For office use only	Team Control Number	For office use only
T1 _____	**1920682**	F1 _____
T2 _____		F2 _____
T3 _____	Problem Chosen	F3 _____
T4 _____	**B**	F4 _____

2019
MCM/ICM
Summary Sheet

DroneGo: Humanitarian in the Sky to Offer Quick and Adequate Response to Disaster-Stricken Regions

Summary

Despite the development of modern science and technology, natural disasters still haunt on people's mind. In 2017, Hurricane Maria, the worst hurricane on record, hit Puerto Rico, causing severe damage and large casualties. Now that natural disasters cannot be killed in the cradle, our central task should be focused on researching new methods to provide timely and adequate response during or after disaster occurs.

To put it into practice, our paper designs a DroneGo disaster response system through which anticipated medical supply demands will be satisfied, if similar disaster scenario takes place like Maria. Within the system, our paper states three basic models to address different subproblems. In the meantime, we implement our models with *a good balance* between two missions-delivery and video reconnaissance. After simulation, we obtain the optimized results of our models and recommend it to HELP, Inc. for future use.

Container Internal Placement Model solves the issue of packing configurations in two aspects. The problem is abstracted into a 3D packaging problem, no matter what object to be packed. We devise an algorithm called **3D Packaging Judgement**, in which **Metropolis-Hasting algorithm** is used to provide specific configurations of three containers respectively and alternative solutions for drone cargo bay packing configurations.

Then, we select three locations for three containers based on two main important factors, demand distribution and battery consumption. Demand distribution divide disaster-stricken area into three separated regions,while battery consumption which indicates the max flight radius locates specific location in the three regions. In location selection part, for every drone fleet, which hospital to serve is also determined.

Route Selection Model decides the route through which each drone goes to the hospital and returns, and offers final bay packing configurations. We take total flight distance and max road coverage in to account. The three regions are divided into two kinds, delivery-oriented and reconnaissance-oriented. **Bézier Curve** is applied to generate return routes, while **Simulated Annealing** is employed for routes simulation. After routes are determined, we generate final recommendations for drone payload packing configurations.

The implementation of our model is integrated into each of the three model, we give results and conclusions after explaining our model. While implementing our model, we also survey the literature, google scholar, and other sources, besides the given dataset.

In the end, we make sensitivity analysis and discuss strengths and weaknesses.We also write a memo to the CEO of HELP, Inc., in which we summarize our main outcomes and put forward recommendations.

Keywords: 3D Packaging Judgement, Metropolis-Hastings algorithm, Analytic Hierarchy Process, Bézier Curve, Simulated Annealing

DroneGo: Humanitarian in the Sky to Offer Quick and Adequate Response to Disaster-Stricken Regions

January 31, 2019

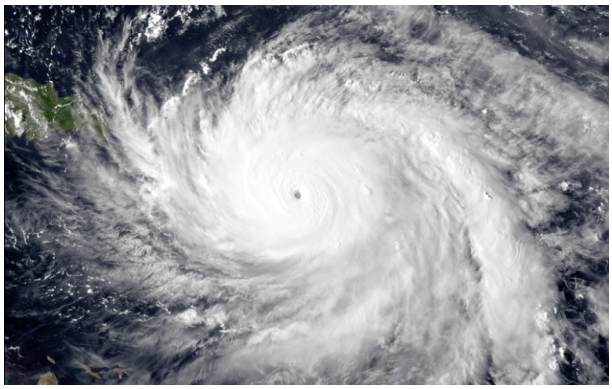
Contents

1 Introduction

1.1 Problem Background

Catastrophic disasters never stops disturbing people's peaceful life, causing many fatalities and huge economic losses. An important method to minimize the effects of terrible disasters is to offer quick and adequate response during or after disasters occur.

In 2017, Hurricane Maria, the worst hurricane on record to strike the United territory of Puerto Rico, caused severe damages and large casualties, the image of which is presented in Figure 1(a). The electrical power and cell service outages lasted for months, and many highways and roads across the island were damaged severely. Demands for medical supplies and other medical facilities increased drastically. However, it was barely possible for ground vehicles to access the disaster-stricken region, since roadways were heavily damaged by widespread flood, as shown in Figure 1(b).



(a) Hurricane Maria near peak intensity, moving north towards Puerto Rico, on September 19, 2017.

(b) A road is littered with structural debris, damaged vegetation and downed power poles and lines, due to floodings cause by Hurricane Maria

Figure 1: Severe Damages Caused by Hurricane Maria[1]

Drones are able to perform humanitarian tasks fully and timely, thanks to its means of shipping and fast speed. Its image is shown in Figure 2. Therefore, a Non-governmental organization(NGO)-HELP, Inc. -aims to promote its disaster response capabilities by applying rotor wing drones to deliver pre-packaged medical supplies and provide high-resolution aerial video reconnaissance, simultaneously or separately. This augments local medical assistance organizations to a great extent.

Figure 2: Drones for Disaster Relief Purpose

In order to save limited resources and equipment, drone fleets and three locations of containers are restricted, so the response system has to go to great lengths to accomplish the two tasks, *adequate medical supply delivery* and *video reconnaissance*.

1.2 Our work

We take three steps to address the problem. First, we need to solve the packing issue in two aspects, while taking relevant constraints into account. Then, we select three locations for containers, and provides route planning for drone fleet to give consideration to both missions mentioned above. In this paper, we keep a good balance between the two missions, and offer our approach to solve the problem.

In Section 2, we state three basic assumptions. Section 3 contains the nomenclature used in our model statement. Section 4 offers sufficient details and discussions about our model, and provides results and conclusions of implementing our model at the same time. We further study our model in Section 5 by analyzing it in detail. At last, we make some conclusions in Section 6, and submits a memo to the CEO of HELP, Inc. in Section 7.

2 Assumptions

Our model makes the following assumptions:

1. HELP, Inc. is able to position each container to the location our model selects, and it is safe and sound to place the container there. This is not influenced by any other factor, such as: flooding or damaged roads.
2. The take-off and landing process of each drone is ignored. This means both process does not cost power consumption. Additionally, we simplify the speed of the drone to a constant value, which is given in Attachment 2. Thus, total flight time indicates how far a drone can go.
3. There is often a window phase for the drone fleet to perform tasks. It is common practice that bad weather conditions do not last for a long period, and each drone fleet can meet several days' demands of the hospital for every flight. Therefore, it is reasonable to assume like this.

3 Nomenclature

In this paper, we use the nomenclature in Table 1 to describe our model. Other symbols that are used only once or twice will be described later in the context.

Table 1: Nomenclature

Symbol	Definition
C_i	The *i*th container, *i*=1,2,3
$D_{i,j}$	The *j*th drone of the *i*th container,*j* follows the order of A to H
$B_{i,j}$	The cargo bay of $D_{i,j}$
$P_{i,j,k}$	The *k*th medical package of $B_{i,j}$
L, W, H	The length, width, and height of a standard ISO container
l_m, w_m, h_m	The length, width, and height of the *m*th object to be placed into the container
$M(x, y)$	The importance distribution of medical demand for each map point (x, y)
$n_{s,t}$	The quantity demand of the *t*th medical package of the *s*th hospital
R	The set of point on roads
$I_R(x, y)$	The road indicator function
S_i	The reachable road area of the ith container
$N_{D,i}$	The number of deployed drones in the ith container

4 Statement of our Model

In this section, we will discuss all details about our model. The model take many factors into consideration, based on the given dataset and online researching of other important things. To begin with, we first put forward our model of packing configurations of three containers and drone payload. Afterwards, three locations are selected to position three containers respectively. Finally, we plan the detailed routes of our drone fleets. The model keeps a great balance between delivery and video reconnaissance[2]. The complete diagram is vividly shown in Figure 3.

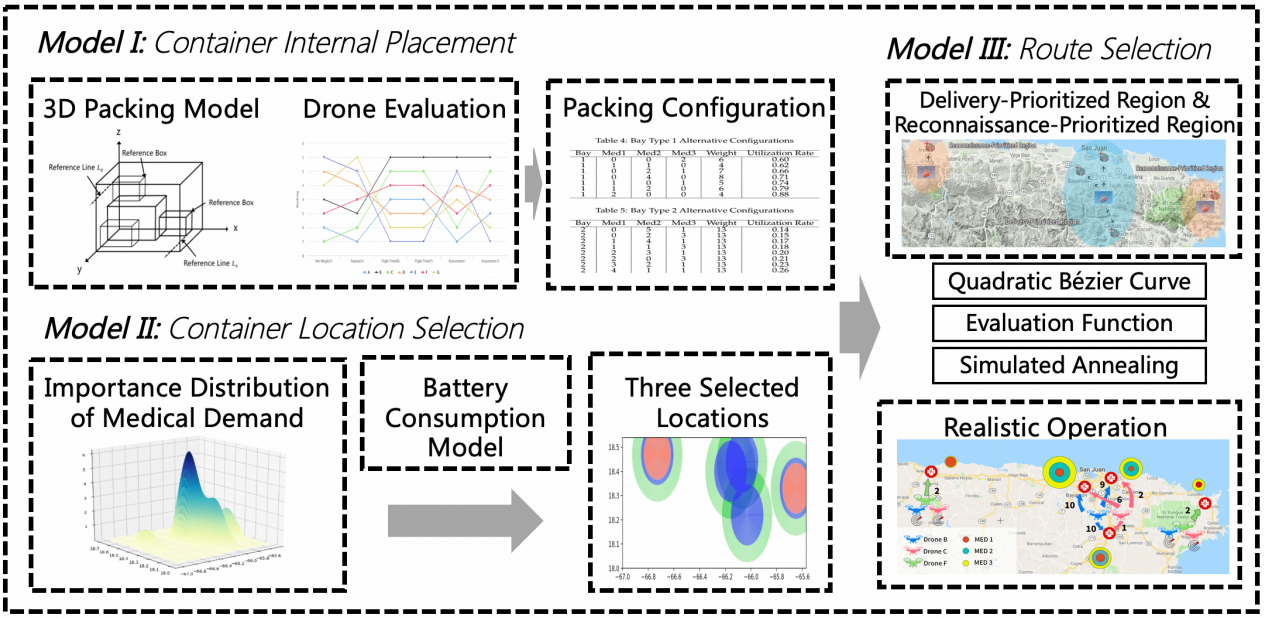


Figure 3: The Entire Diagram of Our Three Models

4.1 Container Internal Placement Model

Container Internal Placement Model gives solutions to *which and how* drones and their cargo bays inside the container are configured, as well as *which and how* medical packages inside the cargo bay are configured. The two configurations are quite similar, so they *share the same Algorithm 1* in our model.

We first solve the problem of *how*, after which we come up with drone evaluation to address the problem of *which*.

Our model gives *determined* configurations of drones and their bays for the three shipping containers, and provides *alternative* configurations of medical packages that make good use of drone cargo bay. However, medical supply demands from the six local hospitals should be satisfied. Therefore, we will select the most proper cargo bay configuration in our implement part in Section 4.3.4.

4.1.1 The Structure of Container and Cargo Bay

Each container is composed of different combinations of a drone and its cargo bay, or simply a drone itself, while every cargo bay consists of different combinations of three medical packages. The relationship among them is clearly depicted in the tree diagram below in Figure 4.

Apart from given size(length, width, and height) of a standard container and two kinds of cargo bays that must be complied with, it is clear that the type of cargo bay relies on its drone, and the number of cargo bays *less than* that of drones. This provides restrictions for Section 4.1.4.

