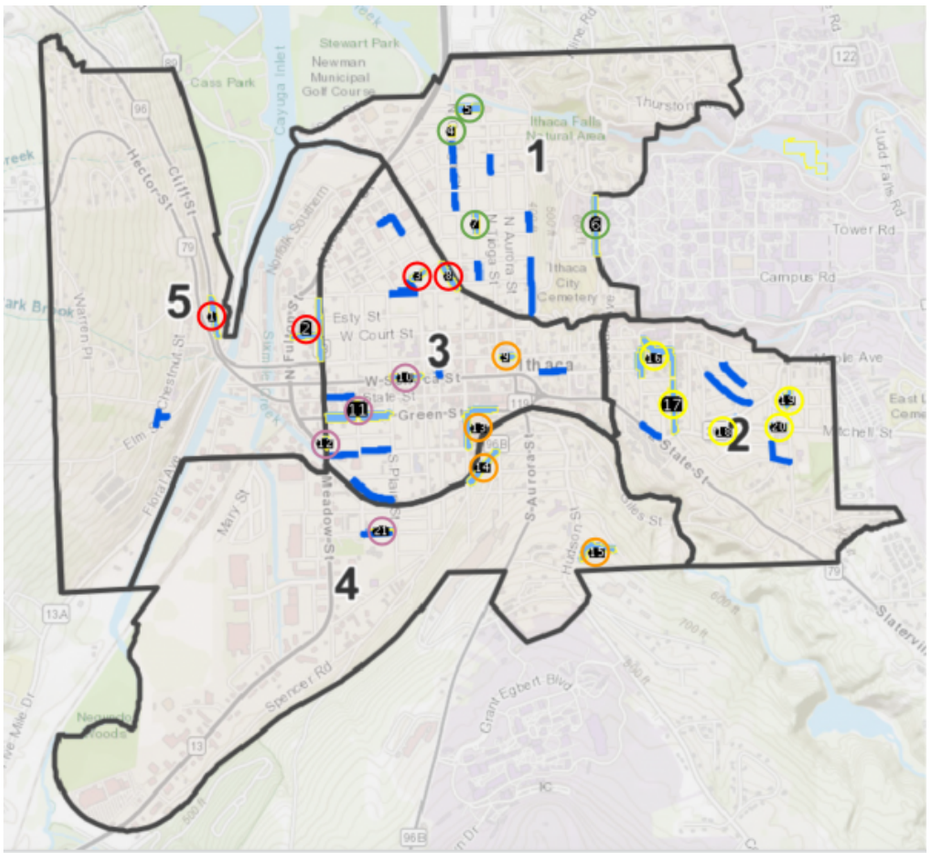

Figure 22

Model 2: Quadratic Assignment Problem, 6 clusters

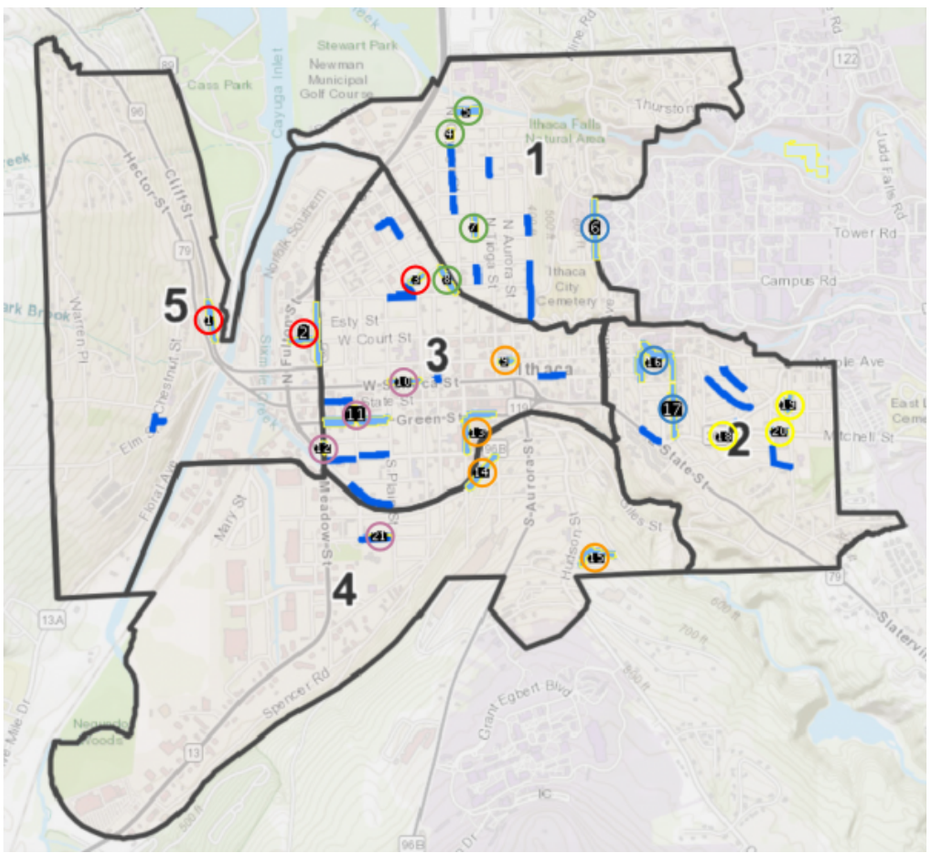

Figure 23

Appendix A3: Clustering with Model 3

Model 3: Averaged Quadratic Semi-Assignment, $(15, 6)$-Partition

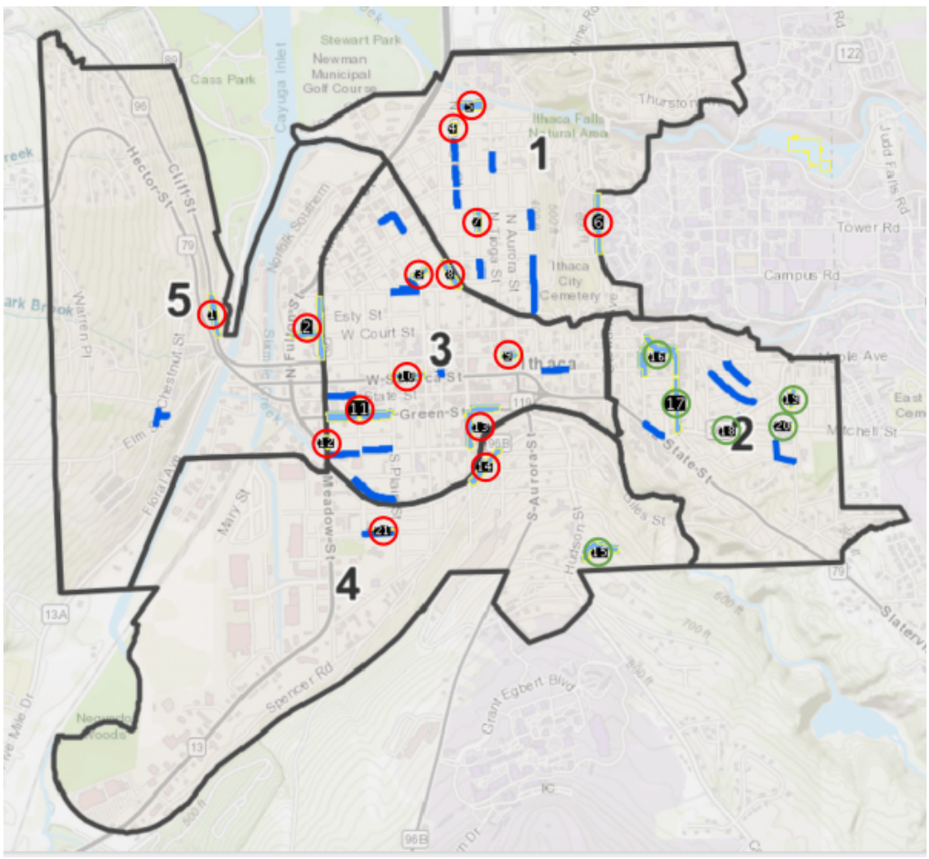

Figure 24

Model 3: Averaged Quadratic Semi-Assignment, $(7, 7, 7)$-Partition

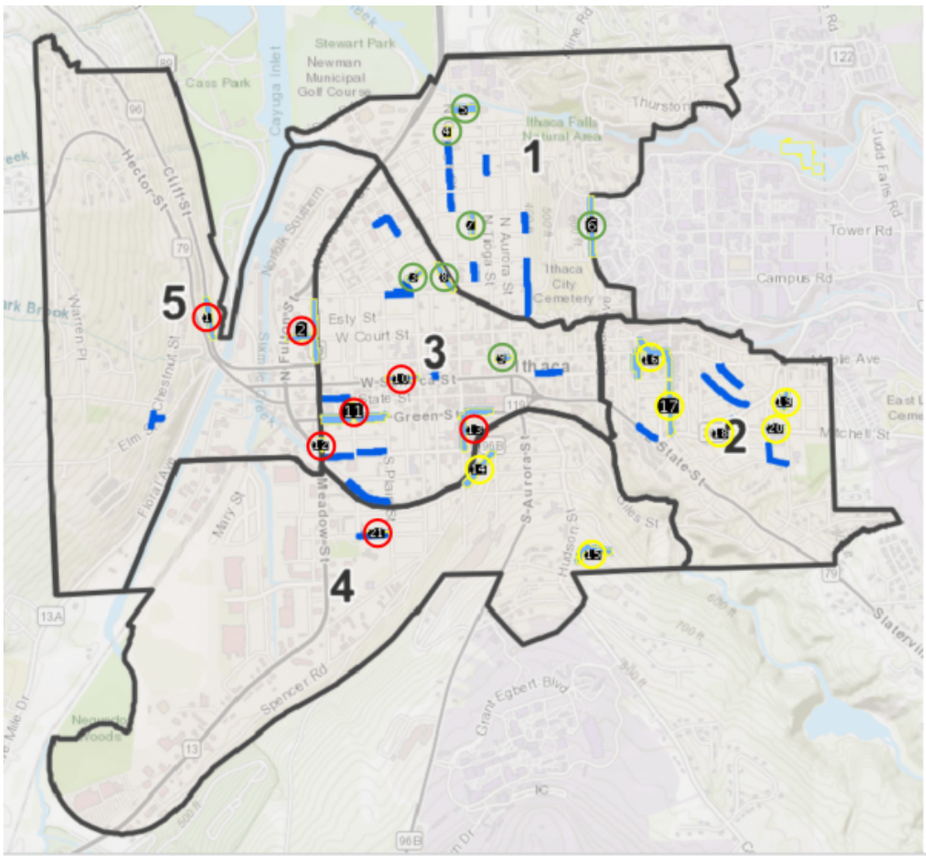

Figure 25

Model 3: Averaged Quadratic Semi-Assignment, $(3, 6, 12)$-Partition

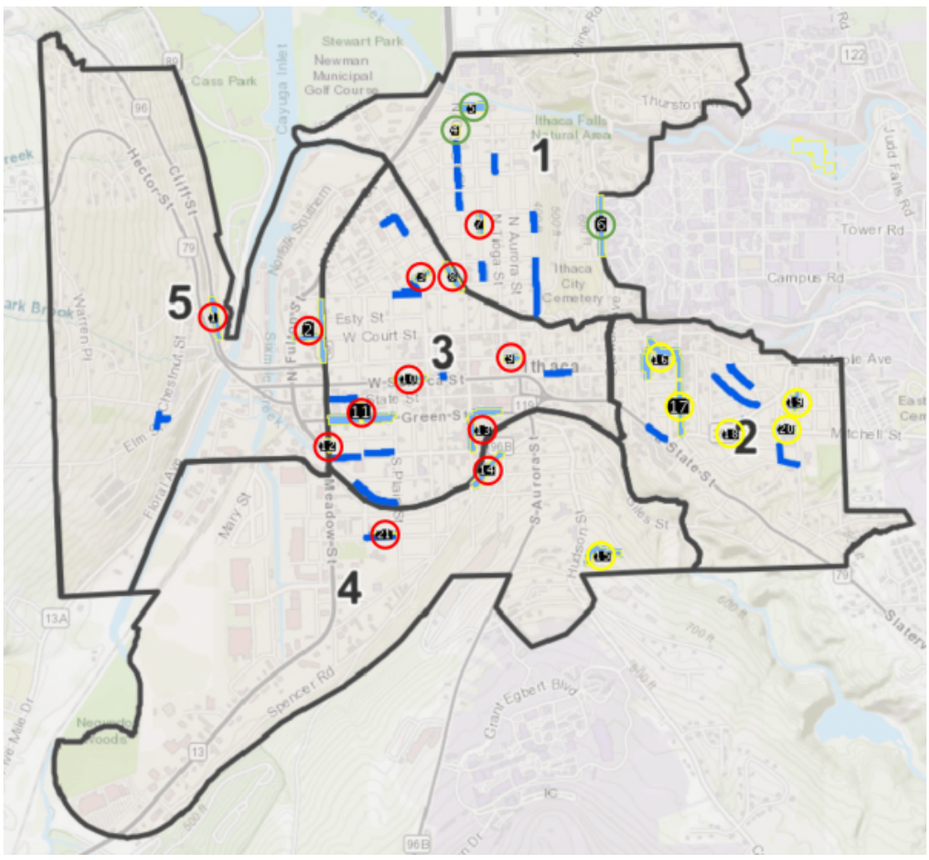

Figure 26

Model 3: Averaged Quadratic Semi-Assignment, $(3,3,3,4,4)$-Partition

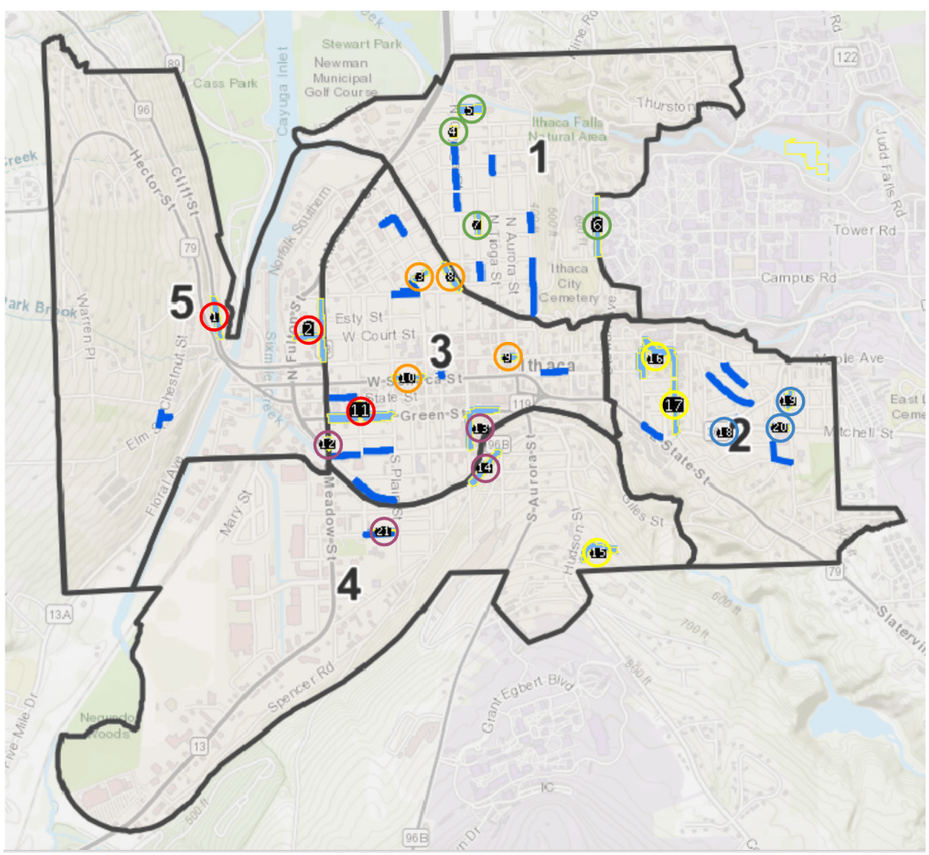

Figure 27

8.1 Codes

```
############################################################
# Part (B)
''' Minimax LP
'''
from gurobipy import *
import numpy as np

def dist(loc, i, j):
    return np.linalg.norm(np.array(loc[i])-np.array(loc[j]))

# Create a new model
m = Model("Minimizing the Maximum Within-Block Distance")
#locations of worksites
loc = [[277, 302], [340, 304], [432, 281], [463, 171],
[467, 154], [573, 225], [481, 237], [455, 276], [501, 347],
[417, 362], [367, 383], [366, 411], [472, 388], [476, 420],
[563, 492], [606, 334], [631, 367], [673, 405], [721, 374], [710, 401], [402, 482]]
s = len(loc)
# prespecify the number of clusters desired
k = 5
# Create binary variables
vv = m.addVars(s, k, vtype=GRB.BINARY)
dd = m.addVar(vtype = GRB.INTEGER, name = "minimax dist")

# Set Objective
obj = LinExpr(dd)
m.setObjective(obj, GRB.MINIMIZE)

# Add constraints
m.addConstr(dd, GRB.GREATER_EQUAL, 0)
for w in range(k):
    for i in range(s):
        for j in range(i+1, s):
            expr = dist(loc, i, j)*(vv[i, w] + vv[j, w]) - dd
            m.addConstr(expr, GRB.LESS_EQUAL, dist(loc, i, j))
for i in range(s):
