A Dynamic Path Finding Model for UAV surveillance

Tarun Chitra
Matthew Guay
Fadi Laham

Abstract

In this paper, we discuss four plausible strategies for the efficient surveillance of the streets of Manhattan by quadrotor unmanned aerial vehicles (UAVs). We present an implementation of the preferable surveillance strategy: an adaptive routing (**need better phrasing**) procedure which naturally accommodates UAV failure and variable surveillance demand between subregions. Simulations of this procedure are constructed using real-world performance specifications of modern-day quadrotor UAVs and a realistic map of Manhattan, which we use to optimize for the number of UAVs and recharging stations. The results of this optimization form the basis of our final recommendations to the mayor of Gotham City for the overall structure of the surveillance program.

Contents

1 Introduction

For years now, the jaywalking underbelly of society has been running rampant through the streets of Manhattan. An entirely reasonable response to this is creating a ubiquitous surveillance program which covers the entire borough. Since London patented putting a surveillance camera on every street corner in the early twenty-first century, we must use unmanned aerial vehicles (UAVs) instead. In particular, given the municipal government's specifications, the constraints induced by the model of UAV to be used require careful distribution of refueling points and UAV route planning in order to ensure acceptable coverage without introducing inefficiencies. Modern-day quadrotor UAVs are unable to carry large payloads, and accordingly are restricted to relatively low-resolution camera setups which cannot effectively record useful video at heights beyond roughly 500 feet[2]. Top speed of these UAVs moreover does not exceed 30 mph, so a single UAV cannot cover more than 7.5 miles of street continuously without letting sections go unobserved for more than 15 minutes. Since there are over 500 miles of road in Manhattan, this places a lower bound of 67 UAVs required to patrol the streets of Manhattan at all times. However, in order to meet the requirements for failure redundancy, variations in surveillance times, and unpredictability of paths, we must develop a more sophisticated procedure to watch the streets.

We have devised four plans to address this question, each with its own advantages and drawbacks. Broadly, they fall into two categories. Two of the plans seek to find optimal routes for UAVs to take through subregions of Manhattan. The other two have the UAVs maintain a queue of which road sections must be soon observed in order to maintain adequate levels of coverage and provide a framework for the UAVs to determine which paths to take in real time. Both of the latter plans naturally incorporate the ability to deal with varying surveillance levels and individual UAV failure, since no paths are prescribed for all time. The more successful variant of these plans has UAVs seeking out road segments needing observation while taking paths which allow for useful surveillance of intervening segments. After implementing this strategy, we addressed the question of number of recharge stations and UAVs needed to maintain coverage. Our final recommendation is an implementation of the dynamic model discussed below, deploying 1000 UAVs into the city with 250 recharging stations interspersed.

2 Model Overview

2.1 A partitioning approach

In the introduction, the total length of the roads in Manhattan was compared to the length of road a UAV could effectively cover in fifteen minutes in order to get a rough estimate on the number of UAVs needed. We can try to build on this concept to get a useful model that satisfies the requirements of the problem. First, we interpret the streets of Manhattan as forming a graph, with nodes at each intersection and each edge representing a segment of road. Let the length of each edge in this graph be the actual length of the street in miles. The 500-foot flight ceiling of the UAVs gives them a relatively small field of view, approximately 700 ft by 500 ft at minimum zoom. Also, this ceiling means that a UAV cannot usefully see over most Manhattan buildings. By a simple geometric argument, a UAV at the maximum height cannot see the street of an adjacent block unless the building height between them is less than 70 feet. So, we can safely assume that a street is considered "observed" only when a UAV has traversed it.

Given this, we therefore conclude that in order to observe a region with a single UAV, such a UAV must traverse every link in the region. Such a region must contain at most 7.5 miles of road in order to be constantly observed every 15 minutes, and at most 2.5 miles of road in order to be observed every 5 minutes. Therefore, we could plausibly partition the roads of Manhattan into 7.5 mile closed paths and set a UAV to patrol each partition. However, this system does not satisfy the requirements of the municipal government. If a UAV were to fail or needs to recharge, the partition it patrolled would go

Figure 1. A road map of Manhattan. When we zoom in, we notice that around every point there is a collection of streets such the distance between any two locations in the region bounded by these streets is the taxi cab or Manhattan distance.

unobserved until the UAV was repaired. Therefore, we would have to cover each partition by multiple UAVs in order to increase the likelihood of the partition being patrolled at all times. These difficulties in implementation however are much less problematic than the question of finding an optimal partitioning of the road graph of Manhattan. A much simpler problem is the cutting stock problem, which is already NP-hard. Here, we must consider a graph which is nearly gridlike, though not entirely, and our partitions may not even be convex. A sophisticated approach then is necessary in order to find a "good-enough" solution. Both of our first two models use this approach, so it is worthwhile to begin by describing the tools needed by both.

2.1.1 A Partitioning Algorithm

From the road map of Manhattan (Figure 1), we see that in general, the roads of Manhattan do not form a rectangular grid. That is, there exist two points of interest a, b in Manhattan for which the distance along streets between a and b is *not* the taxi cab distance. For example, the distance along Broadway between Columbia (Gotham) and New York Universities is neither the Euclidean nor taxi cab distance. However around every point in Manhattan, there is a collection of streets, for which the distance between every two points contained within the area bounded by these streets. Consider figure 1. We see that ever if we are very close to a place where a road makes a wide turn (such as Broadway near West 108th Street), we can construct such a neighborhood.

From these qualitative remarks and restrictions, one can drastically simplify the mathematics required to construct a successful solution. However, we need to first prove that such a partitioning of Manhattan satisfying the above conditions exists. Since our input data is in the form of images of Manhattan island and road maps, our partitions need to separate regions on this image. As such, the natural mathematical setting for partitioning objects is that of *graph theory*.

2.1.2 From slicing images to cutting graphs

A *graph* is a mathematical object that is made up of two sets — a set of *vertices* and a set of *edges*. Vertices represent points of interest while edges represent connections between vertices. For example, if the set of vertices V is the set of cities in New York State with a population of 50,000 or greater, then we can construct a graph G consisting of edges E that are the roads that connect two cities that are in the set V. Given a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$, we denote the graph G constructed from these vertices and edges by $G = G(\mathcal{V}, \mathcal{E})$. Note that in the above example, we can also add a *weighting* by associating each edge $e \in \mathcal{E}$ with a number, such the distance between the two cities the edge e is connecting.

Combinatorial problems such as the four-coloring problem show that graphs allow for one to systematically partition a set of data based on some characteristics and relationships between vertices. This formulation thus allows us to systematically describe how to partition our images. One of the crucial steps in applying graph theory to real-life situations is the choice of vertices and edges. So for the surveillance problem, how do we convert our image data into a usable graph?