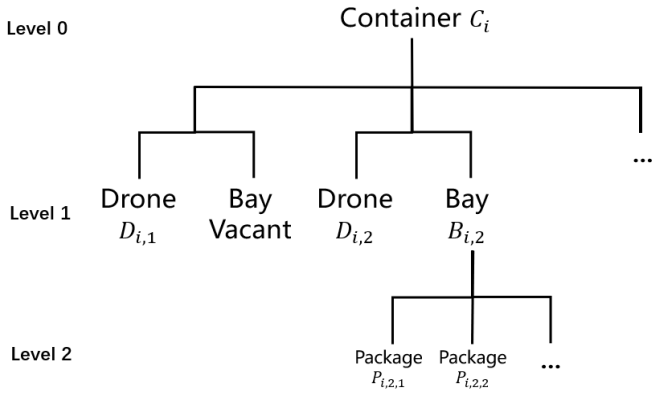


Figure 4: The Relationship Among Container, Drone and Its Bay, and Medical Package

4.1.2 3D Packaging Judgement

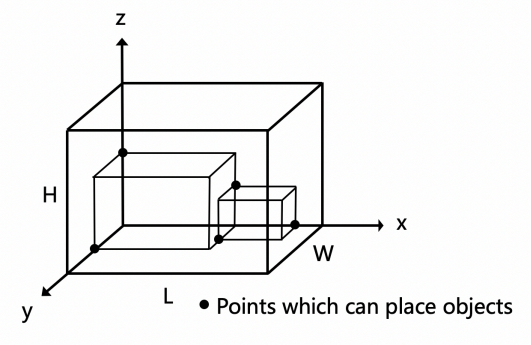
Inspiration When building up a wall, people generally place *a brick for reference* in the first place. Then, people set the height of *the brick for reference* as a benchmark, and stipulate the height of each object placed-inside less than the benchmark. The benchmark can be heightened, only when no object can be placed inside.

Operation During our three-dimension packing process, similar "reference brick" is introduced in x-axis and z-axis directions to guide *the container-loading and drone cargo bay-loading course*. For simplicity, both drones and cargo bays to be put inside the container are called objects. We use points where another object can be placed or *available points* to describe and find where the next object can be placed inside. For each object put into the container, it occupies a single available point and generates three similar points. The operation is derived from **the Greedy Algorithm** and **the Heuristic Algorithm**, but what makes the operation distinctive is that it does not need to fit in with a certain structure. This makes our loading process quite flexible.

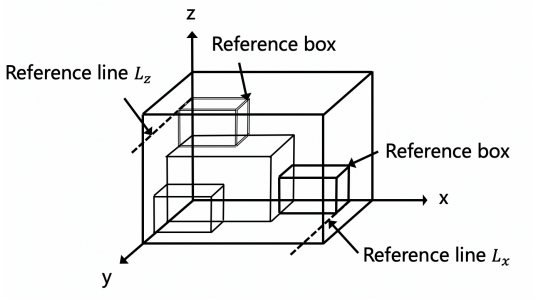
As mentioned above, we are going to put either a drone and a drone's cargo bay into a box. Suppose we have boxes series $\{b_1, b_2, \ldots, b_n\}$. At the beginning, the 1st box b_1 can be put at $(0, 0, 0)$. Afterwards, b_2 has three available points $(l_1, 0, 0), (0, w_1, 0), (0, 0, h_1)$. Suppose the 2nd box chooses $(l_1, 0, 0)$, we delete the available point $(l_1, 0, 0)$ and add $(l_1 + l_2, 0, 0), (l_1, w_2, 0), (l_1, 0, h_2)$ to the set of available points.

As for reference line, we select the reference line L_z and the reference line L_x, both parallel to axis y, as shown in Figure 5(b). We sort the objects attached to available points by y from small to large. If y is equal, sort by x from small to large. If x, y are equal, sort by z. Suppose we are trying to put an object to available point. Before placing, the container C_i cannot overlap in any part. Besides, the object(τ_m) should satisfy the equation: $z + h_m \leq L_z, x + l_m \leq L_x$. When the first feasible available point is found, we attach the object to it. If there is not any feasible points, we conduct the following steps: (1) If $L_x < L$, then we raise the reference line L_x. (2) Otherwise, we raise the reference line L_z. After raising both reference lines, if the object still can't be put in, we no longer consider the object.

After researching, we devise an algorithm called **3D Packaging Judgement**, shown in Algorithm 1 [3]. The time complexity of the algorithm is $O(n^3)$. In order to simplify the problem, we assume that the object(drone or its cargo bay) cannot rotate horizontally. In rotation cases concerning the axis z, rotation should not be taken into account at all, since medical supplies consist of some fragile and delicate equipment. Once damaged, its utility reduces a lot, eventually causing unnecessary losses that outweigh space saved.



(a) Available Points



(b) Horizontal and Vertical Line

Figure 5: Two Major Steps of 3D Packaging Judgment

Algorithm 1 3D Packaging Judgement

Input: $B = \{b_1, b_2, \ldots, b_n\}$
Output: True or False
$I \leftarrow \{(0,0,0)\}, L_z \leftarrow 0, L_x \leftarrow 0, count \leftarrow 0$
for *i from 1 to n* **do**
  $flag \leftarrow$ False
  foreach $(x, y, z) \in I$ **do**
    if b_i *can be put into position* (x, y, z) *and* $x + l_i \leq L_x, z + h_i \leq L_z$ **then**
      $flag \leftarrow$ True, break
    end
    if $flag$ = *False* **then**
      if $L_x = 0$ *or* $L_x = L$ **then**
        if b_i *can be put into position* $(0, 0, L_z)$ **then**
          $x \leftarrow 0, y \leftarrow 0, z \leftarrow L_z, flag \leftarrow$True, $L_z \leftarrow L_z + h_i, L_x \leftarrow l_i$
        end
        else if $L_z < H$ **then**
          $L_z \leftarrow H, L_x \leftarrow L, i \leftarrow i - 1$
        end
      end
      else
        foreach $(x, y, z) \in I$ *which* $x = L_x, y = 0$ **do**
          if b_i *can be put into position* $(0, 0, L_z)$ *and* $z + h_i \leq L_z$ **then**
            $flag \leftarrow$ True, $L_x \leftarrow L_x + l_i$, break
          end
        end
        if $flag =$ *False* **then**
          $L_x \leftarrow L, i \leftarrow i - 1$
        end
      end
    end
    if $flag =$ *True* **then**
      Put b_i into position (x, y, z), $I = I/\{(x, y, z)\}$, move b_i along the X, Y, Z axis coordinate reduction direction and get (x', y', z'), $I = I \cup \{(x'+l_i, y', z'), (x', y'+w_i, z'), (x', y', z'+h_i)\}$
      $count \leftarrow count + 1$
    end
  end
end
return $count = n$

Although the rotation-included program is not realizable during our modeling time, we do design the whole process of it, which can be clearly seen in Appendix B.5. Provided that enough time is given, the results of the program can be surely obtained.

4.1.3 Drone Evaluation System

Since the two missions - *medical supply delivery* and *video reconnaissance* require different capabilities of drones, our drone evaluation system provides two assessment for each task accordingly. When it comes to the delivery mission, drones' max flight distance and shipping ability are both taken into account, while we only consider max flight distance for reconnaissance mission.

Assessment I is relevant to four determining factors, net weight per volume, max payload capability per volume, flight time with no cargo, and flight time with full cargo. Flight time with full cargo is related to battery conditions, which our model will give a detailed discussion in Section 4.2.2. Therefore, the Assessment I equation is can be stated as follows in Equation (1).

$$A_1 = c_{11} \times \frac{N_i}{V_i} + c_{12} \times \frac{L_i}{V_i'} + c_{12} \times \frac{F_{1,i}}{\max\left(F_{1,i}\right)} + c_{14} \times \frac{F_{2,i}}{\max\left(F_{2,i}\right)} \tag{1}$$

Similarly, Assessment II is relevant to flight time with no cargo, and flight time with full cargo. It is stated in Equation (2).

$$A_2 = c_{21} \times \frac{F_{1,i}}{\max\left(F_{1,i}\right)} + c_{22} \times \frac{F_{2,i}}{\max\left(F_{2,i}\right)} \tag{2}$$

where:

- N_i is the net weight of the *i*th drone;
- V_i is the volume of the *i*th drone;
- L_i is the max payload capacity of the *i*th drone;
- V_i' is the volume of the *i*th drone and its cargo bay;
- $F_{1,i}$ is flight time with no cargo of the *i*th drone;
- $F_{2,i}$ is flight time with full cargo of the *i*th drone;
- $c_{m,n}$ is the weight of each term.

The **Analytic Hierarchy Process(AHP)** is a structured technique for organizing and analyzing complex decisions, based on mathematics and psychology. We use AHP to determine $c_{1,k}$ coefficients, shown in Table 2. We apply importance proportion to determine c_{21} and c_{22}. One thing is clear that flight time matters a lot, as it decides whether a local hospital can be supported.

Table 2: AHP production

Coefficient	Weight
c_{11}	0.2292
c_{12}	0.2278
c_{13}	0.1119
c_{14}	0.4221

Afterwards, we use the two formula to evaluate different types of drones, and normalize them by dividing each by the largest value, which is clearly indicated in Figure 6.

Conclusions For *delivery mission*, we select Drone B and Drone C; for *reconnaissance* mission, we select Drone B, Drone C, and either Drone F or Drone G.

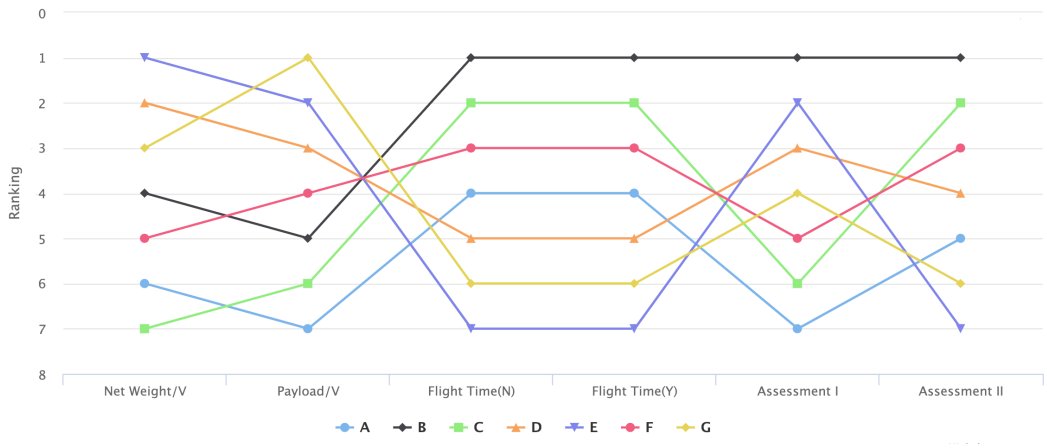


Figure 6: Assessment I and II Chart of Different Drones

4.1.4 Container Packing Configuration

If we have one group of a drone and its cargo bay, or simply a drone, we can make judgment whether it is feasible to be placed inside the container through Section 4.1.2. Clearly, it is unwise to enumerate all possible conditions. Thus, we choose to apply **Metropolis-Hastings algorithm(MHA)** [4] to obtain the optimized outcomes.

STEP 1 Generate a random selection sequence $\theta_k = \{N_{k,1}, N_{k,2}, \ldots, N_{k,n}\}$, where $N_{k,j}$ is the number of jth drone.

STEP 2 Generate θ. Our generation rule is that there is a chance of p to randomly generate new θ, and a probability of $(1-p) : \theta \leftarrow \theta_k + \varepsilon, \varepsilon \sim N(0, \sigma)$. If θ is not feasible, as verified by Section 4.1.2, we regenerate it.

STEP 3 We have a chance of α to accept the newly-generated θ as θ_{k+1}. Here, $\alpha = \min(1, SU(\theta)/SU(\theta_k))$, where $SU(\theta)$ is the space utilization rate of θ.

Therefore, for $N(= N_{k,1}+N_{k,2}+\cdots+N_{k,n})$, after N is determined, we can get reasonable outcomes.