    expr = vv.sum(i, '*')
    m.addConstr(expr, GRB.EQUAL, 1)

# Optimize model
m.optimize()

for v in m.getVars():
    print('%s %g' % (v.varName, v.x))

print('Obj: %g' % m.objVal)

''' QSAP
'''
from gurobipy import *
import numpy as np

def dist(loc, i, j):
    return np.linalg.norm(np.array(loc[i])-np.array(loc[j]))

# Create a new model
m = Model("Quadratic Semi-Assignment Problem")
#locations of worksites
loc = [[277, 302], [340, 304], [432, 281], [463, 171],
[467, 154], [573, 225], [481, 237], [455, 276], [501, 347],
```

```
[417, 362], [367, 383], [366, 411], [472, 388], [476, 420],
[563, 492], [606, 334], [631, 367], [673, 405], [721, 374], [710, 401], [402, 482]]
s = len(loc)
# prespecify the number of clusters desired
k = 3
# Create binary variables

#add vars
vv = m.addVars(s, k, vtype=GRB.BINARY)

# Set Objective
obj = QuadExpr()
for w in range(k):
    for i in range(s):
        for j in range(i+1, s):
            obj.add(vv[i, w]*vv[j, w]*dist(loc, i, j))

#obj.add()
m.setObjective(obj, GRB.MINIMIZE)

# Add constraints
for i in range(s):
    expr = vv.sum(i, '*')
    m.addConstr(expr, GRB.EQUAL, 1)

# Optimize model
m.optimize()

for v in m.getVars():
    print('%s %g' % (v.varName, v.x))

print('Obj: %g' % m.objVal)

''' Sum of averages QIP
'''
from gurobipy import *
import numpy as np

def dist(loc, i, j):
    return np.linalg.norm(np.array(loc[i])-np.array(loc[j]))

# Create a new model
m = Model("Minimizing the Sum of Average Within Group Distance")
#locations of worksites
loc = [[277, 302], [340, 304], [432, 281], [463, 171],
[467, 154], [573, 225], [481, 237], [455, 276], [501, 347],
[417, 362], [367, 383], [366, 411], [472, 388], [476, 420], [563, 492],
[606, 334], [631, 367], [673, 405], [721, 374], [710, 401], [402, 482]]
s = len(loc)