We decided to let the street intersections in Manhattan correspond to a set of vertices $\mathcal{V}$ and the street segments between intersections correspond to a set of edges $\mathcal{E}$. This choice was made in order to better approximate the neighborhoods that need surveillance and because it was easier to extract from the image data given. Moreover, we weighted these edges in $\mathcal{E}$ by their (Euclidean) distance. Since almost every element $e \in \mathcal{E}$ is roughly a straight line, this assumption shouldn't yield any complications. Henceforth, we will consider Manhattan to be the weighted graph $G = G(\mathcal{V}, \mathcal{E})$. In order to partition G, we use an object called a *graph cut*, which intuitively corresponds to a set of streets that cover an "optimal neighborhood." The formal definition of an $s, t-$cut and minimal and maximal graph cuts are a bit cumbersome, but they allow us to use theorems that guarantee us tractable (polynomial time) algorithms. Formally, we have the following definitions [6]:

Definition 2.1. *Let $G = G(\mathcal{V}, \mathcal{E})$ be a weighted graph and $s, t \in \mathcal{V}$. An* $\mathbf{s, t}$***-Graph Cut*** *C relative to s, t is a subset $E \subset \mathcal{E}$ of edges such that in the induced graph $G' = G'(V, \mathcal{E} \setminus E)$, s, t are separated (i.e. there is no connected subgraph containing s, t).*

Hence, we can partition Manhattan by introducing a set of graph cuts such that the union of the induced graphs cover the whole island. Now that we have a technical definition of *what* a partition of Manhattan is, what technical details are necessary to construct such a partition? Moreover, how do we "optimally" partition the city? In the above definition, note that we need to specify vertices s, t, so that any partition we make will depend on a set of distinguishable points in Manhattan. This complicates things severely, since it makes no physical sense to arbitrarily make certain points in the city more important than others.

2.1.3 Energy to the rescue

Luckily, we've formulated our problem in a way that makes prospective solutions amenable to optimization techniques from physics and computer vision. Our goal is to consider the set of all partitions of G relative to any pair of vertices and then pick an optimal partition relative to our problem's constraints. Recall that we have constraints on partitioning Manhattan that are determined by characteristics of the quadrotor and surveillance requirements. We can determine how physically feasible a certain partitioning of Gotham City is by constructing an *energy functional* that takes in geographic data for a partition and computes a number that measures how poorly or how well our partitioning fits our problem. Our goal is to minimize the energy value at each point relative to all possible energy functions. For example consider the graph G consisting of the four corners of Central Park joined by the edges, Central Park East, Central Park West, 59th and 110th streets. If we have a constraint that prefers Northern Manhattan to Southern Manhattan, then an energy function that values the intersections of Central Park East and West with 110th street as less than the intersections with 59th street is valid. Hence our minimal

solution should be something similar to a function that values the vertices on 59th street as 0 and the vertices on 110th street as 1.

In our problem, we the set of all possible partitions of our graph G as a phase space P and construct an energy functional $E[p]$ that acts analogously to the action $S[\gamma]$ on elements $p \in P$. Our problem is more complicated than the classical variational problem for three major reasons. The first reason is that our phase space is discrete, so that the existence and uniqueness theorems for Ordinary Differential Equations cease to hold. Secondly, one of our constraints is purely geometric — we want to consider partitions of Manhattan that locally obey the taxicab or ℓ^1 metric. Finally we will have "discontinuous" boundaries, meaning that the energy functional may not be continuous as one switches partitions. This points out a major assumption hidden in the final implementation of this approach: This plan relies on **restricting quadrotors movements to specific partitions**. As such, we need to answer the following questions:

- How to we choose such a partition?
- Moreover, how to we guarantee that our energy function is minimal, in the sense of the classical variational problem, on each partition?

Before going any further, we should refine our intuitive definition Energy Functional into a more formal definition. In the Computer Science world, much of the underlying intution is stripped away and energy functionals are simply described as functions of labels (i.e. such as a labelling of vertices in a graph). Our perspective is that a better mathematical definition will serve us well.

Let $G = G(\mathcal{V}, \mathcal{E})$ be a graph and suppose $\mathbf{P}$ is a partition of G. Since any partition of G is necessarily finite, we can assume without the loss of generality that the set $\mathbf{P}$ is indexed by a set $I = \{1, 2, \ldots, n\}$, where n depends on the partition. A *labelling* of G relative to $\mathbf{P}$ is a function $f : \mathcal{V} \to I$ such $f(v) = i$ if and only if $v \in P_i$. Hence, to characterize all partitions it suffices to consider the set of all functions that map the set of vertices to a finite subset of $K := \{1, 2, \ldots, |\mathcal{V}|\}$, where $|\mathcal{V}|$ is the number of vertices. From hereforth, for a partition $\mathbf{P}$ with index set$I \subset K$, a partition will refer to the tuple $(\mathbf{P}, I)$ and the size of $\mathbf{P}$ is defined to be $|I|$. This last restriction is required, since it makes no physical sense to have empty elements in a partition. The authors, [6],[5],[7] imply that function spaces of interest follow such a definition. Using their notation, we define $\mathcal{F}^1(G) := \{f : \mathcal{V} \to K | f \text{ is surjective}\}$. Moreover, we can define a function space $\mathcal{F}^k(G) := \{f : \overbrace{\mathcal{V} \times \mathcal{V} \times \cdots \mathcal{V}}^{k} \to K |\ f|_{\mathcal{V}} \in \mathcal{F}^1\}$, where $f|_{\mathcal{V}}$ is the restriction of f to $\{x_1\} \times \cdots \times \{x_{j-1}\} \times \mathcal{V} \times \{x_{j+1}\} \times \cdots \times \{x_k\}$ for some fixed $x_j \in \mathcal{V}$. Intuitively, the set $\mathcal{F}^k$ effectively consists of partition labellings that depend on a vertex and $k-1$ of it's neighbors. Since our energy functional is to choose a partition for us, we want it to be constant on the elements of a partition. This leads us to the definition of a *partition energy functional* as a map $E : \mathcal{F}^k(G) \to \mathbb{R}$. Note that unlike the continuous case, a partition energy functional is not defined to be linear. In fact, since $\mathcal{F}^k$ need not have a vector space or even module structure, we can't employ techniques from functional analysis. From here on, an energy functional will refer to a partition energy functional.

Now if we consider our problem as a local (i.e. nearest neighbor) variational problem, one may guess that it is possible to use local solutions to the variational problem to determine how our partition should look. The idea behind this is to include a term in the energy functional that depends on the local geometry near a vertex. From a geometric perspective, the simplest mathematical function that one can append to an energy functional is a *metric* or *semi-metric*. Once we add in the contribution from the metric or semi-metric, our variational problem provides us with both a minimal path for a quadrotor to follow as well as optimal partitions for how we should divide up surveillance regions. This strategy is precisely what the authors of [7], [6] did in order to solve problems in computer vision. Their goal was to figure out how to optimally partition an arbitrary collection of labelled pixels in an image subject to some constraints. Using their algorithms as a starting point, we know that we can find an optimal partition.

To summarize the above: Our plan of attack is to use an algorithm applied to prospective energy functionals on the nearest neighborhood of a point, to determine an optimal partition of Manhattan

into surveillance areas. This algorithm will in essence be solving the discrete analogue of the classical variation problem for partitions. Once such an optimal partition is determined, we've reduced the phase space of our problem to a "friendly" region of Manhattan, namely a space looks like a gridded space that obeys the taxicab norm.

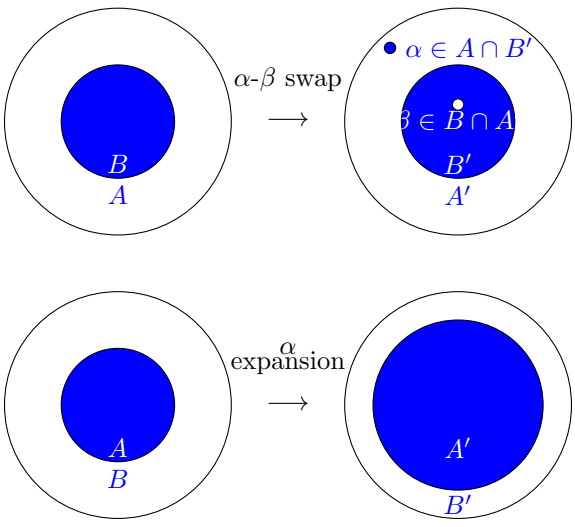
So, we've had a somewhat mathematical discusssion of the problem to be solved. In order to make a model, we need to explicitly define the functions and algorithms in the prior discussion. One important thing to point out before continuing: the problem of minimizing a function of n arguments is NP-hard. This forces us to only consider approximation methods. In practice these are usually easier to implement and give decent solutions. This leaves us with two questions to answer:

- What type of approximations can we use?
- What kind of energy functionals are we to try to optimize?
- What is the explicit polynomial time algorithm do we intend to use?