Drone Fleets' Results For *delivery-oriented mission*, we are prefer to configure twenty-nine Drone Bs all with bays and nine Drone Cs all with their bays, too. Table clearly presents our outcomes; for *reconnaissance-oriented* mission, we are tend to configure twenty Drone Bs with no bay, nine Drone Cs without any bay, and two Drone Fs all with bays

Table 3: Two Kinds of Container Packing Configuration or Drone Fleets

Drone Fleet I			Drone Fleet II		
Drone	Bay	Quantity	Drone	Bay	Quantity
B	$\surd$	29	B	$\times$	20
C	$\surd$	9	C	$\times$	9
			F	$\surd$	2

4.1.5 Drone Payload Packing Configuration

Similar to the methods used in Section 4.1.4, we get different configurations of medical packages in a drone cargo bay. Our configuration offers different combinations of medical packages, and the combination can be chosen according to local hospital demands.

Results Our model produces seven alternative configurations for cargo bay type 1, and twenty-eight potential configurations for cargo bay type 2. The average utilization rate for bay type 1 is 71.4%, while it is 21% for bay type 2. The utilization rate is such low for bay type, because bay type 2 is much larger than bay type 1, and there exists max payload capability restrictions. The entire options are listed in the Appendix A.

After discussion in 4.2, we will screen through these data and give out recommendations. At this step, we only provide alternative solutions for max utilization rate. After taking hospitals' demands and locations into consideration, alternative solutions will be narrowed down to recommendations.

4.2 Container Location Selection Model

In this section, our model identifies the best locations on Puerto Rico to position three cargo containers. First, we give the Importance Distribution of Medical Demand to indicate three isolated regions. Then, based on the speed and max flight time full cargo of each drone, we obtain a max flight radius for each drone. After that, we finally get three container locations.

4.2.1 Importance Distribution of Medical Demand

In this part, we give the Importance Distribution of Medical Demand to divide Puerto Rico into several demand-related regions. Relying on the location(longitude and latitude) of five hospitals and their varied demands for three medical packages, as presented in Figure 7, we generate the formula of $M(x, y)$. Only land areas are considered, and we make map points off land zero, since it is impossible to locate containers on sea. From Figure 8, we can clearly see three peaks, and this divides five hospital into three separated parts. In each of three part, there is a container location, in which its drone fleet *only serves* hospitals in this separated part.

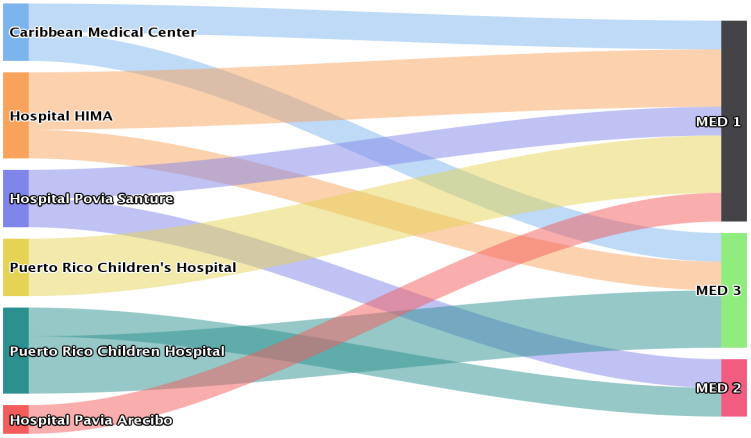


Figure 7: Five Hospital Different Demands for Different Medical Packages

We use latitude($H_{s,x}$) and longitude ($H_{s,y}$) of the sth hospital location, and latitude(x) and longitude(y) of each map point to calculate the straight-line distance between them, presented in Equation (3).

$$d_s = \left(\frac{\pi R_0}{180}\right) \times \arccos[\sin H_{s,x} \sin x \cos\left(H_{s,y} - y\right) + \cos H_{s,x} \cos x] \tag{3}$$

where R_0 is the radium of earth.

$M(x, y)$ relies on the distance d_s.

$$M(x,y) = \sum_{s=1}^{5} \sum_{t=1}^{3} n_{s,t} e^{-\frac{d_s^2}{2\sigma^2}} \tag{4}$$

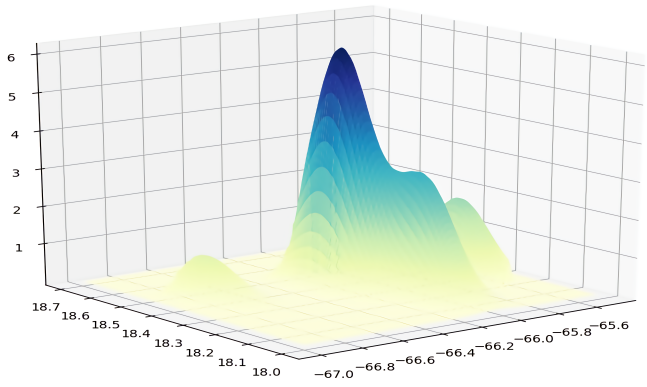


Figure 8: Importance Distribution of Medical Demand

4.2.2 Battery Consumption

Total flight time of a single drone differs, as *the weight of cargo* should be taken into account. After investigating kinds of droness for similar purposes and researching relevant factors that influence flight time, we generate an equation of drones's flight time $(t_i, i \in \{A, B, C, D, E, F, G, H\})$ in Equation (5) [5].

$$T_i = \frac{c_i \cdot d_i \cdot v_i}{a_i \cdot p_i} \tag{5}$$

where:

- T_i stands for the total flight time of each drone;
- c_i is the battery capacity of each drone;
- d_i is the battery discharge of each drone that is allowed for during the flight, which is typically a default value ;
- v_i is the voltage of each drone;
- a_i is the all-up weight of each drone;
- p_i is the power required to lift one kilogram of equipment.

Based on the given *Flight Time No Cargo* data in $Attachment\,2$, and rated parameters of batteries, a_i or *net weight* of each drone can be obtained. We use the parameter m_i to denote net weight of each drone. As soon as the drone is loaded with packaged cargo, a_i equals to the *sum* of m_i and payload carried. In this way, Equation 5 turns to Equation 6 as follows:

$$T_i' = \frac{c_i \cdot d_i \cdot v_i \cdot T_i \cdot p_i}{c_i \cdot d_i \cdot v_i + m_i \cdot T_i \cdot p_i} \tag{6}$$

In Route Selection Model, each drone only has two status, namely *cargo occupied* or *cargo vacant*. Therefore, there are two kinds of total flight time, which represents max delivery time possible(T_{i1}) and max return time possible (T_{i2}). T_{i1} and T_{i2} can be generated from Equation (5) and Equation (6). However, the outright battery capacity is a limited number, so there exists a restricting equation.

$$\frac{t_{i1}}{T_{i1}} + \frac{t_{i2}}{T_{i2}} = 1 \tag{7}$$

where:

- t_{i1} is the actual time spent for Delivery;
- t_{i2} is the actual time spent for Return.

In Equation (7), there are only two unknown numbers, t_{i1} and t_{i2}. After adding another equation of distance ($\frac{t_{i1}}{t_{i2}}=\frac{d_{i1}}{d_{i2}}$), we can calculate the values of them.

4.2.3 Three Selected Locations

Depending on demand distribution and battery consumption, we plot Figure 9. In the picture, the orange circle represents the flight radius of Drone F, blue for Drone C, and green for Drone B. The radius of each circle is determined by Equation (7), where t_{i1} equals to t_{i2}, and T_{i1} and T_{i2} are already known. The radius gives the max coverage of any circle. As is seen from the picture, there are two different sets of circles, namely one on the eastern and the western edge and the other in the middle. Accordingly, we provide two means to finalize our selection of three container locations.

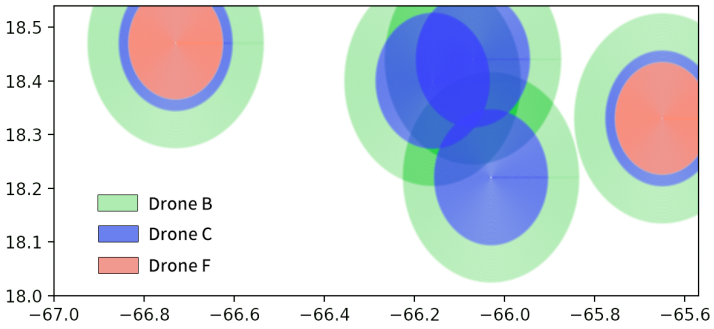


Figure 9: Three Demand-Related Regions

Middle Region For the region in the middle we first focus on the overlapping part of three smaller circles(or blue circles). Then, we calculate the circumferential circle center of the overlapping part, and determine it as the container location for the three hospitals in the middle. This is the place where any type of the drone deployed can reach any of the hospital in this region. Since this place have high demand for medical packages, as shown in Figure 8, we will call this place *delivery-prioritized region* in the following part.

Side Regions For both regions on the east and west, the demand for medical packages is low, thus offering us sufficient space for video reconnaissance. So in these two regions, our top priority is to maximize the road coverage. We denote the set of road points as R. Then we define the indicator function $I_R(x, y)$ [6]:

$$I_R = \begin{cases} 1, & (x, y) \in R \\ 0, & (x, y) \notin R \end{cases} \tag{8}$$

In these two regions, there is only one hospital, respectively. We draw a circle C_H centered at the hospital with radius r_m. r_m should be the minimum reachable radius of all the drones deployed in this region. In our model, this is the radius of Drone F. We choose a point $P_C \in C_H$ and draw another circle C_D centered at P_C with radius r_s. We estimate road area within the ith region by

$$S_i = \iint_{C_D} I_R(x, y)\mathrm{d}x\mathrm{d}y \tag{9}$$

Figure 10: Find Optimal Location for Containers on Side Regions

The method is shown in Figure Here we can use **Simulated Annealing(SA)** to find C_D with the largest road area. Simulated Annealing somehow resembles Metropolis-Hastings Algorithm in their "accept by probability" strategy. The difference is that SA aims to find the optimal solution while MHA aims to generate samples from a function.

Table 4 and Figure 11 showcase specific locations for three containers respectively.

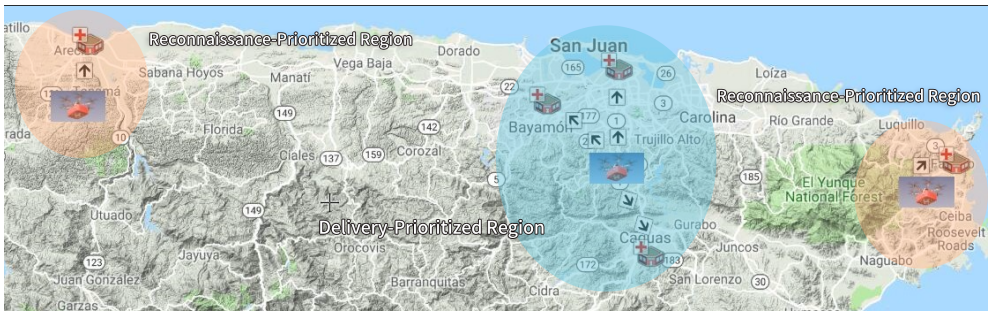


Figure 11: Three Container Locations

Container Location	Longitude	Latitude
1	-65.6848	18.2934
2	-66.0710	18.3200
3	-66.7421	18.3932

Table 4: Three Container Locations

4.3 Route Selection Model

As is mentioned before, the three regions are divided into two kinds, the eastern and western two regions as well as the middle region. Since the first two regions only has one hospital to serve, their mission is mainly reconnaissance-prioritized. Since there are three hospitals to serve in the middle region, the drone' mission there is mainly delivery-prioritized.

4.3.1 Delivery-Prioritized Region

Now we shift our attention to the delivery-prioritized region, which refers to the part around San Juan. We call this place *delivery-prioritized* because we have to serve three of the five hospitals in this region. In this place, the efficiency of transporting medical packages to hospitals is the top priority,

while the video reconnaissance work should not be ignored. With this situation in consideration, we first propose a strategy for each drone in the container to load medical packages into its bay and transport to a certain hospital. Then the drone can decide its path of returning back, which maximizes road coverage.