# prespecify the number of clusters desired
k = 6
#prespecify cluster sizes:
sizes = [3, 3, 3, 4, 4, 4]

# Create binary variables
vv = m.addVars(s, k, vtype=GRB.BINARY)

# Set Objective
obj = LinExpr()
for w in range(k):
    tmp = LinExpr()
    for i in range(s):
```

```
        for j in range(i+1, s):
            tmp += vv[i, w]*vv[j, w]*dist(loc, i, j)**2
    tmp = tmp*(1/sizes[w])
    obj += tmp

m.setObjective(obj, GRB.MINIMIZE)

# Add constraints
for w in range(k):
    expr = vv.sum('*', w)
    m.addConstr(expr, GRB.EQUAL, sizes[w])
for i in range(s):
    expr = vv.sum(i, '*')
    m.addConstr(expr, GRB.EQUAL, 1)

# Optimize model
m.optimize()

for v in m.getVars():
    print('%s %g' % (v.varName, v.x))

print('Obj: %g' % m.objVal)


################################################################
# Part (C)
#Constants

RAISE_RATE = 5.13
CUT_RATE = 16
REPLACE_RATE = 22
DEFAULT_L = 6
DEFAULT_W = 4
BROKEN_RATE = 0.05
CONV_RUN = 1/3 # convert run units
CONV_CROSS = 1/3 # convert cross units

''' Slab class and dynamic programming algorithms for the three slab repair models.
'''
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import random
from constants import *

class Slab():
    def __init__(self, height=0, run=0, cross=0, W=DEFAULT_W, L=DEFAULT_L, broken=False,
            age=0, ft_cycles=0, carb_depth=0):
        self.height = height
        self.run = run
        self.cross = cross
        self.W = W
        self.L = L
        self.broken = broken
        self.age = age
        self.ft_cycles = ft_cycles
        self.carb_depth = carb_depth

def repairs_m1(slabs, htol=3,
        raise_rate=RAISE_RATE, replace_rate=REPLACE_RATE, cut_rate=CUT_RATE):
    ''' Model 1
        Assumes heights are integers
```

```
    '''
    def cost(h, slab):
        ''' Cost to raise or repair from current height to target height
