For office use only	Team Control Number	For office use only
T1 _____	**1923249**	F1 _____
T2 _____		F2 _____
T3 _____	Problem Chosen	F3 _____
T4 _____	**B**	F4 _____

2019
MCM/ICM
Summary Sheet

A Disaster Response System Based on Simulated Annealing Algorithm

Summary

We develop a model to design an aerial disaster response system for medical supply delivery and road-network reconnaissance. And our model consists of three sub-models: loading model, location model and flight planning model. Additionally, our analysis on model shows the extendibility and stability of our model.

In loading model, our goal is to determine the amount of drones and medical packages of each type in each container. We assume that there is no material exchange among containers and thus each container suits three-dimensional knapsack problem. For each container, we minimize the rest space and maximize the amount of each kind of medical packages, under the constraint that the demand of medical packages must be satisfied and drones can deliver all packages in the container. Then we use simulated annealing algorithm to solve this model and the result is intuitively valid.

In location model, we computes optimal locations of the containers given all locations of hospitals and profile of road-network. The basic model is the p-center problem, which is to optimize locations to minimize the distance between a hospital and its nearest container. We assume that the containers can be placed anywhere in the area. The modified model includes the road-network condition and a grid approximation is applied. We solve the basic model using Cooper-NB method and its result is the original solution of iterated local search algorithm for the modified model. The optimal locations that the model produces are (18.3°,-65.8°), (18.4°,-66.1°) and (18.5°,-66.7°).

The fleet is divided into two disjoint sets: one for delivery and the other for reconnaissance. The planning for the former fleet becomes a simple assignment problem to minimize the time spent on transporting all medical packages. For the latter fleet, we reuse the grid approximation and design routes to cover the most major roads. The total time for transportation and reconnaissance is 16 minutes and 40 minutes respectively.

Keywords: 3D-KLP; simulate annealing algorithm; p-center problem; Cooper-NB method; iterated local search

MEMO

To: Chief Operating Officer

From: Team 1923249, MCM 2019

Date: January 28, 2019

Fast and reliable disaster response systems are urgently needed and drones can play a vital role in them because of the advantages in areas with bad road conditions. In order to produce the largest output with minimum cost, the loading configuration, the location decision and the flight plan should be carefully designed. The summary of DroneGo system for Puerto Rico is shown here.

Container Loading Configuration

According to the conditions of Puerto Rico, three 20-foot standard ISO containers are needed to satisfy the demand of medical packages. All containers have the same components and the numbers are: 9 for type-B drone, 33 for type-E drone, 18 for type-G drone; 108 for MED1, 176 for MED2 and 84 for MED3.

Locations of Containers

Three locations are selected considering both medical supply delivery and road network assessment. They are (18.3,-65.8),(18.4,-66.1) and (18.5,-66.7). The choice of locations maximizes the region where roads cross over and ensures the adequate supply to all hospital. One container that lies in the middle provides medical packages for three hospital near San Juan while the other two containers cover Caribbean Medical Center in Fajardo and Hospital Pavia Arecibo in Arecibo.

Flight Planning

The whole fleet in a container is divided into two disjoint sets: the delivery fleet and the reconnaissance fleet. The former fleet can transport all medical packages as soon as possible while the latter one can search all the roads in the largest region centered to the container. To reach the highest transportation efficiency, each drone only carries one type of medical packages. The main force for delivery is type-E and type-G drones while type-B drones are responsible for road-network assessment mainly.

Since the roads in the searching region is less than the roads that the reconnaissance fleet can search in a day, the drones of reconnaissance fleet fly strictly along the roads. However, the routes of delivery fleet are straight lines pointing to destinations.

Contents

1 Introduction

1.1 Problem Background

Drones sometimes can play a more effective role than vehicles in disaster relief. Drones can effectively avoid the obstacles on road and directly deliver the material in need for disaster relief to the places where it is needed. In addition, drones also have significant advantages in detecting road conditions. Therefore, when overland transportation system are paralyzed by large-scale natural disasters , drones can effectively carry out post-disaster road detection and material supply.