When a drone is about to take off, the content of the bay tied to a drone taking off must be decided. Here we can randomly choose a packaging strategy that satisfies the payload capabilities of the drone. The packaging strategy candidates are already provided in Container Internal Placement Model. The best combination of packaging strategies for all drones can be later figured out by a heuristic algorithm.

As demonstrated in Destination Selection Model, the location of the container in delivery-prioritized region is chosen in such a way that the drone of both type deployed, i.e. Drone B and Drone C, can reach any of the hospital in delivery-prioritized region and return safely. After the bay is packaged, a drone randomly choose a hospital in this region. The best distribution of the drone flow can also be figured out by a heuristic algorithm.

After delivering the medical packages, the drone has to return to container and wait for charging. Since the drone has completed its major task, the efficiency of video reconnaissance should be considered. If the drone returns in straight line, the margin of the battery life may not be fulled exploited to assess road conditions. However, as the power is not so efficient, the returning route cannot be too twisted, in which case the drone may have difficulty reaching container.

Here we use **Quadratic Bézier Curve** [7] to depict this returning route. Bézier Curve is widely used in many fields, such as computer graphics and robotics. The nth order of Bézier Curve is determined by $n+1$ points $P_0, P_1, ..., P_n$. A Quadratic Bézier Curve can be expressed mathematically as:

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2, 0 \le t \le 1 \tag{10}$$

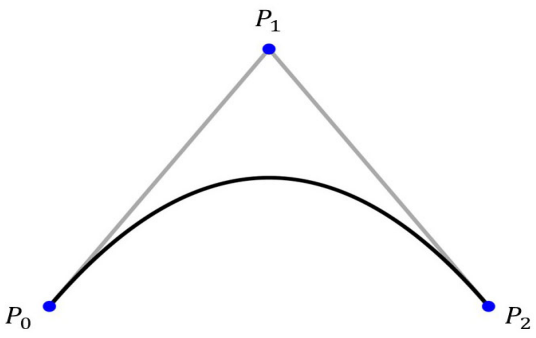


Figure 12: Plot of Quadratic Bézier Curve

A plot of Quadratic Bézier Curve is shown in Figure 12. Higher orders of Bézier Curve are also available, but need more control points and may look more twisted. That's why we choose the quadratic one.

The length of a Bézier Curve can be expressed by the following integral:

$$\begin{aligned} |\mathbf{B}| &= \int_B \mathrm{d}l = \int_0^1 \sqrt{[P_x'(t)]^2 + [P_y'(t)]^2}\mathrm{d}t \\ &= \int_0^1 2\sqrt{[-(1-t)P_{0x} + (1-2t)P_{1x} + tP_{2x}]^2 + [-(1-t)P_{0y} + (1-2t)P_{1y} + tP_{2y}]^2}\mathrm{d}t \end{aligned} \tag{11}$$

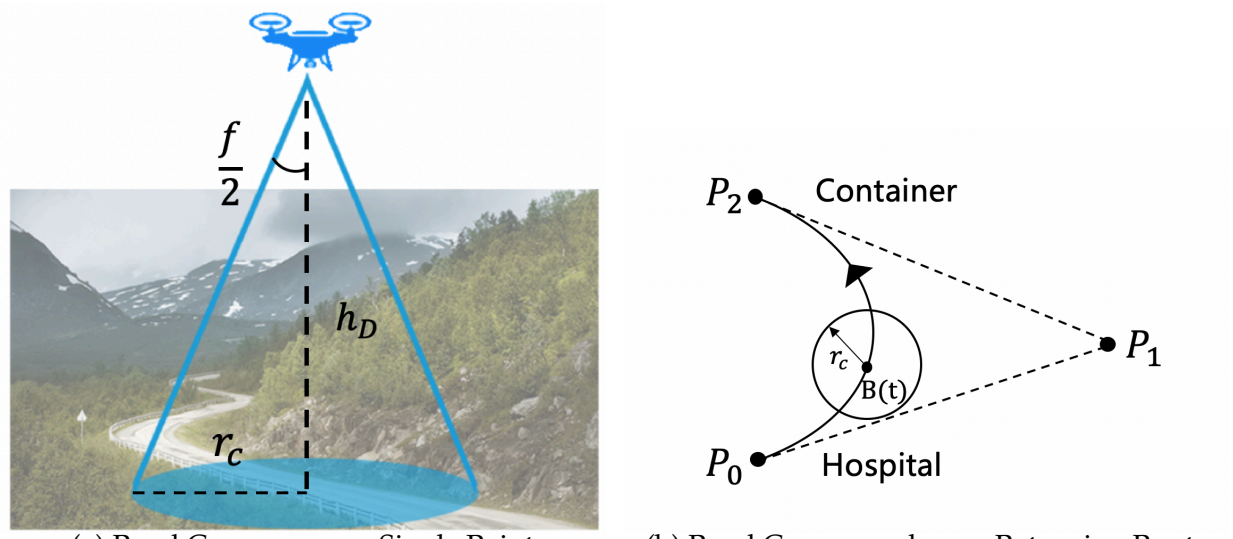
Note that this equation is rather difficult to get its analytic solution. In practice, we subdivide this curve into several line segments and compute the sum of their lengths. With Equation (11), we can get the length of a drone's returning route.

A drone has to assess road conditions in Puerto Rico, so a criterion must be defined to evaluate the effectiveness of video capture. First we consider a certain point on the route. Suppose a drone with field of view [8] f is at height h_D, then the ground area covered by the camera can be approximated by a circle of radius $r_c = h_D \cdot f/2$. With this in mind, we can define road coverage of a route $\mathbf{B}$ as:

$$CV(\mathbf{B}) = \frac{w_r}{\pi r_c^2} \int_B \mathrm{d}l \iint_{C_t} I_R(x, y) \mathrm{d}x\mathrm{d}y \tag{12}$$

where:

- w_r is the average width of road
- C_t is the circle area centered at $\mathbf{B}$(t) with radius r_f
- $I_R(x, y)$ is the indicator function which indicates whether the point (x, y) is on roads.



(a) Road Coverage on a Single Point (b) Road Coverage along a Returning Route

Figure 13: Determination of Road Coverage

After defining the coverage $CV(\mathbf{B})$, we have to find a way that can maximize road coverage on the returning route. Because of the difficulty of integrating over the whole curve, we resort to a local search method. We connect the hospital and container with a straight line segment l_s. Then we draw a line segment l_p of length $2|l_s|$ which is perpendicular to l_s and its midpoint coincides with the one of l_s. We divide l_p into several segments, put Bézier Curve control point P_1 at each endpoint of these segments and try to find the position which has the largest $CV(\mathbf{B})$. The method is demonstrated in Figure 14.

4.3.2 Reconnaissance-Prioritized Region

In this part we discuss the other two regions: the one on the north-east coast and the one on the north-west cost. The two share in common that there is only one hospital in each of the region and the demand for medical packages are relatively low. In this region, the delivery is not so important as it is in delivery-prioritized one. More drones should be deployed to assess the road conditions instead of sending goods. The deployment strategy could be quite different from the one discussed in delivery-prioritized region.

Since the demand for goods is low, a small number of drones sending a fixed set of medical packages can sufficiently meet the demand. Therefore, no randomness is involved in this region and we don't need any heuristic algorithm to find an optimal solution. Among all the drones in the

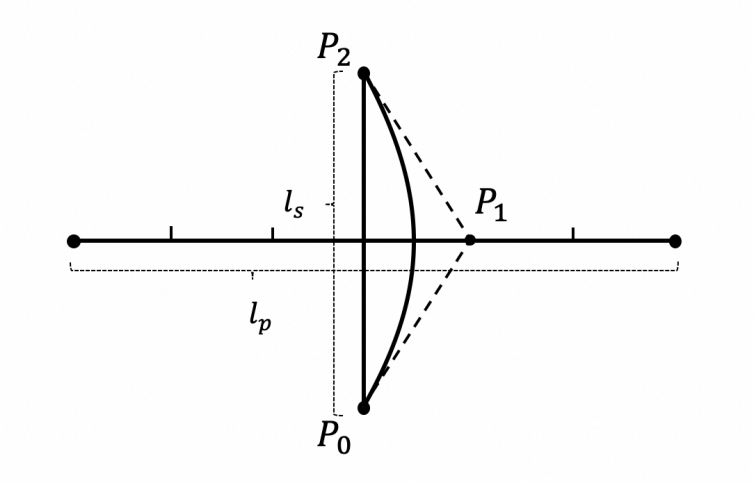


Figure 14: Local Search of Best-Covering Route

containers transported to these two locations, Drone F is only capable of delivering goods, so the two Drone F's should shoulder the responsibility of transporting medical packages. The bay packaging for Drone F's in these regions can specified explicitly, according to the demand of target hospitals. The other two types, Drone B and Drone C are assigned to assess nearby road conditions.

For video reconnaissance-oriented drones, the routes design should be different from the one discussed in previous part, in that it doesn't have to transport packages and should fully explore the roads nearby. The strategy is that when a drone takes off, it searches for the nearest road that has not been discovered and march to that point. It follows the roads and captures videos of them, while keeping track of the distance from the container and judging whether it should return, in case it may not be able to get back if battery life runs out. If the battery life could not support drone to cruise further, it returns to the container along the straight line route.

The situation gets complicated if we consider of the variable shapes of roads. In that way, the total road coverage of drones may be hard to expect. Here we firstly consider a simplified case, as shown in Figure 15 . Suppose there is a semi-infinite road. A container is away from the road by d. Drones are released from containers, one by one. The route of the first, second and third drone are marked on the figure. The furthest point on axis X on the road the ith drone can reach is denoted as x_i, and the length of road covered by the ith drone l_i. Therefore, we have $l_i = x_i - x_{i-1}$. Suppose the maximum length a drone can fly through is s, using Pythagorean theorem, we have:

$$\sqrt{d^2 + x_{i-1}^2} + l_i + \sqrt{d^2 + x_i^2} = s \tag{13}$$

According to Equation (13), we can get the recursion formula of series x_i:

$$\begin{cases} x_0 = 0 \\ x_i = \dfrac{(s + x_{i-1} - \sqrt{d^2 + x_{i-1}^2})^2 - d^2}{2(s + x_{i-1} - \sqrt{d^2 + x_{i-1}^2})} \end{cases} \tag{14}$$

We set s to be the average flight distance of drones in the fleet, and d to be average distance from the containers to the roads nearby. We compute the first several elements in the series and plot it on the graph. We find that this series can be fitted with an exponential function. See Figure 16 for the plotted graph. We find that the the more drones we put into use, the less coverage a single drone can produce. That's because when more drones are launched, the farther the drone has to fly to reach the starting location of video capture. Since the battery life of a drone is fixed, the distance that a drone can actually spend on road assessment decreases continuously.

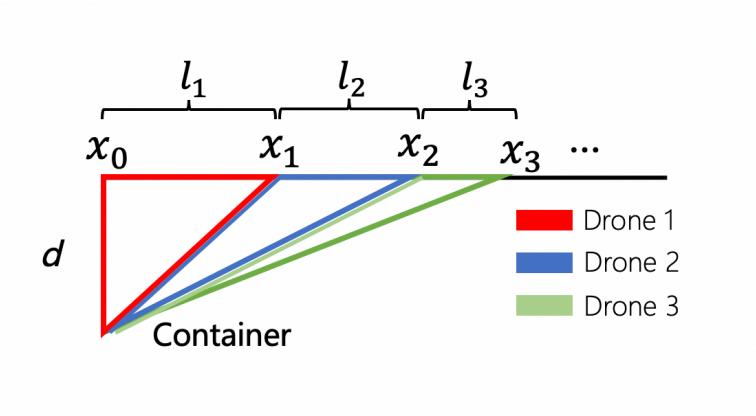


Figure 15: A Simplified Case of Drone Routes for Video Reconnaissance

The actual road condition may be far more complicated than the one discussed preciously, but we can infer that they have similar growing trend. So we can define the coverage function of the ith reconnaissance-prioritized region CV_i as the following:

$$CV_i = S_i(1 - e^{-\alpha N_{D,i}}) \tag{15}$$

where

- S_i is the road area that is reachable by drones with straight line routes in the ith reconnaissance-prioritized region.
- $N_{D,i}$ is the number of drones assigned to carry out video reconnaissance work

In our implementation, we set α to be 1.7293. The sensitivity analysis of this parameter is discussed in Section 5.