        '''
        if slab.broken:
            return replace_rate*slab.W*slab.L
        elif h == slab.height:
            return 0
        elif slab.height-2 <= h < slab.height:
            return cut_rate*slab.L
        elif h > slab.height:
            return raise_rate*slab.W*slab.L
        else:
            return replace_rate*slab.W*slab.L

    min_h, max_h = min([s.height for s in slabs]), max([s.height for s in slabs])
    height_ran = list(range(min_h, max_h+1))
    M = [[np.inf for i in height_ran] for k in range(len(slabs))]

    for i, h in enumerate(height_ran):
        M[0][i] = cost(h, slabs[0])

    for k in range(1, len(slabs)):
        for i, h in enumerate(height_ran): # checking current height
            mini = np.inf
            for i_prev in range(max(i-htol, 0), min(i+htol, len(height_ran)-1)+1):
                cost_k = cost(h, slabs[k]) + M[k-1][i_prev]
                mini = min(mini, cost_k)
            M[k][i] = mini

    min_cost = np.min(M[-1])
    return min_cost, M


def repairs_m2(slabs, road_runs, htol=3, rtol=6,
        raise_rate=RAISE_RATE, replace_rate=REPLACE_RATE, cut_rate=CUT_RATE):
    ''' Model 2
    '''
    def cost(h, run, slab):
        if slab.broken:
            return replace_rate*slab.W*slab.L
        elif h == slab.height and run == slab.run:
            return 0
        elif (slab.height-htol <= h <= slab.height) and
                ((slab.height-h-2)/slab.L <= run <= h/slab.L):
            return cut_rate*slab.L
        elif (h >= slab.height) and (h+run*slab.L >= slab.height+slab.run*slab.L):
            return raise_rate*slab.L*slab.W
        else:
            return replace_rate*slab.L*slab.W

    n = len(slabs)
    min_h, max_h = min([s.height for s in slabs]), max([s.height for s in slabs])