2.1.4 Optimally partitioning a graph

Let's tackle the first question. As mentioned before, we will likely have an energy functional that is discontinuous on the boundaries of our partitions. In order to side-step this problem, one would hope that an algorithm to solve this problem will take "large steps." Effectively the algorithm shouldn't construct a functional that enforces continuity on boundaries of partitions. Recent work on approximation methods for energy minimization have relied on two "large step" approximations: α-expansions and α-β swaps. Since approximations are crucial to the advantages and flaw of this plan, let's briefly consider how they work.

First, we define α-β swaps by using a graph cut to partition a graph $G = G(\mathcal{V}, \mathcal{E})$. Formally, let G be a graph and suppose that we have a graph cut C relative to two points $\alpha, \beta \in \mathcal{V}$. Let the subgraph containing α be denoted A and the subgraph containing β be denoted B. An α-β *swap* is a new partition of G, A', B' such that $A \cap B' \neq \emptyset, B \cap A' \neq \emptyset$ and such that $\alpha, \beta \notin A \cap B'$, $\alpha, \beta \notin B \cap A'$. This is equivalent to swapping an element of A for an element of B, hence the nomenclature. Similarly, an α *expansion* is a new partition of G, A', B' such that $A \subset A', B' \subset B$. Intuitively, the set A' represents A "growing" into B while B' represents B shrinking in response. These two moves are graphically represented below:



Now note that the discussion about energy functions showed that we can characterize a partition $\mathbf{P}$ with an index set I by an surjective function $f : \mathcal{V} \to I$. Given such an f, there is a unique map $\hat{f}$ associated

to a α-β swap. The map $\hat{f}$ is defined by the following two conditions

- For distinguished $\alpha, \beta \in \mathcal{V}, \hat{f}(\alpha) = f(\beta), \hat{f}(\beta) = f(\alpha)$
- $f(\gamma) = \hat{f}(\gamma), \forall \gamma \in \mathcal{V}, \gamma \neq \alpha, \beta$

. Similarly, if $A \subset \mathcal{V}, B \subset \mathcal{V}$ are the partitions associated with the points α, β, we can define a map $\tilde{f}$ associated to a α expansion by the conditions, $f(A) \subset \tilde{f}(A), \tilde{f}(B) \subset f(B)$. Hence we can define α-β swaps and α expansions in a function form, so that it is possible to use these two moves to constrain an energy functional.

Since we partition a graph by using graph cuts relative to $\alpha, \beta \in \mathcal{V}$, our goal is to find an algorithm that provides an optimal set of $\{(\alpha, \beta)\} \subset \mathcal{V} \times \mathcal{V}$ such that a given energy functional is minimized. Thus given any partition, we claim that it is possible to create a *refinement* of the partition such that our energy functional is minimized. As mentioned in [6], any iterative scheme to refine a partition can be decomposed into a sequence of α-β swaps and α expansions. Moreover, the same authors provide us with two algorithms to optimally refined a partition of size n relative to a given energy functional by using α-β swaps and α expansions. Let G be a graph and suppose that E be an energy functional on $\mathcal{F}^k(G)$. Then we have the following algorithms [6]:

Algorithm 1 α-β swap partition optimization

1: **initialize** Choose an arbitrary partition **P** of the graph G.
2: **initialize** Let $I = \{1, 2, \ldots, n\}$ be the index set of the partition
3: **initialize** Let $\hat{f} : \mathcal{V} \to I$ be an arbitrary partition function for **P**
4: **set** Success := 0
5: **for** $\alpha, \beta \in I$ **do**
6: **let** $\hat{f} = \operatorname{argmin} E(f')$ for f' within one α-β swap of f
7: **if** $E(\hat{F}) < E(f)$ **then**
8: **set** $f := \hat{f}$ and Success := 1
9: **end if**
10: **end for**
11: **if** Success = 1 **then**
12: **goto** [4]
13: **end if**
14: **return** f

Algorithm 2 α expansion partition optimization

initialize Choose an arbitrary partition $\mathbf{P}$ of the graph G.
initialize Let $I = \{1, 2, \ldots, n\}$ be the index set of the partition
initialize Let $\hat{f} : \mathcal{V} \to I$ be an arbitrary partition function for $\mathbf{P}$
set Success := 0
for $\alpha, \beta \in I$ **do**
  let $\hat{f} = \operatorname{argmin} E(f')$ for f' within one α expansion of f
  if $E(\hat{F}) < E(f)$ **then**
    set $f := \hat{f}$ and Success := 1
  end if
end for
if Success = 1 **then**
  goto [4]
end if
return f

In [6], [7], these algorithms are proved to be optimal for all $\mathcal{F}^k$ and moreover they converge in polynomial time. In fact, [Zabih] proved that for a partition of size k, the algorithm concludes in $\mathcal{O}(k)$ iterations. The realistic validity of this approximation has been proven in over 1000 computer vision applications.

2.1.5 Construction of a Surveillance-friendly Energy Functional

Now that we have a method to construct a partition, *given* an energy functional we come to a pragmatic question — what energy functional do we choose? As mentioned before, our optimal partition is subject to constraint of providing a locally taxi cab metrizable neighborhood. Let's first construct an energy functional that provides a way of measuring how close a given partition is from being a taxi cab neighborhood. Then, once we lay out what our explicit constraints for the surveillance problem on a neighborhood are, we can edit this functional to provide us an optimal solution.

Recall that in the graph representation of Manhattan, each node represents a street intersection and that every street intersection has an associated Latitude and Longitude. Since we can naturally embed this graph into $\mathbb{R}^2$ by mapping each intersection into the tuple $(\text{Latitude}, \text{Longitude})$. Let $\iota : \mathcal{V} \to \mathbb{R}^2$ be the aforemented embedding. Then taxi cab distance $d : \iota(\mathcal{V}) \times \iota(\mathcal{V}) \to \mathbb{R}$ between two nodes x_1, x_2 with embedded coordinates $(\text{Latitude}_1, \text{Longitude}_1), (\text{Latitude}_2, \text{Longitude}_2)$ is simply,

$$d(x_1, x_2) = |\text{Latitude}_1 - \text{Latitude}_2| + |\text{Longitude}_1 - \text{Longitude}_2|$$

Since our graph is weighted with the street distances between two points, a natural energy functional to consider is one that looks at the difference between minimal distance along the graph and taxi cab distance. Thus, if we can define a graph distance then we have a prospective energy functional. Since our graph of Manhattan is connected, we know that there exists a graph metric $d_g : G \times G \to \mathbb{R}$ [Citation]. Given these two functions, we know want to construct an energy functional approximates the difference between d, d_g over some partition $(\mathbf{P}, I)$. Let $f^{\mathbf{P}}$ be the function associated to a partition $\mathbf{P}$ and let $\delta_{i,j}$ be the Kronecker delta. Then a prospective energy functional E_{taxi} is,

$$E_{\text{taxi}}(f)(x_1, x_2) := \sum_{x_i \in \mathcal{V}} \sum_{x_j \in \mathcal{V}} \delta_{f(x_i), f(x_j)} \left(d(\iota(x_i), \iota(x_j)) - d_g(x_i, x_j) \right)$$

This functional effectively looks at the difference between the taxi cab and graph metrics on the partition defined by a function $f \in \mathcal{F}^1$. Hence minimizing E_{taxi} will provide us with a function $\hat{f}$ that determines the taxi cab neighborhoods of Manhattan. In the rest of the paper these partitions will be denoted "Larger Partitions."