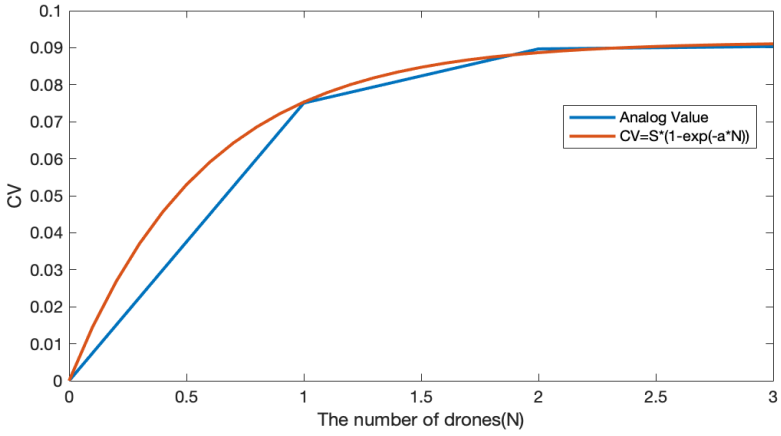


Figure 16: Plot of the Series and Fitting Function

4.3.3 Route Evaluation

We have already discussed the situation in delivery-prioritized and reconnaissance-prioritized regions. To eventually find the combination of bay packaging and drone dispatch strategy that maximizes the general efficiency of delivering response system to Puerto Rico, including both package delivery and video reconnaissance, we have to firstly define an evaluation function that assesses the comprehensive performance.

For the package delivery part, each hospital has its certain demand for medical packages, and each container has drones that are different in their payload. The problem may seem complicated if we want to take all of the supply and demand into consideration. Here we follow the Cask Principle, which claims that the performance of a system is determined by its weakest part. In the case of package delivery, this refers to the minimum number of days any hospital can sustain itself without additional packages. This seems reasonable in the case of package delivery, because whenever any of the hospital is in lack of the any type of the package, the company has to deliver packages again. We want a delivery that can support all the hospitals on the island to survive more days. Suppose package demand of the tth type package in the $s$the hospital is recorded in $n_{s,t}$, and the actual delivered package of the tth type to the sth hospital is $n'_{s,t}$, the sustaining days for the tth package in the sth hospital is:

$$m_{s,t} = \begin{cases} \left\lfloor \dfrac{n'_{s,t}}{n_{s,t}} \right\rfloor, & n_{s,t} \neq 0 \\ \infty, & n_{s,t} = 0 \end{cases} \tag{16}$$

Then the day interval between two deliveries m_d can be expressed as:

$$m_d = \min_{1 \leq i \leq 5, 1 \leq j \leq 3} \{n_{s,t}\} \tag{17}$$

Then we can define the delivery term E_d:

$$E_d = \frac{2}{1 + e^{-km_d}} - 1 \tag{18}$$

This function looks similar to Logistic function [9]. The difference is that this function maps natural numbers $\mathbb{N}$ to real numbers in $[0, 1)$. When m_d decreases to numbers around zero, the value of this function falls dramatically. This property helps avoid extremely low m_d and ensures the effectiveness of delivery.

As for the video reconnaissance evaluation, we just need to compute the ratio of the sum of the road area covered in all regions CV_i to the total road area S_{all}:

$$E_v = \frac{\sum_{i=1}^{3} CV_i}{S_{all}} \tag{19}$$

Then we define our final evaluation function E:

$$E = E_d + E_v = \frac{2}{1 + e^{-km_d}} - 1 + \frac{\sum_{i=1}^{3} CV_i}{S_{all}} \tag{20}$$

With this equation, **Simulated Annealing** can be used to find the optimal solution. Just before we run SA, the last thing that should be clarified is the randomness involved in the process, and the dimension of the problem. For the reconnaissance-prioritized region, the container packaging strategy and drone routes are already determined, so there is no randomness in this part. For the delivery-prioritized region, we use local search to determine returning routes, so there is also no randomness. However, we randomly choose the bay packaging strategy and the hospital to deliver for each drone. Therefore, the maximum dimension of this problem is twice the number of the drones deployed in the delivery-prioritized region. If we reuse samples in our implementation, the dimension may be reduced.

4.3.4 Realistic Operation

Bay Packing In Section 4.1.4, we generate two kinds of drone fleets for container configurations. After generating the optimal outcomes, we employ Drone Fleet I for two reconnaissance-prioritized

Table 5: Set of Packages for Drone Fleet II

Container	MED 1	MED 2	MED 3	Quantity
1	3	0	3	2
3	6	0	1	2

Table 6: Set of Packages for Drone Fleet I

Drone Fleet I Bay 1(Drone B)				Drone Fleet I Bay (Drone C)			
MED 1	MED 2	MED 3	Quantity	MED 1	MED 2	MED 3	Quantity
1	1	0	7	4	1	1	2
0	2	1	5	2	2	2	1
1	0	1	4	1	1	3	1
0	0	2	4	1	0	4	1
2	0	0	3	2	0	3	1
0	0	2	2	5	0	1	1
0	4	0	2	3	1	2	1
1	2	0	2	4	0	2	1
5(sum)	9(sum)	6(sum)	29(sum)	22(sum)	5(sum)	18(sum)	9(sum)

regions in the side, and Drone Fleet II for one delivery-prioritized region on both sides. We provide cargo bay configurations for each container in Table 5 and Table 6.

Afterwards, based on each drone fleet and its set of packages, we obtain satisfied demands for once flight for every hospital. Accordingly, we calculate the time interval it takes for the next flight to depart. Both are shown in Table 7.

Table 7: Satisfied Demands for Five Hospital and Satisfied Days

Hospital	MED 1	MED 2	MED 3	Satisfied Days
1	6	0	6	6
2	18	6	8	8
3	14	9	17	9
4	13	10	15	6.5
5	12	0	2	12

Schedule Since we need to ensure local hospitals' demands, and the three regions do not affect each other. Therefore, the schedule for the eastern and middle region is six days per flight, and twelve days each flight for the western region.

The whole blueprint of our operation is depicted in Figure 17.

Drone Flight Plan In the eastern and western two regions, Drone Bs and Drone Cs are completely used for video reconnaissance, they go out to reach their longest possible radius, while their definite routes are determined by Section 4.3.2 to achieve the largest road coverage. Drone F's are only used for video reconnaissance mission when they come back, again using formulas in Section 4.3.2 to achieve the biggest detection zone.

In the middle region, Drone Bs and Drone Cs only assess major highway conditions and roads when they return to the base. These drones choose a route to maximize road coverage, based on theory in Section 4.3.1.

We draw a map of the highways and roads that are assessed by our drone fleets, which account for 11.596% of the total length of highways and roads. In Figure 18, the orange roads indicate the roads already assessed by our drone fleets while the black roads are left unassessed.

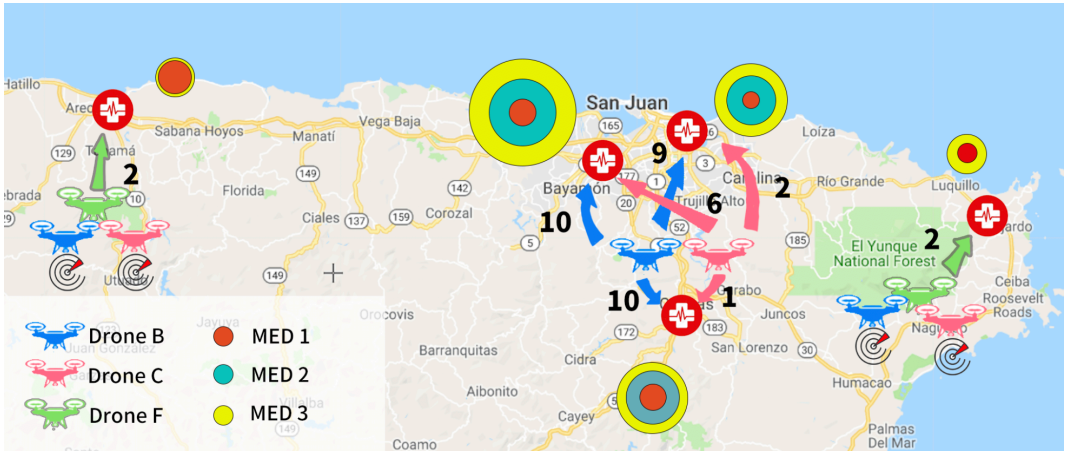


Figure 17: A Panarama of Realistic Operation

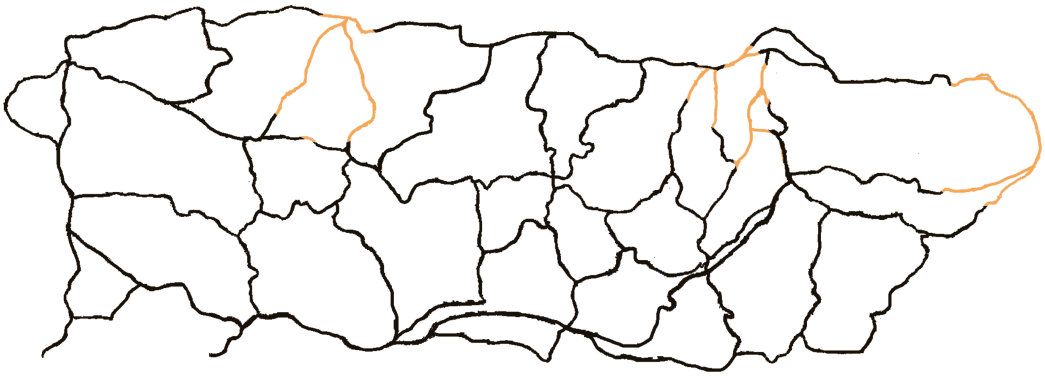


Figure 18: Road Covered VS The Entire Roads

5 Model Analysis

5.1 Sensitivity Analysis

Our model contains several parameters. We determine some of the parameters through AHP, digital image processing, and some of them by knowledge in the article or other methods. In the following subsections, we would like to produce a sensitivity analysis to show whether our model is sensitive to different values of parameters.

We will study the impact of two parameters in our model.

5.1.1 The Impact of α in Drone Coverage

The parameter α in Equation (15) is a result of function-fitting with given s and d in the simplified case of road coverage. As shown in Figure 19, if s is fixed, α changes a little bit with d. So with different S_i and $N_{D,i}$, we think changing α in the equation does not affect the result much.

5.1.2 The Impact of k in Delivery Evaluation Function

The parameter k in Equation (18) is decided by the implementor of the system. For different $k \in [0, 1)$, its value will influence Equation . When $k < 0.4$, it is quite hard to have high E_d value, even with rather high m_d. k can be understood as the tolerance of low m_d values.