In 2017, a hurricane hit Puerto Rico in the United States, leaving nearly 80% of the island's power systems and transportation system damaged. After the hurricane, the island's conditions were severe and medical rescue equipment was urgently needed. Based on the distribution of roads on the island and the demand for medical supplies, an aerial disaster relief response system called "DroneGo" needs to be developed. DroneGo will use rotor wing drones to deliver pre-packaged medical supplies and provide high-resolution aerial video reconnaissance. The DroneGo system consists of a number of drones and a large amount of medical supplies. The whole system will be transported in up to three 20-foot standard ISO containers and placed to single or at most three locations. Each shipping container's contents should be loaded properly in order to minimize any need for buffer materials for unused space.

1.2 Related Research

Model 1 is to solve a modified three-dimensional knapsack loading problem (3D-KLP). The 3D-KLP is a NP-hard problem where a set of boxes should be loaded into a container to maximize the total value of the boxes. Researchers have done a lot of work to solve it using heuristic algorithms. Bortfeldt et al.[1] invented a layer-wise hybrid genetic algorithm (GA) with well-defined data structure and it performed well on problems with heterogeneous boxes. Tiao et al.[2] combined the 3D-KLP and the capacitated vehicle routing problem (CVRP), and proposed a multi-stage hybrid algorithm centered on tabu search (TS) algorithm. Their algorithm were effective on this joint optimization problem according to their paper. Crainic et al.[3] explored a two-level tabu search algorithm that optimize both the container size and the packing of boxes, and it reached state-of-the-art performance at that time.

Facility location problem is a classical operation research problem and it requires to select several locations for facilities to minimize the maximum distance from any demand point to the nearest facility. This problem is proposed by Hakimi and proven to be an NP-hard problem by Hakimi and Kariv[4]. After that, Drezner et al.[5] invented the Drezner-Wesolowsky method to decide m facilities through n demand points using nonlinear programing. With the rise of heuristic algorithms, they have been widely used to solve this problem. Hassin et al.[6] developed a heuristic method based on lexicographic comparsion. A year later, Levin et al.[7] proposed a hybrid method combining clustering

problem and optimal center problem to solve large-scale facility location problems.

1.3 Our Work

Fast and efficient disaster response system can save large amounts of lives and properties and drones can make a difference in it since they can overcome disadvantages of vehicles. In this paper, we design a drone system for both medical supply delivery and road-network search. We divide the whole problem into three parts: container loading, location identification and route planning. In each part, we first form a general model, and then apply it to the Puerto Rico case. The division is based on assumptions we propose, and all assumptions are shown in section 2.

First, we give the optimal loading plan by forming a three-dimensional knapsack loading model based on some assumptions and the given condition. In this model, drones and medical packages are treated as boxes and we aim to maximize the total value of boxes placed in a container. Afterwards we apply the model to Puerto Rico and use simulated annealing algorithm to solve the model. Details are in section 4. Second, we compute the locations of containers that are bounded by two factors: the transportation requirements and the road search task. Our method is to transfer the real problem of locating containers into a well-defined operation research problem, namely the facility location problem (i.e. p-center problem). Then we use a heuristic method named Cooper-NB method to solve it. Considering road search task, we use iterated local search to optimize the result. Third, based on the previous models, we design the routes of drones and the payload packing configurations for each kind of drone, both for delivery and video Reconnaissance. Then we give an overall solution in section 6. Finally, we analyze our models and show the strength and weakness in section 7.

2 Assumptions

We introduce several assumptions to simplify the real-world problem and they are the premises of our models. The assumptions are based on our prior knowledge and we consider both the actual condition and mathematical convenience.

- Containers can be transported to any location in affected area. Based on this assumption, the containers can be placed at any point on Puerto Rico.
- There is no material exchange between any pair of containers. That is, the drones loaded in a single container can complete the transportation of the medical packages and the reconnaissance of the road network.
- The speed of the drone carrying medical packages is the same as the flying speed with no cargo. That is, the one-way flight time is a half of the total flight time, and the distance the drone can fly is not affected by the cargo it carries.

- The way that the drones and medical packages are placed does not affect the medical device itself. Then, the drones and medical package can be placed in six different ways in the container. That is, it is possible to use each of the six faces of the package as the bottom faces.
- The drones that carry medical packages and the drones that do video reconnaissance complete their tasks separately.
- Assume that there are no charging devices in disaster areas, which means all drones can fly once only.