    height_ran = list(range(min_h-htol, max_h+htol+1))
    min_run, max_run = min([s.run for s in slabs]), max([s.run for s in slabs])
    min_run, max_run = min(min_run, min(road_runs)), max(max_run, max(road_runs))
    run_ran = list(range(min_run-rtol, max_run+rtol+1))
    M = [[[np.inf for j in run_ran] for i in height_ran] for k in range(n)]
    for i, h in enumerate(height_ran):
        for j, r in enumerate(run_ran):
            M[0][i][j] = cost(h, r, slabs[0])
    for k in range(1, n):
```

```
        for i, h in enumerate(height_ran): # cost of height h
            for j, r in enumerate(run_ran): # cost of run r
                mini = np.inf
                min_valid_h = h - slabs[k-1].L*slabs[k-1].run*CONV_RUN - htol
                max_valid_h = h - slabs[k-1].L*slabs[k-1].run*CONV_RUN + htol
                valid_prev_h = intv_to_range(height_ran, min_valid_h, max_valid_h)
                for i_prev in valid_prev_h:
                    for j_prev in range(max(road_runs[k]-rtol,0), min(road_runs[k]+rtol,
                        len(run_ran)-1)+1):
                        cost_t = cost(h, r, slabs[k]) + M[k-1][i_prev][j_prev]
                        mini = min(mini, cost_t)
                M[k][i][j] = mini

    min_cost = np.min(M[-1])
    return min_cost, M

def repairs_m3(slabs, road_runs, htol=3, rtol=6, ctol=3,
    raise_rate=RAISE_RATE, replace_rate=REPLACE_RATE, cut_rate=CUT_RATE):
    ''' Model 3
    '''
    def cost(h, run, cross, slab):
        if slab.broken:
            return replace_rate*slab.W*slab.L
        elif h == slab.height and run == slab.run:
            return 0
        elif (slab.height-htol <= h <= slab.height) and
                ((slab.height-h-2)/slab.L <= run <= h/slab.L):
            return cut_rate*slab.L