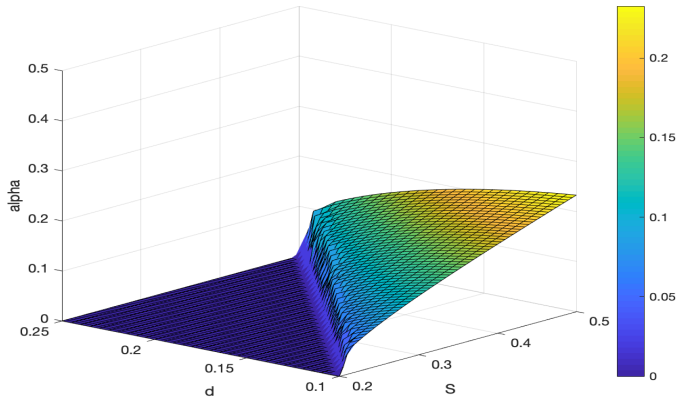


Figure 19: Different α about s and d

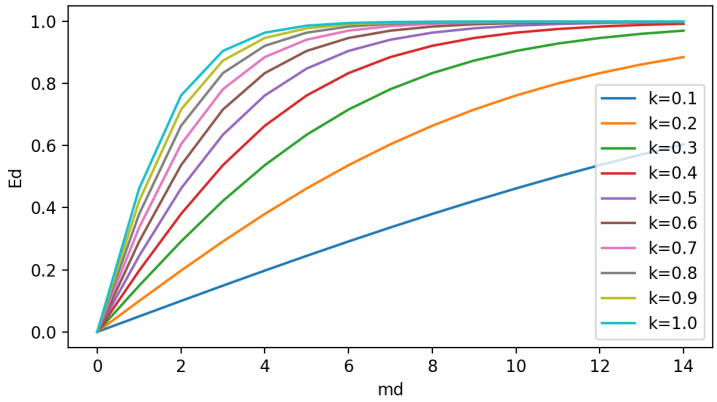


Figure 20: Different values about k

5.2 Strengths and Weaknesses

5.2.1 Strengths

1. Our model is logically rigorous. Our model completely puts forward specific drone fleets ,container and bay packing solutions to HELP, Inc.
2. We adopt interdisciplinary methods in our model. The knowledge and techniques of digital image processing are used in route selection part.
3. Our model is highly practial. All the theories in our model are concretely implemented and we offer comprehensive analysis and discussion of results.
4. We make great efforts in designing our memo, to make sure the CEO of HELP, Inc. can quickly understand our solutions and therefore share it with her Board of Directors.
5. We use fancy visualization and clear diagram to state our model.

5.2.2 Weaknesses

1. Simulated annealing has randomness. Because of the high dimensional constraints, we used SA as our simulation algorithm. We cannot always get the optimal outcomes for limited times of simulation.

2. Some real-time situations are ignored. For each drone, speed cannot always be the same value, and conditions of terrain had better be considered.

3. Although our 3D packaging algorithm considers triple rotations, we consider level rotation. It takes to much time to run the codes of triple rotations.

6 Conclusions

Our paper provides a solution for HELP, Inc. to improve its response capabilities. We offers detailed packing configurations for drone and each of three containers. Three locations are selected in our model to position three containers respectively. We also give the routes of drone fleets, which keeps a balance between delivery mission and reconnaissance mission.

We use given data and real data online to implement our model.Since this is a high dimension problem, we implement SA to find a global optimal outcome.

Modeling outcomes are fully discussed to offer a specific memo to the CEO of HELP,Inc. We believe the anticipated medical supply demands during a potential similar future disaster scenario like Hurricane Maria will be satisfied by applying our modeling outcomes to the DroneGo disaster response system.

7 Memo

Our memo to the CEO of HELP, Inc. is shown in following two pages.

HELP, Inc.

Memo

To: HELP,Inc. CEO

From: Our team

cc: Board of Directors

Date: 01/29/2019

Our model successfully designs a DroneGo disaster response that meet your requirements.

In the following part, we are going to briefly display our modeling results and conclusions. Based on this, we give our recommendations for drone fleets, set of packages, and flight plan.

Modeling Results & Conclusions

- **Three Container Locations & Two kinds of Reigions**

We divide five hospitals into three separated regions. One on the east, one on the west, and three in the middle. Accordingly, we select three locations for these regions respectively.

Container Location	Longitude	Latitude
1	-65.6848	18.2934
2	-66.0710	18.3200
3	-66.7421	18.3932

- **Two kinds of DroneFleets**

Three regions are divided into two kinds. One is delivery-oriented, the other is video reconnaissance-oriented. For the first region, DroneGo chooses Drone B, Drone C to perform missions, while Drone B, Drone C, and Drone F are selected for the latter.

- **Bay Packing Configurations**

Similarly, DroneGo packs its bays according to regions.

• **Satisfied Hospital Demands & Flight Schedule**

Our DroneGo satisfies at least six days demands of each hospital. Drone Fleets don't have to begin their next flight for at least six days.

Hospital	MED 1	MED 2	MED 3	Satisfied Days
1	6	0	6	6
2	18	6	8	8
3	14	9	17	9
4	13	10	15	6.5
5	12	0	2	12

Recommendations

• **Excellent Drone fleet**

We recommend the following drone fleet, as it keep a balance between delivery and reconnaissance, while making good use of space.

Drone Fleet I			Drone Fleet II		
Drone	Bay	Quantity	Drone	Bay	Quantity
B	×	20	B	√	29
C	×	9	C	√	9
F	√	2			

• **Superior Bay Packing Configuration**

We advise a bay packing configuration that not only meets the demands of hospitals, but also makes the best use of space.

Container	MED 1	MED 2	MED 3	Quantity
1	3	0	3	2
3	6	0	1	2

Drone Fleet II Bay 1(Drone B)				Drone Fleet II Bay (Drone C)			
MED 1	MED 2	MED 3	Quantity	MED 1	MED 2	MED 3	Quantity
1	1	0	7	4	1	1	2
0	2	1	5	2	2	2	1
1	0	1	4	1	1	3	1
0	0	2	4	1	0	4	1
2	0	0	3	2	0	3	1
0	0	2	2	5	0	1	1
0	4	0	2	3	1	2	1
1	2	0	2	4	0	2	1
5(sum)	9(sum)	6(sum)	29(sum)	22(sum)	5(sum)	18(sum)	9(sum)

We hope our DroneGo system could help your humanitarian!

References

[1] "Hurricane maria," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Puerto_Rico

[2] L. Z. . K. G. . S. Y. . K. T. . Y. Ji, "Mission planning for uav-based opportunistic disaster recovery networks," in *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2018.

[3] C. Q.-S. C. H.-W. ZHANG De-Fu, WEI Li-Jun, "A combinational heuristic algorithm for the three-dimensional packing problem," *Journal of Software*, vol. 18, pp. 2083–2089, 2007.

[4] "Metropolis-hastings algorithm," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Metropolis-Hastings_algorithm

[5] "Drone flight time," Omino. [Online]. Available: https://www.omnicalculator.com/other/drone-flight-time#drone-flight-time-formula

[6] S. Ross, *First Course in Probability*. Person Education Limited, 2014.

[7] D. D. H. M. P. B. W. Carithers, *Computer Graphics with OpenGL Fourth Edition*. Pearson Education Limited, 2014.

[8] "Field of view," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Field_of_view

[9] "Logistic function," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Logistic_function

Appendices

Appendix A Alternative Configurations for Bay Packing

Table 8: Bay Type 1 Alternative Configurations

Bay	Med1	Med2	Med3	Weight	Utilization Rate
1	0	0	2	6	0.60
1	1	1	0	4	0.62
1	0	2	1	7	0.66
1	0	4	0	8	0.71
1	1	0	1	5	0.74
1	1	2	0	6	0.79
1	2	0	0	4	0.88

Table 9: Bay Type 2 Alternative Configurations

Bay	Med1	Med2	Med3	Weight	Utilization Rate
2	0	5	1	13	0.14
2	0	2	3	13	0.15
2	1	4	1	13	0.17
2	1	1	3	13	0.18
2	2	3	1	13	0.20
2	2	0	3	13	0.21
2	3	2	1	13	0.23
2	4	1	1	13	0.26
2	5	0	1	13	0.29
2	0	4	2	14	0.15
2	0	1	4	14	0.16
2	1	3	2	14	0.18
2	1	0	4	14	0.19
2	2	2	2	14	0.21
2	3	1	2	14	0.24
2	4	0	2	14	0.27
2	0	6	1	15	0.16
2	0	3	3	15	0.17
2	0	0	5	15	0.17
2	1	5	1	15	0.19
2	1	2	3	15	0.20
2	2	4	1	15	0.22
2	2	1	3	15	0.23
2	3	3	1	15	0.25
2	3	0	3	15	0.26
2	4	2	1	15	0.28
2	5	1	1	15	0.31
2	6	0	1	15	0.34

Appendix B Implemented Algorithms and Codes

B.1 Information Provided by the Problem

```
# info.py

import numpy as np

BOUND = np.array([[-67.293988, 17.897988], [-65.568656, 18.539222]])

class Placeable:
    def __init__(self, size):
        self.size = size
        self.volume = self.size[0] * self.size[1] * self.size[2]

class Medicals:
    def __init__(self, weight, size):
        self.weight = weight
        self.size = size

MEDICALS = [
```

```
    Medicals(2, [14, 7, 5]),
    Medicals(2, [5, 8, 5]),
    Medicals(3, [12, 7, 4])
]

class Hospital:
    def __init__(self, loc, req):
        self.location = loc
        self.requirement = req

HOSPITALS = [
    Hospital([-65.65, 18.33], [1, 0, 1]),
    Hospital([-66.03, 18.22], [2, 0, 1]),
    Hospital([-66.07, 18.44], [1, 1, 0]),
    Hospital([-66.16, 18.40], [2, 1, 2]),
    Hospital([-66.73, 18.47], [1, 0, 0])
]

class Bay(Placeable):
    def __init__(self, size):
        Placeable.__init__(self, size)

    def load(self, pkgList):
        self.packages = pkgList # a list representing medical package indices
        self.weight = self.totalWeight()

    def totalWeight(self):
        wt = 0
        for i in range(len(MEDICALS)):
            wt += MEDICALS[i].weight * self.packages[i]
        return wt

BAY_1 = Bay([8, 10, 14])
BAY_2 = Bay([24, 20, 20])

class Drone(Placeable):
    def __init__(self, size, payload, speed, time, video, bay):
        Placeable.__init__(self, size)
        self.payload = payload
        self.speed = speed / 111 # speed in deg/hr
        self.emptyTime = time / 60 # time in hrs
        self.video = video
        self.bay = bay

    def load(self, pkgList):
        self.bay.load(pkgList)
        k = 1500 / 60
        self.fullTime = k / (k / self.emptyTime + self.bay.weight)

DRONES = [
    # Drone([45, 45, 25], 3.5, 40, 35, True, BAY_1), # A is abandeoned
    Drone([30, 30, 22], 8, 79, 40, True, BAY_1), # B: 0
    Drone([60, 50, 30], 14, 64, 35, True, BAY_2), # C: 1
    Drone([25, 20, 25], 11, 60, 18, True, BAY_1), # D: 2
    Drone([25, 20, 27], 15, 60, 15, True, BAY_2), # E: 3
    Drone([40, 40, 25], 22, 79, 24, False, BAY_2), # F: 4
    Drone([32, 32, 17], 20, 64, 16, True, BAY_2) # G: 5
]

NUM_DRONES_PER_CONTAINER = 36
```