3 Notations

Table 1: Notations

Symbol	Meaning	Unit
L_c	interior length of container	in.
W_c	interior width of container	in.
H_c	interior height of container	in.
Med_i	medical package of type i(i=1,2,3)	
L_{mi}	length of medical package of type i	in.
W_{mi}	width of medical package of type i	in.
H_{mi}	height of medical package of type i	in.
Q_{mi}	weight of medical package of type i	lbs.
Cb_j	drone cargo bay of type j(j=1,2)	
L_{cbj}	length of drone cargo bay of type j	in.
W_{cbj}	width of drone cargo bay of type j	in.
H_{cbj}	height of drone cargo bay of type j	in.
Dr_k	drone of type k($k = A \sim G$)	
Q_{dk}	max payload capability of drone of type k	lbs.
L_{dk}	length of drone's shipping container of type k	in.
W_{dk}	width of drone's shipping container of type k	in.
H_{dk}	height of drone's shipping container of type k	in.
v_{dk}	speed of drone of type k	km/h
T_{dk}	flight time with no cargo of drone of type k	min
N_{dk}	amount of drones of type k	
N_{mi}	amount of medical packages of type i	

4 Model of Packing Configuration for Cargo Containers

4.1 Three-dimensional Loading Model of Cargo Containers

Based on the first and the second assumptions, the containers can be placed to any position in the disaster area and the materials inside each container are independent of each other. That is, it is necessary to ensure that every container is self-sufficient. More specifically, the two tasks (i.e. medical supply delivery and video reconnaissance) must be completed without the help from drones in other containers. Therefore, we only need to design the packing configuration for one independent ISO cargo containers.

For one independent ISO cargo container carrying drones and medical packages, based on the assumptions above, it needs to satisfy the following constraints.

- The sum of the volume of the drones and the medical packages in the container cannot be larger than the interior volume of the container.
- After the drones and medical packages are loaded, the total size of packed drones and medical packages must be less than or equal to the size of the container on each axis.
- The drones in the container must meet the transportation needs of the medical packages and the search work of road networks. That is, drones of type B must be loaded into the cargo container since it has the longest flight distance. And there must be drones carrying cargo bay of type 2.
- Under the premise that the drones can complete the transportation of the medical packages, medical packages must be loaded into the cargo container as many as possible.

Assume that the total number of medical packages of type i is N_{mi}, and the number of drones of type k is N_{dk}. Consider the container as a three-dimensional bin and establish a three-dimensional rectangular coordinate system as shown in Fig. 1.

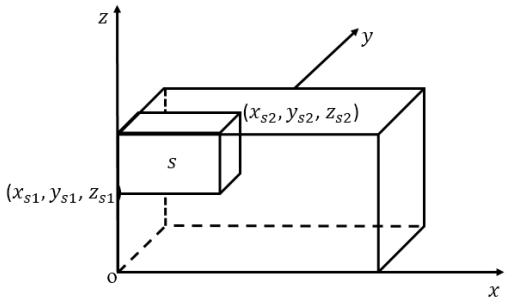

Figure 1: Three-dimensional rectangular coordinate system of the container

Assume that the demand of medical packages of each type in the affected area is N_{ni}. Then, we can get the demand vector $\overrightarrow{N_n}$ for all medical packages. Note that the demand vector is based on the demand for medical packages of the affected area.

$$\overrightarrow{N_n} = (N_{n1}, N_{n2}, N_{n3})$$

It is assumed that the number of medical packages filled in each container can meet the integer multiple demand of the affected area. Then we can get the quantity vector $\overrightarrow{N_m}$of the medical packages loaded into the container.

$$\overrightarrow{N_m} = (N_{m1}, N_{m2}, N_{m3})$$

And there we can get the equation between demand vector and quantity vector of medical packages.

$$\overrightarrow{N_m} = \alpha \cdot \overrightarrow{N_n}$$

Assume that the quantity vector of the drones loaded into the container is $\overrightarrow{N_d}$.

$$\overrightarrow{N_d} = (N_{dA}, N_{dB}, N_{dC}, N_{dD}, N_{dE}, N_{dF}, N_{dG})$$

There, we have built the input vector $\overrightarrow{N_I}$ and the output vector $\overrightarrow{N_O}$ of the three-dimensional loading model.