Another possible partitioning scheme is to construct partitions that a single UAV can traverse entirely. From the UAV data sheet [**UAV DATA**], a UAV has a range of roughly 7.5 miles. Hence, we want to find an optimal partition of Manhattan into 7.5 mile "blocks." Note, however, that there is no restriction that a block be convex, since it is possible for a UAV to traverse an L-shaped region. Luckily, the aforementioned algorithms were invented as ad hoc replacements for convex optimization in computer vision. Since the function we receive from the graph cut methods is discontinuity preserving, non-convex sets are allowed. In fact, it is easy to see that many partitions constructed only by using α-β swaps will not be convex. Hence, we only need to find an energy functional that gives us arbitrary 7.5 mile subgraphs of G. The energy functional $E_{7.5}$, defined as,

$$E_{7.5}(f)(x_i, x_j) = \sum_{x_i \in \mathcal{V}} \sum_{x_j \in \mathcal{V}} \delta_{f(x_i), f(x_j)} |d_g(x_i, x_j) - 7.5|$$

fits these requirements. Partitions constructed using this energy functional will be referred to as "smaller partitions" from here on.

So, using this technique, we could compute a reasonable approximation of an optimal partitioning of the Manhattan streets. Call the graph of the roads of Manhattan G, and the set of regions in a partition $\mathcal{P} = \{P_i\}$. This is only half of the problem, however. Once an (approximate) partition has been constructed, we still must decide what to do with them. Since every segment must be traversed, a natural plan is to find a closed path which traverses each link in the partition which is as short as possible. Thus, we can cast our problem as a traveling salesperson problem in a natural way: For each partition graph P_i, let P_i' be a graph such that the node set V_i' of P_i' contains a vertex for each edge in P_i, and two nodes $x, y \in V_i'$ are joined if the corresponding edges in P_i are adjacent. A solution to the traveling salesman problem on P_i' then gives us a route ρ_i^* for UAVs to take through P_i. Note however, that again the traveling salesman problem is NP-hard, and so if the size of the vertex set of P_i is large, it may be intractable to solve this exactly. So, we may have to resort to an approximate solution. Many good approximation algorithms exist[9],[10], and so we can use one to generate an acceptable path through the region. Let us call this $\hat{\rho}_i$. Then, we set some UAVs about following this path, with enough UAVs on each $\hat{\rho}_i$ so that the probability of all three being down for repair is low enough.

Of course, there needs to be a better understanding of what this probability is in order to address this point. There are many distributions traditionally used to model failure rate of mechanical devices, the two most common being the exponential and Weibull distributions. Ideally, we would model the interfailure times as being exponentially distributed with rate parameter $\frac{1}{\mu}$, where μ is the mean time-to-failure. However, with a graph as large as the road layout of Manhattan, computational issues dominate, and we simply did not have the computational resources to let simulations run long enough for such a failure model to be useful. Therefore, instead we model the UAVs in two scenarios. One, at 100% capacity, where all UAVs assigned to each partition work perfectly. In the other scenario, we examine their behavior at 70% capacity, with 30% of UAVs removed at random without regard to partitions. This framework almost entirely characterizes the first two plans, the only variations being in partition size and correspondingly, the number of UAVs needed.

2.1.6 Plan 1: Larger partitions

Assume that we have some partitioning scheme that has given us a partition $\mathcal{P}$ of our graph. With k UAVs in each partition running at all times, the probability that a region loses all UAVs at 70% capacity is $(0.3)^k$. Then, if we want this probability to be no greater than 0.1%, we must have $k \geq 4$. This, however, does not take into account recharge time. With careful planning, we can make sure that only one UAV is recharging at any point, since it takes 30 minutes. So, we in fact need 5 UAVs in this partition in order to have 4 always running. Suppose then we decide that each region should have 5 UAVs policing it at full capacity, how large should each region be? Since each UAV can traverse 7.5 miles in 15 minutes and 4 are always in the air at any time, through the areas of Manhattan that need

to be surveiled every 15 minutes, we should have each group of five UAVs cover 30 miles of road. If they follow the path through the region while maximally spaced apart, the entire region will be observed every 15 minutes as specified even with one charging. Similarly, in regions that must be surveiled every 5 minutes, 5 UAVs behaving in the same way will cover 10 miles of road, and in Central Park we can get away with 5 UAVs covering 40 miles of road. So, we proceed as follows: We first subdivide our graph G into three regions: G_1 the graph containing all edges which must be surveyed every 15 minutes, G_2 containing the edges which must be surveyed every 5 minutes, and 20 minutes for G_3. In G_1, we partition the area, using the techniques described previously, into subregions $\{P_i^1\}$, each containing at most 30 miles of road. Similarly, in G_2 we partition into regions $\{P_i^2\}$ containing 10 miles of road, and 40 mile regions $\{P_i^3\}$ in G_3. We then solve or approximate a solution to the TSP for each region P_i^j, and then put 4 UAVs on each P_i^j following $\hat{\rho}_i^j$. This gives us good coverage of G, and moreover assures that no section of the graph will ever be too far from a UAV. However, should more than one or two UAVs fail in a region, the streets could go unpatrolled for an unacceptably long period of time. For the municipal government with more money to spend, we present plan two.

2.1.7 Plan 2: Smaller partitions

The setup of the second plan proceeds similarly to the first. As before, we divide G into G_1, G_2, G_3, but now instead of dividing the regions into 30, 10, and 40 mile partitions as before, we will divide them respectively into partitions of sizes 7.5, 2.5, and 10 miles. Each such partition can be covered efficiently by 1 UAV, so by placing 5 in each region, we effectively guarantee that, at 70% capacity, each region in fact has only a 0.01% chance of being unobserved for longer than the required time. However, this does scale up the number of UAVs needed rather dramatically.

2.2 A dynamic approach

Our final plan focuses heavily around creating a full-scale tractable computer simulation of a surveillance scheme for Manhattan. Unlike the previous plans, the following plan proposed is non-deterministic and non-optimal. The scheme does not attempt to create a perfect theoretical course of action. Instead, a scale graph of Manhattan is imported into a computer simulation with streets represented by graph edges. UAVs are "flown" through the graph one step at a time to visit each edge as needed. The algorithm itself is a straightforward greedy algorithm; each UAV in the simulation checks the reachable edges of the graph near it and selects the edge with the greatest priority (hadn't been visited in the longest time). The UAV plots a weighted path using Dijkstra's algorithm to the edge in question and steps towards it each clock tick of the simulation.

The simulation is written in the Java programming language and takes heavy advantage of object oriented mechanics. The roads, UAVs, map, and refueling stations are all self-contained objects that interact with each other as they would in a physical setting. The program was built with the idea of high tractability close in mind, allowing us to modify and run the simulation over multiple variables to optimize for greatest effectiveness and cost minimization.