B.2 Bézier Curve and Road Map Utility

```
# route.py

import cv2
import numpy as np

import info

BOUND = info.BOUND

def distSq(pt1, pt2):
    return (pt1[0] - pt2[0]) ** 2 + (pt1[1] - pt2[1]) ** 2

class Bezier:
    def __init__(self, pts):
        self.pt = pts
        # Perform integration
        nInteg = 50
        dt = 1 / nInteg
        prevP = self.pt[0]
        len = 0
        for i in range(nInteg):
            newP = self.at((i + 1) * dt)
            len += np.sqrt(distSq(newP, prevP))
            prevP = newP
        self.length = len

    def at(self, t):
        return (1-t)**2 * self.pt[0] + 2*t*(1-t) * self.pt[1] + t**2 * self.pt[2]
```

```
class RoadMap:
    def __init__(self, path):
        rgb = cv2.imread(path)
        gray = cv2.cvtColor(rgb, cv2.COLOR_BGR2GRAY)
        _, self.img = cv2.threshold(gray, 64, 255, cv2.THRESH_BINARY)
        self.bnd = BOUND.transpose()
        self.pixelPerDeg = np.mean(np.array([self.img.shape[1], self.img.shape[0]]) / np.array([BOUN

    def count(self):
        cnt = 0
        for i in range(self.img.shape[0]):
            for j in range(self.img.shape[1]):
                if self.img[i, j] > 0:
                    cnt += 1
        return cnt

    def countAround(self, pt, deg):
        return len(self._getAround(pt, deg * self.pixelPerDeg))

    def _getAround(self, pt, radius):
        imgPtList = []
        x, y = self._locToPixel(pt[0], pt[1])
        for i in range(y - int(radius), y + int(radius)):
            for j in range(x - int(radius), x + int(radius)):
                if i >= self.img.shape[0] or j >= self.img.shape[1]:
                    continue
                if distSq([x, y], [j, i]) <= radius ** 2 and self.img[i, j] > 0:
                    imgPtList.append([j, i])
        return imgPtList

    def eraseAround(self, pt, radius):
        ptList = self._getAround(pt, radius * self.pixelPerDeg)
        self._erase(ptList)

    def _erase(self, imgPtSet):
        for p in imgPtSet:
            self.img[p[1], p[0]] = 0

    def _getAlong(self, bez, radius):
        # Decide segment number
        imgBezPt = []
        for p in bez.pt:
            x, y = self._locToPixel(p[0], p[1])
            imgBezPt.append([x, y])
        imgBez = Bezier(np.array(imgBezPt))
        imgBezLen = imgBez.length
        nSeg = 2 * int(np.ceil(imgBezLen / radius))
        # Count loop
        imgPtSet = set()
        dt = 1 / nSeg
        for seg in range(nSeg):
            lst = self._getAround(bez.at(dt * seg), radius)
            for p in lst:
                imgPtSet.add(tuple(p))
        return imgPtSet

    def countAlong(self, bez, radius = 15):
        st = self._getAlong(bez, radius)
        return len(st)

    def eraseAlong(self, bez, radius = 15):
        st = self._getAlong(bez, radius)
        self._erase(st)

    def show(self):
        cv2.imshow("Road Map", self.img)
        cv2.waitKey(0)

    def write(self, path):
        cv2.imwrite(path, self.img)

    def _locToPixel(self, longi, lati):
        x = self._rangeMap(longi, self.bnd[0], [0, self.img.shape[1] - 1])
        y = self._rangeMap(lati, self.bnd[1], [self.img.shape[0] - 1, 0])
        return int(x), int(y)

    @staticmethod
    def _rangeMap(val, valRange, newRange):
        t = (val - valRange[0]) / (valRange[1] - valRange[0])
        if t < 0 or t > 1:
            return 0
        return (1 - t) * newRange[0] + t * newRange[1]

roadMap = RoadMap("img/roads.png")

if __name__ == '__main__':
    bez = Bezier(np.array([[-66.73, 18.47], [-66.16, 18.40], [-66.07, 18.44]]))
    roadMap.eraseAlong(bez)
    roadMap.show()
```

B.3 Terrain Utility

```
# terrain.py

import info

import cv2
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

BOUND = info.BOUND

class Terrain:
    def __init__(self, path):
        self.imgRgb = cv2.imread(path)
        self.imgHsv = cv2.cvtColor(self.imgRgb, cv2.COLOR_BGR2HSV)
        self.bnd = BOUND.transpose()

    def getHeight(self, pt):
        x, y = self.__locToPixel(pt[0], pt[1])
        hsv = self.imgHsv[y, x]
        if hsv[0] < 10: # indicate that it's black color
            return 0
        return 1.071e4 * np.exp(-0.06058 * hsv[0]) # convert to population density

    def plotHeight(self):
        nStep = 50
        X = np.linspace(BOUND[0, 0], BOUND[1, 0])
        Y = np.linspace(BOUND[0, 1], BOUND[1, 1])
        gX, gY = np.meshgrid(X, Y)
        Z = np.ndarray((nStep, nStep))
        for i in range(nStep):
            for j in range(nStep):
                Z[j, i] = self.getHeight([X[i], Y[j]])
        fig = plt.figure()
        ax = Axes3D(fig)
        ax.plot_surface(gX, gY, Z, rstride=1, cstride=1, cmap=plt.cm.Blues)
        plt.show()

    def __locToPixel(self, longi, lati):
        x = self.__rangeMap(longi, self.bnd[0], [0, self.imgRgb.shape[1] - 1])
        y = self.__rangeMap(lati, self.bnd[1], [self.imgRgb.shape[0] - 1, 0])
        return int(x), int(y)

    @staticmethod
    def __rangeMap(val, valRange, newRange):
        t = (val - valRange[0]) / (valRange[1] - valRange[0])
        if t < 0 or t > 1:
            return 0
        return (1 - t) * newRange[0] + t * newRange[1]

terrain = Terrain("img/terrain.png")

if __name__ == '__main__':
    terrain.plotHeight()
```

B.4 Metropolis-Hastings Sampling and Simulated Annealing

```
# mtr.py
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt

class Metropolis:
    def __init__(self, pLarge, sigma):
        self.pLarge = pLarge # probability of large step
        self.sigma = sigma # standard deviation of perturbation

    def sample(self, func, dim, nSamples):
        samples = []
        prevSample = rd.rand(dim)
        prevVal = func(prevSample)
        total = 0
        while total < nSamples:
            newSample = self.__mutate(prevSample, dim)
            newVal = func(newSample)
            if newVal < prevVal:
                if prevVal == 0: # newVal = preVal = 0, must try again
                    continue
                elif rd.rand() > newVal / prevVal:
                    samples.append(prevSample) # reject new sample
                    total += 1
                    continue
            # otherwise accept
            prevSample = newSample
            prevVal = newVal
            samples.append(newSample)
            total += 1
```

```
        return np.array(samples)

    # Input a function of samples in primary sample space and get the minimum cost
    def anneal(self, func, dim, r, T, T_min, nIte):
        # Initialize samples
        prevSample = rd.rand(dim)
        bestSample = prevSample
        prevVal = func(prevSample)
        bestVal = prevVal

        # Begin annealing loop
        curT = T
        print("Begin annealing")
        while curT > T_min:
            print("T:", curT)
            for _ in range(nIte):
                # Generate new sample
                curSample = self.__mutate(prevSample, dim)

                # Record if best solution so far is found
                curVal = func(curSample)
                if curVal > bestVal:
                    prevVal = bestVal = curVal
                    prevSample = bestSample = curSample
                    print("Best:", bestVal)
                    continue

                # Accept samples or not
                dE = curVal - prevVal
                if dE > 0 or (dE < 0 and rd.rand() < np.exp(dE / curT)):
                    prevVal = curVal
                    prevSample = curSample
                    print("Accepted: ", curVal)

            curT *= r # anneal

        return bestVal, bestSample

    def __mutate(self, sample, dim):
        if rd.rand() < self.pLarge: # large step
            return rd.rand(dim)
        else: # small step
            return self.__wrap(sample + rd.randn(dim) * self.sigma, 0, 1)

    @staticmethod
    def __wrap(val, low, high):
        step = high - low
        ret = []
        for v in val:
            if v < low:
                while v < low:
                    v += step
            elif v >= high:
                while v >= high:
                    v -= step
            ret.append(v)
        return np.array(ret)

if __name__ == '__main__':
    mtr = Metropolis(0.1, 0.005)
    print(mtr.anneal(lambda x: 7 * np.sin(8 * x) + 6 * np.cos(5 * x), 1, 0.9, 1, 0.01, 100))
    X = mtr.sample(lambda x: np.sin(np.pi * x), 1, 10000)
    plt.hist(np.transpose(X).tolist())
    plt.show()
```

B.5 Rotation-Included 3D Packaging Algorithm

```
#judge.py
import numpy as np

length = 231
width = 94
height = 94
def processingB(B):
    _list = [(0,0,0)]
    lx = 0
    lz = 0
    E=np.zeros((length,width,height))
    def sum_(x,y,z,v):
        _sum_ = 0
        for ii in range(v[0]):
            for jj in range(v[1]):
                for kk in range(v[2]):
                    _sum_+=E[x+ii][y+jj][z+kk]
        return _sum_

    def packinv(x,y,z,v):
        if x+ v[0]<= length and y+v[1]<=width and z+v[2]<=height:
            if sum_(x,y,z,v)==0:
                return True
```

```
        return False

    def sort_(_list):
        for i in range(len(_list)):
            for j in range(len(_list)):
                if i<j:
                    if _list[i][1]>_list[j][1] or (_list[i][1]==_list[j][1] and _list[i][0]>_list[j]
                        t=_list[i]
                        _list[i]=_list[j]
                        _list[j]=t
        return _list
    num=0
    success =0
    while num<len(B):
        v= B[num]
        flag = False
        for item in _list:
            x=item[0]
            y=item[1]
            z=item[2]
            if x+ v[0]<= lx and z+v[2]<=lz and packinv(x,y,z,v):
                flag = True
                break
        if flag == False:
            if lx==0 or lx == length:
                if packinv(0,0,lz,v):
                    x=0
                    y=0
                    z=lz
                    flag = True
                    lz= lz+ v[2]
                    lx= v[0]
                elif lz<height:
                    lz=height
                    lx=length
                    num-=1
            else:
                for item in _list:
                    x=item[0]
                    y=item[1]
                    z=item[2]
                    if x==lx and y==0 :
                        if packinv(x,y,z,v) and z+v[2]<=lz:
                            flag = True
                            lx=lx+v[0]
                            break
                if flag == False:
                    lx=length
                    num-=1
        if flag == True:
            for i in range(len(_list)):
                if _list[i]==(x,y,z):
                    _list.pop(i)
                    break
            while x>0:
                if sum_(x-1,y,z,v)==0:
                    x-=1
                    bo=1
                else:
                    break
            while y>0:
                if sum_(x,y-1,z,v)==0:
                    y-=1
                    bo=1
                else:
                    break
            while z>0:
                if sum_(x,y,z-1,v)==0:
                    z-=1
                    bo=1
                else:
                    break
            for ii in range(v[0]):
                for jj in range(v[1]):
                    for kk in range(v[2]):
                        E[x+ii][y+jj][z+kk]=num+1
            _list.append((x+v[0],y,z))
            _list.append((x,y+v[1],z))
            _list.append((x,y,z+v[2]))
            _list = sort_(_list)
            success +=1
        num+=1
    return success == len(B)

def rotate(v,i):
    if i==0:
        return [v[0],v[1],v[2]]
    elif i==1:
        return [v[0],v[2],v[1]]
    elif i==2:
        return [v[1],v[0],v[2]]
    elif i==3:
        return [v[1],v[2],v[0]]
```

```
    elif i==4:
        return [v[2],v[0],v[1]]
    else:
        return [v[2],v[1],v[0]]

def packing3d_(B,now,a):
    if now>=len(B):
        return processingB(a)
    for i in range(1):
        if packing3d_(B,now+1,a+[rotate(B[now],i)])==True:
            return True
    return False

def packing3d(B):
    return packing3d_(B,0,[])
```

```
# package.py

import judge as jd
import info

import numpy as np
import numpy.random as rd
import functools as fc
import copy as cp

MEDICALS = info.MEDICALS

BAY_1 = info.BAY_1
BAY_2 = info.BAY_2

DRONES = info.DRONES

NUM_DRONES_PER_CONTAINER = info.NUM_DRONES_PER_CONTAINER

def packageContainer():
    # Define function that map samples to container layout
    def samplesToIndices(samples):
        indices = list(map(lambda s: int(s * len(DRONES)), samples))
        indices.sort(key=lambda i: DRONES[i].volume + DRONES[i].bay.volume)
        # Try and judge
        popped = 0
        while len(indices) > 0:
            print(indices)
            # Add drone and bay to placeable list
            droneLst = list(map(lambda i: DRONES[i], indices))
            placeLst = list(map(lambda d: d.size, droneLst))
            placeLst.extend(list(map(lambda d: d.bay.size, droneLst)))
            # Judge if the container can pack them items
            if jd.packing3d(placeLst):
                return indices
            else:
                return []
                # Wrap samples and remove an element
                #sp = samples[popped] * 10
                #sp = sp - np.floor(sp)
                #indices.pop(int(sp * len(droneLst)))
                #popped += 1

    def containerUtilization(indices):
        if len(indices) == 0:
            return 0
        volContainer = jd.length * jd.width * jd.height
        volPackageList = list(map(lambda i: DRONES[i].volume + DRONES[i].bay.volume,
            indices))
        volUsed = fc.reduce(lambda x, y: x + y, volPackageList)
        return volUsed / volContainer

    def packageStrategy(indices):
        indicesSet = set(indices)
        pkgDict = {}
        for i in indicesSet:
            pkgDict.update({i: indices.count(i)})
        return pkgDict

    def sample():
        pLarge = 0.7
        sigma = 0.1
        nStrat = 20
        dim = NUM_DRONES_PER_CONTAINER
        func = lambda s: containerUtilization(samplesToIndices(s))

        def wrap(val, low, high):
            step = high - low
            ret = []
            for v in val:
                if v < low:
                    while v < low:
                        v += step
                elif v >= high:
                    while v >= high:
                        v -= step
                ret.append(v)
            return np.array(ret)
```

```
        def mutate(sample, dim):
            if rd.rand() < pLarge: # large step
                return rd.rand(dim)
            else: # small step
                return wrap(sample + rd.randn(dim) * sigma, 0, 1)

        prevSample = rd.rand(dim)
        prevVal = func(prevSample)
        count = 0
        while count < nStrat:
            newSample = mutate(prevSample, dim)
            newVal = func(newSample)
            if newVal < prevVal:
                if prevVal == 0 or rd.rand() > newVal / prevVal: # newVal = preVal = 0, must try aga
                    continue
            # otherwise accept
            prevSample = newSample
            prevVal = newVal
            indices = samplesToIndices(newSample)
            if len(indices) != 0:
                print(packageStrategy(indices), containerUtilization(indices))
            count += 1

    sample()

if __name__ == '__main__':
    packageContainer()
```