$$\begin{cases} \overrightarrow{N_I} = \overrightarrow{N_n} \\ \overrightarrow{N_O} = (\overrightarrow{N_m}, \overrightarrow{N_d}) \end{cases}$$

Then we have to establish constraints for the three-dimensional loading model. Based on the assumption that the placement of drones and medical packages does not affect the medical devices themselves. The drones and medical packages can be placed in six different ways in the container. That is, it is possible to use each of the six faces of the packages as the bottom faces. Based on this assumption and Figure 1, we can establish the following constraints.

$$\begin{cases} (x_{s2} - x_{s1}, y_{s2} - y_{s1}, z_{s2} - z_{s1}) \in P_s \\ s = 1, 2, \cdots, n, n = N_m + N_d \\ P_s \in \{(L_{mi}, W_{mi}, H_{mi}), (L_{dk}, W_{dk}, H_{dk})\} \\ i = \{1, 2, 3\} \\ k = \{A, B, C, D, E, F, G\} \\ 0 \leq x_{sj} \leq L_c, 0 \leq y_{sj} \leq W_c, 0 \leq z_{sj} \leq H_c, j = 1, 2 \end{cases}$$

4.2 A Solution based on Simulated Annealing Algorithm

Generally, for any system, when the temperature of the isolated particle system drops at a sufficiently slow rate, it is considered to be approximately thermodynamically balanced. By simulating the cooling process of the classical particle system in thermodynamics, the simulated annealing algorithm can do efficient search and get out from local

optima with high chance. of solving the programming problem. In most cases, simulated annealing algorithm can give a satisfactory solution. Thus it has become an important heuristic method in the past three decades.

4.2.1 The Basic Process of Simulated Annealing Algorithm

Usually, the simulated annealing algorithm consists of 7 steps[8].

Step 1. Initialization. Select the initial temperature T_s and randomly generate the initial status value s. Determine the length L of Markov chain at different temperatures.

Step 2. At the current temperature T_s, from $k = 1$ to L, do steps 3 to 6.

Step 3. According to the current status value, the status value generation function randomly generates a new status value s^*.

Step 4. Calculating the difference $\triangle f = f(s^*) - f(s)$, and f is fitness function.

Step 5. If $\triangle f < 0$, regard s^* as the new status value. Otherwise, a random number $rand$ is generated within the interval of (0,1), and s^* is accepted as the new status value with probability of $exp(-\frac{\triangle f}{T_s}) > rand$.

Step 6. If the termination condition is met, the current status value is the optimal solution to the problem and this is the end of the algorithm. Usually, the termination condition is a series of new status values that are not accepted.

Step 7. Slowly lower the temperature T_s and return to step 2, but ensure that $T_s > 0$ is satisfied.

4.2.2 Solving Three-dimensional Loading Problem

Now we can apply simulated annealing algorithm and give an optimal loading plan given the container model in the previous sector. First, we need to ensure that the drones in a single container can independently complete the transportation of medical packages and the detection of road network. Through observation and calculation, we find that type-A drone is superior to type-B on all performance characteristics such as loading capacity, volume and largest flight distance. So, when finding the solution vector of the 3D loading problem, we set the amount of A-type drones to 0. That means, the output vector is

$$\overrightarrow{N_O} = (0, N_{dB}, N_{dC}, N_{dD}, N_{dE}, N_{dF}, N_{dG}, N_{m1}, N_{m2}, N_{m3}).$$

Additionally, it is necessary to repeat the input, optimization objective and constraints of the model. The input vector is

$$\overrightarrow{N_I} = (N_{n1}, N_{n2}, N_{n3}).$$

The optimization objective is

$$max \sum (N_{dk} L_{dk} W_{dk} H_{dk} + N_{mi} L_{di} W_{di} H_{di}).$$

The constraints are

$$\begin{cases} (x_{s2}-x_{s1}, y_{s2}-y_{s1}, z_{s2}-z_{s1}) \in P_s \\ s = 1, 2, \cdots, n, n = N_m + N_d \\ P_s \in \{(L_{mi}, W_{mi}, H_{mi}), (L_{dk}, W_{dk}, H_{dk})\} \\ i = \{1, 2, 3\} \\ k = \{A, B, C, D, E, F, G\} \\ 0 \le x_{sj} \le L_c, 0 \le y_{sj} \le W_c, 0 \le z_{sj} \le H_c, j = 1, 2 \\ N_{dB} > 0 \\ N_{dk} > 0, k = C, E, F, G \end{cases}.$$

A flow chart for solving the three-dimensional loading problem by the simulated annealing algorithm is shown in Fig. 2.