2.2.1 Simulation Objects/Structures

To best take advantage of the object oriented programming infrastructure Java has to offer, the entire project was broken up into multiple individual objects for organization clarity and running speed. These objects interacted with each other just as one would expect them to in a real physical setting; for instance, UAV objects flew along the length of RoadSegment objects, and refueled at FuelingStation objects.

Graph Data

To build a full-scale graph framework of Manhattan for the simulation to run over, road data was gathered and imported from the Topologically Integrated Geographic Encoding and Referencing System (TIGER). The encoded road data files were parsed and a to-scale physical graph of Manhattan was built in a Java environment. In the constructed graph, nodes represented road intersections and graph edges represented road segments. Aside from standard road values such as length, starting and end locations (in latitude and longitude), individual road segments held an individual delay threshold and delay counter. The delay threshold was the maximum amount of permissible time the road can go unobserved. Since each road contained its own unique threshold value, the program could be easily modified to allow some districts with heavy traffic to hold lower thresholds than other districts with mild traffic. The delay counter holds the time since the road was last observed by a UAV. If the delay counter ever reaches the delay threshold, an error is thrown, and the simulation reports a failure.

As previously explored, the flight ceiling of the UAVs is not high enough to see substantially far over buildings. Furthermore, considering the limited field of view of the UAV cameras, it is fair to assume the UAVs follow paths along streets and cannot cut over geographic areas to maneuver between streets for best efficiency. With this assumption, we can model UAV motion by traversing paths along the graph data structure. With this limit on motion in place, for all roads to be observed, we require that Manhattan must be strongly connected. Fortunately, this is indeed the case and Manhattan roads are strongly connected. The only potential problem is Governor's Island off the Southern coast of Manhattan. Governor's Island is a small island separated from the rest of Manhattan with access only by ferry. The island roads are foot paths only however; the island is a vehicle-free zone with driving prohibited[3]. For this reason, Governor's Island was omitted from the graph data structure.

Finally, assuming that UAV's fly above traffic level and need not obey road directions, the one-way or two-wayness of streets is ignored. The UAVs may travel in both directions freely down streets, making the graph structure naturally undirected.

UAVs

UAVs are modeled by individual instances of a UAV object. Each UAV object holds information relevant to that plane: the current amount of fuel remaining, the maximum fuel tank, the recharging rate, a pointer to the current edge the UAV is on, and finally an ordered stack of edges representing a path the plane is assigned to fly in.

A UAV object can be in one of three states: "busy", "recharging" and "idle". As the names suggest, a busy UAV is one that currently has a path assigned that it is traversing, and a recharging UAV is one that is currently standing still at a refueling station. The idle state is an intermediate state for a UAV that has just finished refueling or traversing its assigned path and needs to be assigned a new job. The idle state is only used internally by the program to signal the simulation that the UAV in question needs a job, UAVs do not actually sit idly for any period of time.

Recharging Stations

Each edge in the graph data structure contains a boolean variable declaring whether or not the edge contains a recharging station. For a UAV to recharge, it must fly to an edge equipped with a station and remain there until recharging is complete (30 minutes for a 5 hour battery). To simplify the simulation, we assume a recharge station has no bound on the number of planes it can charge at any given time. This is not unreasonable considering UAVs are very small and many can fit on a rooftop. Furthermore, to charge a UAV, an extra power chord needs only to be extended to the low battery UAV; power cords are very cheap and many spares can be placed at the same station.

2.2.2 Algorithm

The algorithm for planning UAV motion is at the core a simple greedy algorithm. There are no deterministic routes each plane is assigned from the start. Instead, at each moment of time, a plane's state is determined by the state of the entire map instance. Routes are planned dynamically depending on the map at the time of plotting, making the UAV routes sufficiently random and unpredictable to pedestrians and employees alike. Furthermore, since routes are plotted dynamically, if a plane spontaneously fails, the remaining planes can very easily cover for the failure without adapting their core strategy.

When a plane is idle, the strategy in a nutshell is to find the road within reachable range that is closest to reaching its observation delay threshold. The plane queues a path to the road segment in question, and traverses the path to it, observing each road along the way. The following sections explore in detail the queueing process of the simulation.

Queueing

If at any time a UAV is in the idle state, it is immediately assigned a road to fly to. For a single plane, the assignment process begins by creating a set of reachable edges. An edge is considered reachable if the plane can reach it before the edge reaches its threshold, and a path from the UAV's current position to the destination edge plus the distance from the destination edge to the nearest recharge station will cost less than the UAV's total remaining fuel. That is, an edge is reachable if the UAV can fly to it in time and still have enough fuel left to fly to a recharge station if needed.

Determining shortest path between each point in a graph is at best $O(n^2)$ in the number of vertices. Repeating this operation for all edges to find the set of reachable edges is very costly. To speed up the process, the distance from the plane's current edge to the destination is approximated with a taxicab distance rather than crawling the graph and finding the actual minimum length path. Also, at initialization, the simulation pre-computes the exact path lengths to the nearest refueling station for each edge. This reduces the computation to determine if a node is reachable to constant order.

When constructing the set of reachable roads, road segments queued by other UAVs are omitted. This is done to avoid assigning multiple planes to the same edge and wasting resources.

Once the set is constructed, the UAV selects the edge with a current unobserved time counter closest to its threshold. A shortest weighted path is constructed using Dijkstra's algorithm for this edge. The exact path algorithm is described in the next section. If the path constructed is unreachable with the current remaining fuel, the edge is thrown away end the next highest priority edge is considered. This is repeated until an edge is found or the UAV runs out of edges to consider. If the UAV has no edges to consider, either the reachable set was too small to begin with (meaning the UAV has too little fuel left), or all adjacent edges are being taken care of by other planes. In either case, the plane is assigned to refuel since it is either low on fuel, or is not needed so it may as well use the opportunity to refuel.

When a reachable path is produced, the plane appends the ordered list of edges to a stack internal to the UAV object. The UAV marks all of the edges in the path as "queued." This notifies all other UAVs that the edges in question will be visited before they reach their threshold. This way other UAVs would not waste time attempting to observe edges that we already know will be observed before their threshold.

The queueing method is summarized below in Algorithm 3.

Path Construction

A weighted graph is one in which each edge holds with it a numerical weight. In the case of graph edges representing road segments, the obvious weight is the length of the segments. Given two points on a

Algorithm 3 Queueing Algorithm

input UAV p
initialize Let E^r be the set of reachable edges
while $|E^r| > 0$ **do**
  $\hat{e} = \max\{e : e \in E^r\}$
  Construct a path, $\mathcal{P} = \text{Dijkstra}(\hat{e}, p)$
  if length($\mathcal{P}$) < fuel(p) **then**
    return $\mathcal{P}$
  end if
end while

graph, Dijkstra's algorithm is a well-known algorithm for computing the path between the points that minimizes the total weight of the path's edges.

In the simulation, we extend the weight function to incorporate multiple factors beyond just distance. One problem specification claimed that turning the planes should be avoided if possible. This naturally suggests weighting paths with turns heavier than ones without them. Two other factors considered were the edges' observation delay, and whether or not the edges have been queued by another UAV.

Turning Factor: As the path is constructed, the angle the path needs to turn to go from the current edge to the next edge is used as the weight for the next edge. The angle is divided by pi to scale the weight so a straight path is 0, and a completely reversing direction is 1.

Queuing Factor: When constructing paths, we ideally want to hit as many edges unaccounted for as possible. This way, when constructing a path we also visit as many edges that need observation as possible. This weighting attempts to keep the UAVs evenly distributed and following paths that overlap as little as possible. An edge that has been queued by another UAV is weighted heavily as 1, and lightly as 0 if it is not queued.