        elif (h >= slab.height) and (h+run*slab.L >= slab.height+slab.run*slab.L):
            return raise_rate*slab.L*slab.W
        else:
            return replace_rate*slab.L*slab.W

    n = len(slabs)
    min_h, max_h = min([s.height for s in slabs]), max([s.height for s in slabs])
    height_ran = list(range(min_h-htol, max_h+htol+1))
    min_run, max_run = min([s.run for s in slabs]), max([s.run for s in slabs])
    min_run, max_run = min(min_run, min(road_runs)), max(max_run, max(road_runs))
    run_ran = list(range(min_run-rtol, max_run+rtol+1))
    min_cross, max_cross = min([s.cross for s in slabs]), max([s.cross for s in slabs])
    cross_ran = list(range(min_cross-ctol, max_cross+ctol+1))
    M = [[[[np.inf for l in cross_ran] for j in run_ran] for i in height_ran]
                for k in range(n)]

    # initial values
    for i, h in enumerate(height_ran):
        for j, r in enumerate(run_ran):
            for l, c in enumerate(cross_ran):
                M[0][i][j][l] = cost(h, r, c, slabs[0])

    for k in range(1, n):
        for i, h in enumerate(height_ran): # height h
            for j, r in enumerate(run_ran): # run r
                for l, c in enumerate(cross_ran): # cross c
                    mini = np.inf
                    min_valid_h = h - slabs[k-1].L*slabs[k-1].run*CONV_RUN - htol
                    max_valid_h = h - slabs[k-1].L*slabs[k-1].run*CONV_RUN + htol
                    valid_prev_h = intv_to_range(height_ran, min_valid_h, max_valid_h)
                    for i_prev in valid_prev_h:
                        prev_h = height_ran[i_prev]
                        for j_prev in range(max(road_runs[k]-rtol,0),
                                min(road_runs[k]+rtol,
```

```
                                len(run_ran)-1)+1):