B.6 Possible Container Location

```
# location.py

from mtr import Metropolis
import info
import terrain as tr
import route as rt

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from mpl_toolkits.mplot3d import Axes3D

MEDICALS = info.MEDICALS

HOSPITALS = info.HOSPITALS

def distSq(p1, p2):
    return (p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2

def getDensity(pt):
    if tr.terrain.getHeight(pt) == 0:
        return 0
    sigma = 0.1
    func = lambda x: np.exp(-x / (2 * sigma ** 2))
    den = 0
    for h in HOSPITALS:
        for r in h.requirement:
            den += r * func(distSq(pt, h.location))
    return den

def plotDensity():
    nStep = 100
    X = np.linspace(-67.00, -65.50, nStep)
    Y = np.linspace(18.0, 18.7, nStep)
    gX, gY = np.meshgrid(X, Y)
    Z = np.ndarray((nStep, nStep))
    for i in range(nStep):
        for j in range(nStep):
            Z[j, i] = getDensity([X[i], Y[j]])
    fig = plt.figure(figsize=(12.8, 7.2))
    ax = Axes3D(fig)
    ax.plot_surface(gX, gY, Z, rstride=1, cstride=1, cmap=plt.cm.YlGnBu)
    #pcm = ax.pcolor(X, Y, Z, norm=colors.LogNorm(vmin=Z.min(), vmax=Z.max()), cmap='Blues')
    plt.show()

def sample():
    def mapToLocation(sp):
        longi = (1 - sp[0]) * tr.BOUND[0][0] + sp[0] * tr.BOUND[1][0]
        lati = (1 - sp[1]) * tr.BOUND[0][1] + sp[1] * tr.BOUND[1][1]
        return np.array([longi, lati])

    def computeDensity(sp):
        return getDensity(mapToLocation(sp))

    mt = Metropolis(0.1, 0.005)
    samples = mt.sample(computeDensity, 2, 10000)
    locs = list(map(mapToLocation, samples))
    locTran = np.array(locs).transpose()
```

```
    plt.scatter(locTran[0], locTran[1], s=0.2)
    df = pd.DataFrame({"Longitude": locTran[0], "Latitude": locTran[1]})
    df.to_excel("data/location.xlsx", sheet_name='Locations', index=False)
    plt.show()

def optimizeLocation():
    loc = np.array([-65.65, 18.33])
    radius = 0.104260
    def samplesToLoc(samples):
        theta = samples[0] * 2 * np.pi
        r = samples[1] * radius
        pt = loc + r * np.array([np.cos(theta), np.sin(theta)])
        return pt
    def getRoadDensity(pt):
        if tr.terrain.getHeight(pt) == 0:
            return 0
        return rt.roadMap.countAround(pt, radius)

    m = Metropolis(0.1, 0.05)
    val, samples = m.anneal(lambda sp: getRoadDensity(samplesToLoc(sp)), 2, 0.9, 10000, 1000, 10)
    print(val, samplesToLoc(samples))

if __name__ == '__main__':
    #plotDensity()
    optimizeLocation()
```

B.7 Simulation of Response Network

```
# simulation.py

import numpy as np
import numpy.random as rd
import copy as cp
import pandas as pd
import functools as fc

import info
import route as rt
from mtr import Metropolis

MEDICALS = info.MEDICALS
HOSPITALS = info.HOSPITALS
BAY_1 = info.BAY_1
BAY_2 = info.BAY_2
DRONES = info.DRONES

def distSq(p1, p2):
    return (p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2

class Container:
    def __init__(self, loc, nDrones):
        self.location = loc
        self.drones = []
        for i in range(len(DRONES)):
            for _ in range(nDrones[i]):
                self.drones.append(newDrone(i))

def newDrone(idx):
    drone = cp.deepcopy(DRONES[idx])
    return drone

DELIVERY_CONTAINER_DRONES = [29, 9, 0, 0, 0, 0]
VIDEO_CONTAINER_DRONES = [20, 9, 0, 0, 2, 0]

CONTAINERS = [
    Container([-65.6848, 18.2934], VIDEO_CONTAINER_DRONES), # on the north-east
    Container([-66.071, 18.32], DELIVERY_CONTAINER_DRONES), # in the middle
    Container([-66.7421, 18.3932], VIDEO_CONTAINER_DRONES) # on the north-west
]

BAY_1_CANDIDATES = np.array(pd.read_excel("data/bay.xlsx", sheet_name="bay1",
    usecols=range(1, 4), convert_float=True))
BAY_2_CANDIDATES = np.array(pd.read_excel("data/bay.xlsx", sheet_name="bay2",
    usecols=range(1, 4), convert_float=True))

def reuseSample(sp):
    step = 13
    nsp = sp * step
    return nsp - np.floor(nsp)

def simulate():
    def samplesToConfig(samples, verbose):
        config = cp.deepcopy(CONTAINERS)
        delivery = np.zeros((len(HOSPITALS), len(MEDICALS)))
        roadMap = cp.deepcopy(rt.roadMap)
        initalArea = roadMap.count()
        coverage = 0
        if verbose:
            visMap = cp.deepcopy(roadMap)

        nSp = 0
```

```
    for iCont in range(3):
        # Choose container packaging strategy
        container = config[iCont]
        locCtn = container.location
        if verbose:
            print("Container: ", locCtn)

        if iCont == 0 or iCont == 2:
            contArea = roadMap.countAround(locCtn, 0.104260)
            ratio = 1 - np.exp(-2.44 * 29 / 2)
            coverage += contArea / initalArea * ratio
            if verbose:
                visMap.eraseAround(locCtn, 0.104260 * ratio)

        for drone in container.drones:
            # Choose bay packaging configuration
            if (iCont == 0 or iCont == 2) and drone.size != DRONES[4].size:
                continue
            bay = drone.bay
            sp = samples[nSp]
            nSp += 1
            if bay.size == BAY_1.size:
                drone.load(BAY_1_CANDIDATES[int(sp * len(BAY_1_CANDIDATES))])
                sp = reuseSample(sp)
            else:
                if iCont == 0:
                    drone.load(BAY_2_CANDIDATES[24])
                elif iCont == 2:
                    drone.load(BAY_2_CANDIDATES[27])
                else:
                    while True:
                        idx = int(sp * len(BAY_2_CANDIDATES))
                        drone.load(BAY_2_CANDIDATES[idx])
                        if bay.weight > drone.payload:
                            sp = reuseSample(sp)
                        else:
                            break
                    sp = reuseSample(sp)

            # Choose a hospital that the drone can reach
            time = 1.0 / (1.0 / drone.emptyTime + 1.0 / drone.fullTime)
            r = time * drone.speed
            hosCand = list(filter(lambda h: distSq(h.location, locCtn) < r ** 2, HOSPITALS))

            if verbose and len(hosCand) == 0:
                print("Invalid drone!")
                print(locCtn)
                print(drone.__dict__)
                continue

            hospital = hosCand[int(sp * len(hosCand))]
            sp = reuseSample(sp)
            dist = np.sqrt(distSq(locCtn, hospital.location))
            reachTime = dist / drone.speed

            # Transport packages to hospital and scan roads
            # hosIdx = 0
            for iHos in range(len(HOSPITALS)):
                if hospital.location == HOSPITALS[iHos].location:
                    hosIdx = iHos
            if verbose:
                for k in range(len(DRONES)):
                    if DRONES[k].size == drone.size:
                        drnIdx = k
                print("Drone:", drnIdx, "to hospital:", hosIdx)
            delivery[hosIdx] += np.array(bay.packages)
            midPt = (np.array(locCtn) + np.array(hospital.location)) / 2
            roadMap.eraseAlong(rt.Bezier(np.array([locCtn, midPt, hospital.location])))
            if verbose:
                visMap.eraseAlong(rt.Bezier(np.array([locCtn, midPt, hospital.location])))

            # Try returning route that can cover more loads
            if verbose:
                # spline computation is time-consuming
                lMax = drone.speed * drone.emptyTime * (1 - reachTime / drone.fullTime)
                dirct = np.array(hospital.location) - np.array(locCtn)
                nSeg = 10
                rotatedDir = np.array([-dirct[1], dirct[0]])
                seg = rotatedDir / nSeg

                maxArea = 0
                maxIdx = 0
                for i in range(int(-nSeg / 2), int(nSeg / 2) + 1):
                    pCtrl = midPt + i * seg
                    bez = rt.Bezier(np.array([locCtn, pCtrl, hospital.location]))
                    bezLen = bez.length
                    if bezLen >= lMax:
                        continue
                    area = visMap.countAlong(bez)
                    if area > maxArea:
                        maxArea = area
                        maxIdx = i
                visMap.eraseAlong(rt.Bezier(np.array([locCtn, midPt + maxIdx * seg, hospital.loc
```

```
        coverage += (initalArea - roadMap.count()) / initalArea
        coverage += 0.03
        if verbose:
            visMap.write("img/coverage.png")
            coverage = (initalArea - visMap.count()) / initalArea
        return delivery, coverage

    def evaluate(delivery, coverage):
        dlv = 1e10
        for i in range(len(HOSPITALS)):
            for j in range(len(MEDICALS)):
                if HOSPITALS[i].requirement[j] == 0:
                    continue
                dlv = min(float(dlv), float(delivery[i, j] / HOSPITALS[i].requirement[j]))
        return 0.5 * (2 / (1 + np.exp(-dlv / 1.4)) - 1) + 0.5 * coverage

    def function(samples):
        deliv, covr = samplesToConfig(samples, False)
        return evaluate(deliv, covr)

    m = Metropolis(0.1, 0.005)
    val, samples = m.anneal(function, 50, 0.9, 0.1, 0.07, 10)
    #samples = rd.rand(50)
    print(samplesToConfig(samples, True))

if __name__ == '__main__':
    simulate()
```