Threshold Factor: When given the option between to unqueued roads along the path, we would ideally want to visit the one closer to its delay threshold. The threshold factor is a scaling from 0 to 1 of each edge's current delay, $(d_{thresh} - d_{cur})/d_{thresh}$. Edges are weighted lightly as 0 if the edge's current delay is equal to it's threshold, and heavy as 1 if its delay is 0.

The final weight function is a scaling of length as well as a linear combination of the other three factors mentioned.

$$w = \lambda_1 \cdot t_{len} + \lambda_2 \cdot t_{turn} + \lambda_3 \cdot t_{queue} + \lambda_4 \cdot t_{thresh} \tag{1}$$

The values of λ_i were tweaked manually for greatest performance in the shortest path algorithm.

2.2.3 A comparison

The first two plans, derived from the partitioning approach, have some advantages over the dynamic model. In its current state, the queueing procedure is not optimized for individual UAVs to cover multiple street segments in close proximity with roughly the same time remaining until their threshold. This can lead to "clumping", as multiple UAVs converge on a relatively small area to observe nearby road segments. Since paths are fixed in the first two plans, no such clumping can occur, so there is no chance of reaching a state in which all UAVs have moved too far away from some segment of the graph to observe it before it reaches its threshold. From a practical viewpoint, the deterministic nature of the first two plans also allows the system to be analyzed much more easily: The dynamic model is equally deterministic, albeit more chaotic, but due to our inability to search the entire (or a reasonable portion of) configuration space for recharge stations throughout the city, each run places them at random throughout the graph, so in the limited time we have it is more difficult to determine what numbers of recharge stations and UAVs are sufficient to maintain adequate coverage. The tradeoffs between plans

1 and 2 are clear: Plan 2 guarantees better coverage, but requires roughly four times as many UAVs to implement.

However, the first two plans have their own share of substantial conceptual limitations which limit their appeal in comparison with the dynamic approach. Since the UAVs in each region do not communicate with UAVs in other regions, each region must be multiply covered in order to allow for redundancy in case of UAV failure. With a low enough failure rate, it should be much more effective to let certain UAVs move from region to region in case of failures. Workarounds sharing some UAVs between certain regions involve solving additional partitioning problems and call into question the fundamental partitioning strategy. Moreover, the inability of UAVs to change the paths they follow can lead to absurd situations, in which one partition loses all UAVs in some trial at 70% capacity, and nearby regions with multiple UAVs remaining are unable to send one to the unsurveiled area. The third plan does not have these limitations, as UAVs can move freely throughout the entire area. This partitioning strategy also requires that we separate regions based on how frequently they have to be observed, while the dynamic approach handles these regions fluidly without any distinction between them. Moreover, one of the requirements was that program planners not be able to avoid UAVs given knowledge of their operation. Under the first two plans, UAVs can be avoided easily if one knows the paths they are taking. Since paths are not predetermined, it is very unlikely that anyone designing this system could easily predict the whereabouts of a UAV at a future point in time. Due to these problems and the time constraints induced by the competition, we did not implement either of the first two models, and instead chose to focus on the development of the third model.

3 Assumptions and justifications

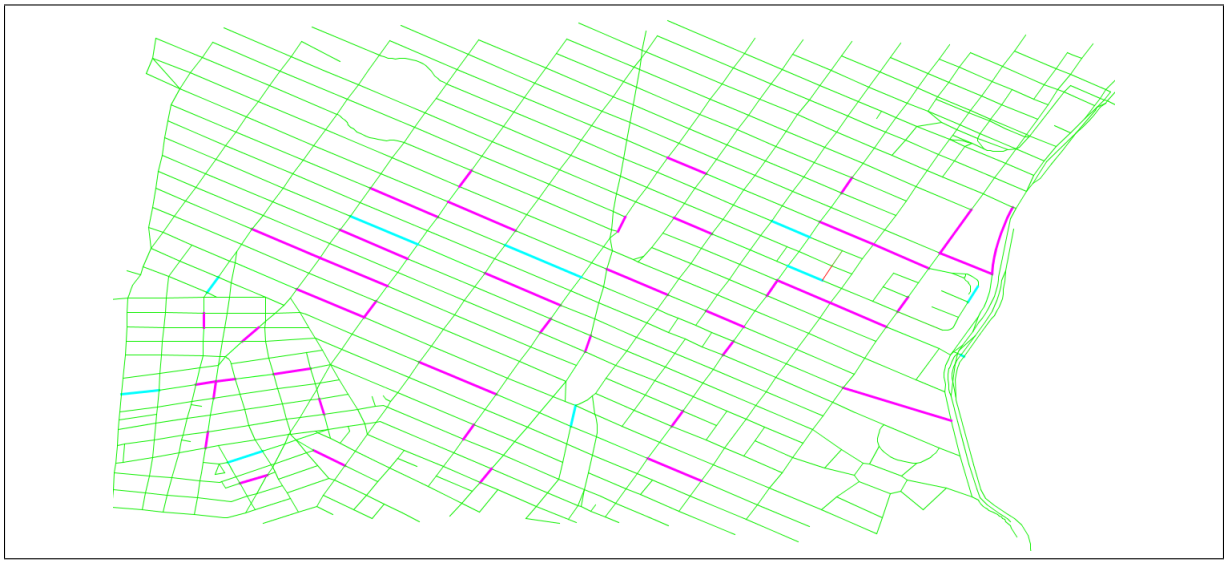
Assumption	Model 1	Model 2	Model 3	Model 4	Justification for Assumption
There are no bounds on the number of planes	×	×	✓	✓	The planes are extremely small and as such they can fit on a roof top. The only supplies needed to charge them are an electric outlet and a charging station (which is also rather compact). Moreover at roughly $32,000 per plane, this fiscally reasonable assumption when compared to the long-term cost of human enforcement.
There are no bounds on the number of charging stations	×	×	✓	✓	A similar line of reasoning as above applies. Since there are no manufacturer provided charging stations, we surmise that a charging station will effectively be a secured access point with plenty of electric outlets. The cost of such a set up is negligible and is even cheaper than that of a cell phone tower.
Manhattan, when represented as a graph, is a connected graph.	✓	✓	✓	✓	This assumption is only true if we discount the contribution from Governor's Island. However Governor's Island is a recreational area and its roads are only used for maintenance vehicles. For ordinary citizens, the only methods to get to Governor's Island are via Ferry or subway.[**Gov. Island ref**] Hence we may safely ignore the contributions of the island. This assumption is important connectedness of the graph gives us a unique, natural graph metric on the one connected component [**Graph Theory/Comb. Book**].
Planes cannot fly over buildings	✓	✓	✓	✓	This assumption is validated by both geological data from the City of New York [**New York Geological Data**] as well as a practical consideration. The geological data says that the average height of a building in Manhattan is 142.6m or 467.85ft, so that roughly 50% of the time, the building would be taller than the UAV's ceiling height. Moreover, while the UAV's could hover over some buildings, there is an associated cost in terms of time and fuel while a UAV is over a building. This loss in surveillance capacity is large enough that it makes more sense to have the UAV's traverse along roads.