                            min_valid_cross = (h + c*slabs[k].W - prev_h - htol)
                                                                    /slabs[k-1].W
                            max_valid_cross = (h + c*slabs[k].W - prev_h + htol)
                                                                    /slabs[k-1].W
                            valid_prev_crosses = intv_to_range(cross_ran,
                                            min_valid_cross, max_valid_cross)
                            for l_prev in valid_prev_crosses:
                                cost_t = cost(h, r, c, slabs[k]) +
                                        M[k-1][i_prev][j_prev][l_prev]
                                mini = min(mini, cost_t)
                    M[k][i][j][l] = mini

    min_cost = np.min(M[-1])
    return min_cost, M

def intv_to_range(A, low, high):
    ''' returns list of indices k of A s.t. low <= A[k] <= high
        assumes A sorted
    '''
    mini_i, maxi_i = -1, -1
    for k, a in enumerate(A):
        if mini_i == -1 and a >= low:
            mini_i = k
        elif maxi_i == -1 and a > high:
            maxi_i = k-1
    if mini_i != -1 and maxi_i == -1:
        maxi_i = len(A)-1
    ret = list(range(mini_i, maxi_i+1))
    return ret

################################################################
# Part (D)
''' Birth and deaths of slabs and budgets.
'''
start_year = 2019
num_years = 25
start_CO2 = 0.0061*start_year-11.51

def init_slabs(n=1000, mean_age=10):
    if mean_age == 0:
        ages = [0]*n
    else:
        ages = np.random.chisquare(mean_age,size=n)

    slabs = []
    for age in ages:
        ave_ft_cycles = (-0.0842*8.1**2 - .154*8.1 + 13.1)*12
        ft_cycles = ave_ft_cycles*age
        carb_depth = 4.4*np.sqrt(start_CO2*age*365)
        slab = Slab(age=age, ft_cycles=ft_cycles, carb_depth=carb_depth)