Assumption	Model 1	Model 2	Model 3	Model 4	Justification for Assumption
The field of view of the UAVs is restricted to the road segment the UAV is traversing	✓	✓	✓	✓	As mentioned in the beginning of section 2.1, one can use a simple trigonometry argument to show that the maximum height that a building can be in order for a UAV to peer over a street is 70ft.
The UAVs are strictly restricted to partitions or neighborhoods of Manhattan	✓	✓	×	×	This assumption is not necessarily valid, but it simplifies our the problem significantly. Assuming an optimal partition allows for us to compute an approximation to a deterministic, analytic solution, as described in section 2.2. Note, however, that it is possible for this solution to be optimal

4 Results and Conclusions

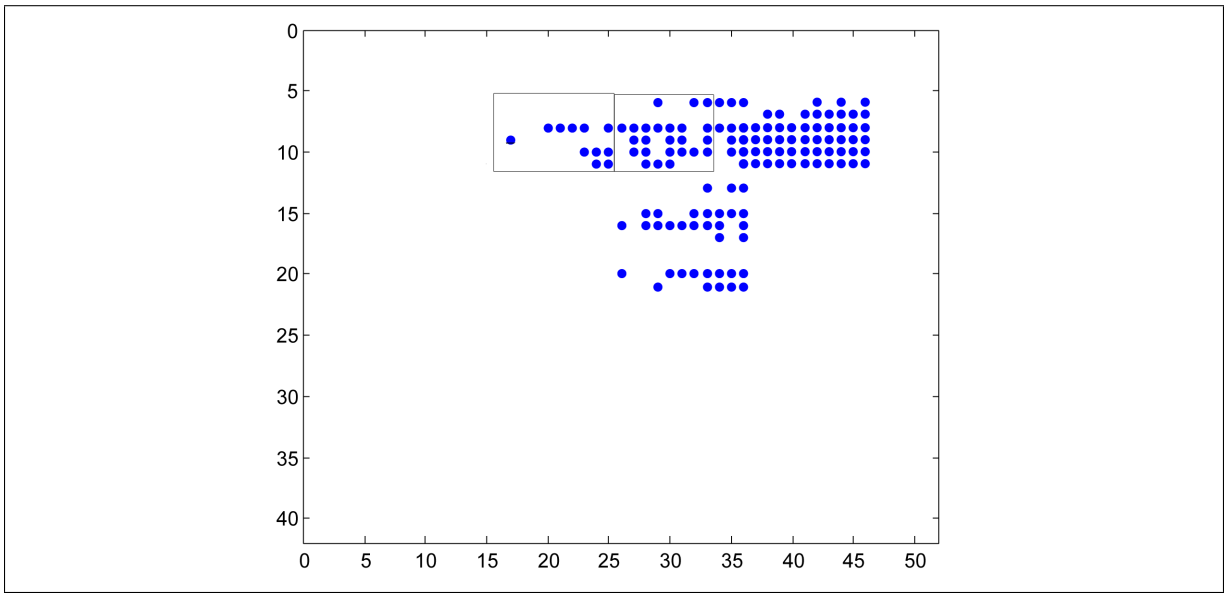
4.1 Results

Two drawbacks were faced by both modeling strategies attempted here. The first was the scale of the problem: the Manhattan road network is just barely too large to efficiently simulate on the computers available. Second, and related to the first problem, the parameter space was too large to search well given time constraints. In particular, for the dynamic model, we were unable to optimize for placement of recharging stations, only their number, which were then placed randomly throughout the graph. The strategy used to estimate the number of UAVs and recharge stations necessary was to solve the problem more thoroughly on a subdomain of the full graph, picture below, and then extrapolate.

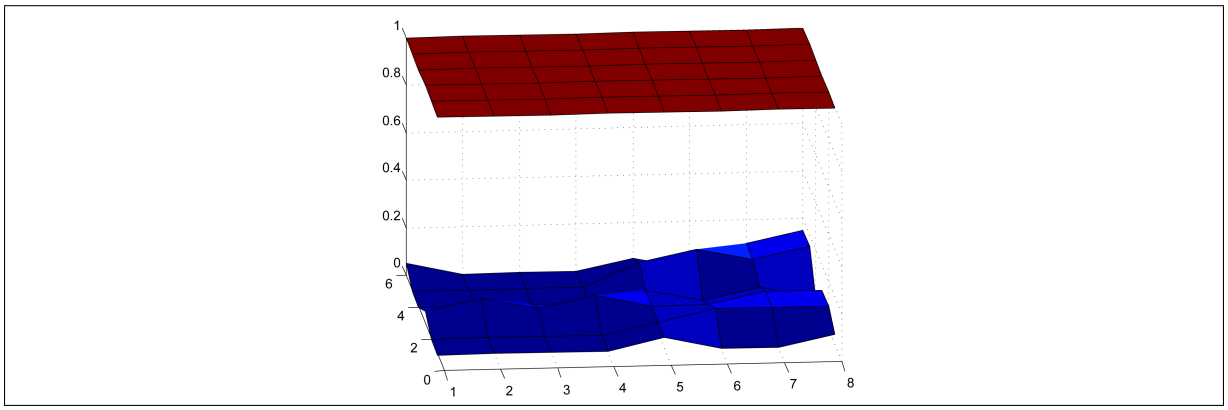


Each run of the equivalent of 12 hours simulated time took about 1 minute to run, and so optimization was carried out in the following way: The overall goal was to find a combination of number of charging stations c and UAVs that minimized total UAVs u, kept charging stations at a "reasonable" quantity, and allowed the simulation to run successfully with high probability. Observations of multiple runs with the same number of chargers and UAVs led us to conclude that charger placement strongly affects simulation

performance. Therefore, in a region in this parameter space with high success rates, it is reasonable to conclude that optimizing for charger placement given a fixed number of chargers (discussed further in future work) would lead to even more consistent results. At first, we did one run on each parameter combination for $c \in [15, 30]$ and $u \in [115, 145]$. This gave us the following picture, where a blue dot represents success at that parameter combination:

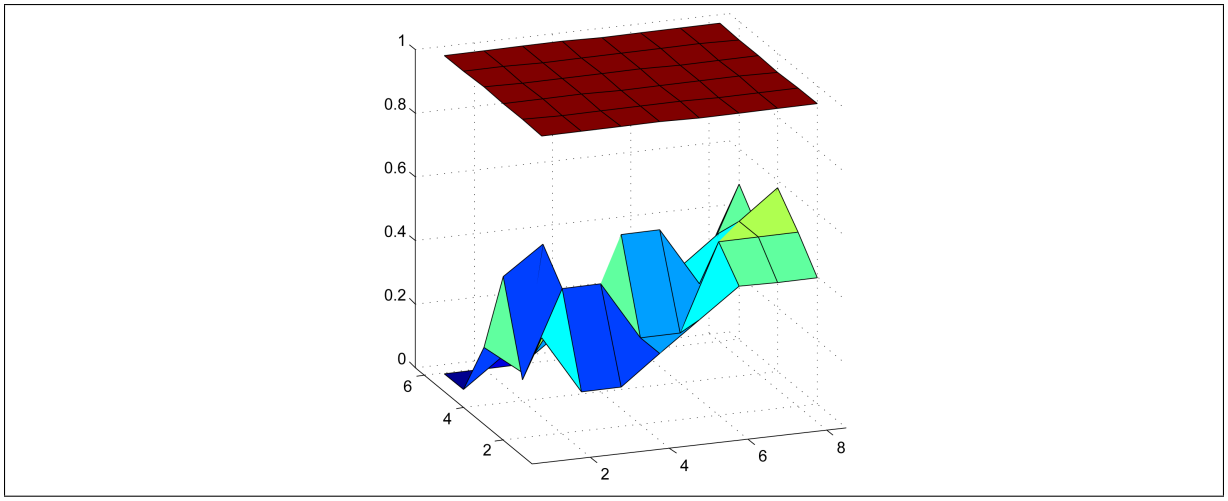


Two areas within this parameter space were explored further. The first, with $c \in [15, 20]$ and $u \in [115, 122]$, proved not to be useful. This is the area enclosed in the leftmost rectangle in the above figure. 20 runs over the parameters in this rectangle produced probabilities of success not greater than 20%. The following picture demonstrates this, with the back-left corner representing $c = 15, u = 115$, and the front-right corner representing $c = 20$, $u = 22$. The red plane is $P = 1$, for reference.



The second area explored had $c \in [15, 20]$ and $u \in [125, 132]$. This gave much more useful parameters. In particular, after 11 runs we had 77% success with $c = 17$ and $u = 132$. The entire space looked like

We deemed 77% success to be acceptable, given that with more time we could optimize placement of the charging stations to increase success rate.



4.2 Conclusions

Given this, we derived results for the entire graph as follows: The subgraph used here contained 1229 edges, while the full graph has 8402. Since the graph is roughly self-similar, the required chargers and UAVs ought to scale linearly with number of edges. This would give an estimated 900 UAVs and 115 charging stations needed for all of Manhattan, assuming everything was observed at 15 minute intervals. However, to compensate for the areas where 5 minute intervals are needed, more UAVs are likely necessary. To account for this, we adjusted the initial guess for number of UAVs needed up to 1000, and the number of chargers to 250. While we were unable to run a full simulation on the entire graph, we were able to simulate 15 minutes at these numbers, during which time there were no portions of the road that went unobserved longer than specified. We can therefore conclude that using the dynamic model presented in this paper, along with a system of 1000 UAVs and 250 chargers placed throughout the city provides an infrastructure to efficiently observe the city in a manner which meets all specifications of the municipal government.

5 Future Work

While the models presented in this paper do depict some progress in solving the surveillance problems, both the partition-based and dynamic methods are amenable to significant improvement. Our future work focuses on the following improvements:

- Develop and test partition-based methodologies
- Improve dynamic method's implementation
- Develop hybrid solutions that incorporate both partition-based and dynamic methodologies
- Better implementation of UAV failure rates
- Better exploration of Parameter Spaces for optimizing path selection
- An improved queueing system to minimize cluster

Let's start with the partition-based implementations. The glaring failure of the partition-based procedures presented is that they were not developed or tested. We initially believed that these methods were computationally intractable given our meagre computing resources, since the methods of sections 2.3, 2.4 relied on approximating solutions to NP-Hard problems. Moreover, we only discovered that for a

partition of size n, the partitioning algorithm ran as $\mathcal{O}(n)$. The authors of the algorithm provided a proof of concept implementation in Matlab [6] and as such we realized toward the end of the competition that a partition-based implementation is possible. Since our entire code-base was in Java, we decided to stick with the dynamical methods that had already been coded up. However, we still came up with a list of questions for the partition-based methodologies, namely:

- Does an efficient partition-based implementation for the surveillance problem exist?
- Can a partition-based scheme yield a better lower bound on the number of UAVs necessary?
- Is a partition-based scheme provably worse (or better) than a dynamic scheme?

While these questions are fundamental to the implementation and understanding of partition-based methods, future work on the dynamic method will be more detail-oriented. Due to time constraints, we were unable to incorporate a number of important features into our dynamic model. One of the stated problem objectives was to evaluate any implementation for failure (as mentioned in latter portion of section 2.2) and our model was unable to accomodate an implementation. While we developed a theoretical way for this implementation to run, we ran into coding difficulties and were ultimately forced to abandon failure rate analysis. As discussed in section 2.2, we hope to incorporate failure analysis by having simulations run for long enough periods of time so that the overall distribution of failed UAVs is observable. We believe that given enough time to run our simulations, we can show that the dynamic method has an acceptable failure rate tolerance.

Another important area to delve into is the choice of weights in our shortest path computation function (Dijkstra's algorithm). Recall that our weight function on each edge was computed by a linear combination of four factors. The constants used in the linear combination were picked manually after sampling a large number of feasible weights. While this *ad hoc* procedure gives a good heuristic of what the optimal parameters should be, we hope that we can develop an optimization method that can pick optimal parameters.

Finally, during our simulations we noticed that our queueing procedure was far from optimal. Since queueing is a crucial part of finding a stable solution, we feel that any improvement to our queueing system will reap immediate benefits. Some of the improvements to our queueing system that we noted would help the efficiency and accuracy of our simulation include:

- We want the queue to take into account *both* the threshold time and the distance a UAV is from a node that needs to be visited. Currently, our queue only accounts for the threshold time, which is time until an area hasn't been monitored in the past 15 minutes. This often leads to UAV's traversing long graph distances to satisfy the queue. We attempted to add exponential distance factors to this aspect of the queueing mechanism; however, we ran into some computational difficulties. With more time, we feel that such a mechanism can be easily implemented
- At the moment, our queue has all of the UAV's making future path decisions via a greedy algorithm. Ideally, we would like for the UAV's to be able to communicate so that UAV's that are closer to certain nodes of interest can be deployed instead of UAV's that are farther away. While this may be challenging to code, it would likely give us far better solutions. We noticed that occassionally under our greedy algorithm scheme, the UAV's would "compete" for nodes of interest and cluster. This is certainly not the behavior we are looking for, and a complete solution to the problem would be excised of this issue.

In an effort to bridge the deterministic viewpoint of the partition methods and the probabilistic viewpoint of the dynamic method, we also considered hybrid models. These models would first partition Manhattan into subgraphs that either met the big partition or small partition condition. Then we would apply a dynamic model on these subgraphs in such a way that UAVs are allowed leave their "home" partition, but cannot leave neighboring partitions. For example, if we put a UAV in the financial district, then it would be allowed to move through Chinatown, SoHo and NoHo, but could not leave these areas. By combining these methods, we hope that we can arrive at a tractable solution that is better than either

the partition or dynamic solutions. Moreover, a hybrid model would have all of the benefits of the deterministic and probabilistic model, while isolating their flaws to the small subgraphs. Again, this implementation was time intensive and as such we could not pursue it. We hope that by combining a hybrid model with the other forward-looking suggestions, we can arrive at a unique, optimal solution for the surveillance problem.

References

[1] (2010a).

[2] (2010b).

[3] (2010c).

[4] Aldo Jaimes, Srinath Kota, J. G. (2009). An approach to surveillance an area using swarm of fixed wing and quad-rotor unmanned aerial vehicles uav(s). *Haha probably none*, 1:123–125.

[5] Boykov, Y. and Kolmogorov, V. (2003). Computing geodesics and minimal surfaces via graph cuts. In *International Conference on Computer Vision*, Nice, Frace.

[6] Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1–17.

[7] Freedman, D. and Turek, M. W. (2010). Graph cuts with many-pixel interaction: Theory and application to shape modelling. *Image and Vision Computing*, 28(3):467–473.

[8] Griffiths, P., Grossman, D., and Bryant, R. L. (2003). *Exterior Differential Systems and Euler-Lagrange Partial Differential Equations*. University of Chicago Press.

[9] Harris, J. M., Hirst, J. L., and Mossinghoff, M. J. (2000). *Combinatorics and Graph Theory*. Springer.

[10] Kolmogorov, V. and Zabih, R. (2004). What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159.

For the above citations:
1) `http://www.govisland.com/html/visit/faq.shtml`
2) `http://www.draganfly.com/uav-helicopter/draganflyer-x4/specifications/cameras.php`
3) `http://www.draganfly.com/uav-helicopter/draganflyer-x8/specifications/`