        slabs.append(slab)
    return set(slabs)

def does_break(slab):
    ''' return true if this slab breaks now
    '''
    r1, r2, r3 = 0, 0, 0
    if slab.ft_cycles > 500:
        r1 = np.random.rand()
    if slab.carb_depth > 4*25.4: # inch to millimeter
        r2 = np.random.rand()
```

```
    if slab.age > 30:
        r3 = np.random.rand()
    return r1 > .98 or r2 > .98 or r3 > .98

def time_step(slab, CO2, ft_cycles=80):
    slab.age += 1
    slab.ft_cycles += ft_cycles

    carb_depth = 4.4 * np.sqrt(CO2*365)
    slab.carb_depth += carb_depth

def simulate():
    num_sims = 10
    start_blocks = 700
    sim_results = []
    means = []
    live_slabs = []
    for s in range(num_sims):
        live_slabs = init_slabs(n=start_blocks*78, mean_age=7)
        broke_per_year = []
        for k in range(num_years):
            broken_slabs = 0
            y = start_year + k
            CO2 = (0.0061*y-11.51)/1000
            ave_temp = 8.1 + k * (6/80)
            ft_cycles = (-0.0842*ave_temp**2 - .154*ave_temp + 13.1)*12
            for slab in live_slabs:
                time_step(slab, CO2, ft_cycles=ft_cycles)
            for slab in list(live_slabs):
                if does_break(slab):
                    live_slabs.remove(slab)
                    new_slab = init_slabs(n=1, mean_age=0)
                    live_slabs.add(new_slab.pop())
                    broken_slabs += 1
            new_slabs = init_slabs(n=new_blocks_per_year*78, mean_age=0)
            live_slabs.update(new_slabs)
            broke_per_year.append(broken_slabs)
        sim_results.append(broke_per_year)

    means = [np.mean([res[k] for res in sim_results]) for k in range(num_years)]
    costs = [m*22*4*6*(1.02)**k for k, m in enumerate(means)]
    return costs
```