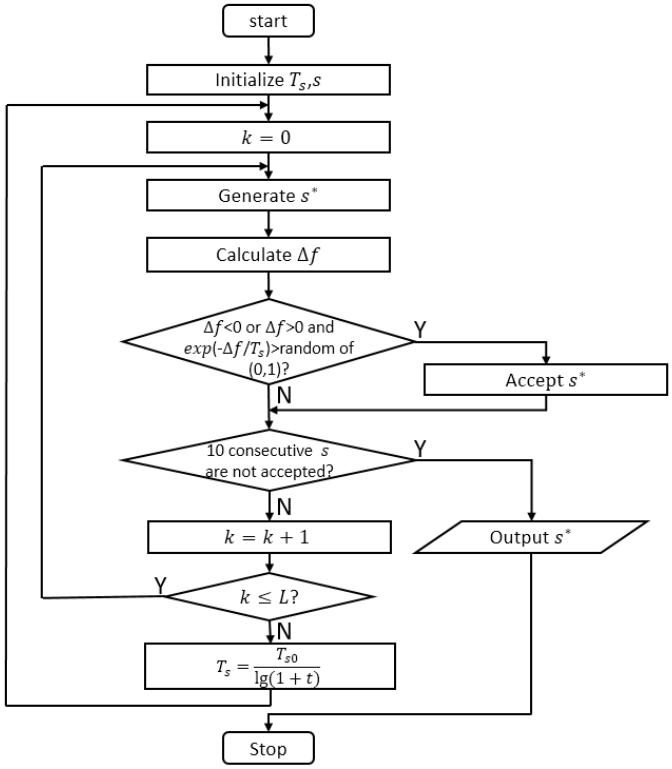

Figure 2: The flow chart of simulated annealing algorithm

(1)Status Value

Since we optimize the number of various types of drones and medical packages, we use the number vector of drones and medical packages as status values. Then we get the calculation method of status value s.

$$s = (0, N_{dB}, N_{dC}, N_{dD}, N_{dE}, N_{dF}, N_{dG}, N_{m1}, N_{m2}, N_{m3})$$

(2)Initialization

There we select the initial temperature T_{s0} as 100. Assume that the initial state value s is $(0, 10, 10, 10, 10, 10, 10, 100, 100, 100)$ and the length L of Markov chain is 1000.

(3)Cooling Method

Here we use the classic cooling method. Assume that the temperature at time t is $T_s(t)$. Then we can get $T_s(t)$.

$$T_s(t) = \frac{T_{s0}}{lg(1+t)}$$

(4)Fitness Function

Here we use the remaining volume of the container as a fitness function. Assume that the fitness function is $f(s)$. Then we can get $f(s)$.

$$f(s) = L_c W_c H_c - \sum (N_{dk} L_{dk} W_{dk} H_{dk} + N_{mi} L_{di} W_{di} H_{di})$$

(5)Termination Condition

When 10 consecutive status values are not accepted, we regard the current status value s is optimal solution.

Through programming, we get the optimal status value of the simulated annealing algorithm. The optimal status value is

$$s = (0, 9, 2, 0, 20, 6, 12, 120, 60, 210).$$

There we get the optimal packing configuration of a cargo container that is shown in Table 2.

Table 2: Packing Configuration of a Cargo Container

	Drones							Medical Packages		
Type	A	B	C	D	E	F	G	MED1	MED2	MED3
Quantity	0	9	0	0	33	0	18	108	176	84

5 Model of the Identification on Containers' Locations

5.1 Optimal Transportation Locations

5.1.1 Additional Assumptions and Validation

In this part, we aim to decide the optimal locations of containers. The containers can be seen as service points and the hospitals are demand points. In this way can we transfer the actual problem to a facility location problem that the service points must be located to minimize the maximum distance from any demand point to its nearest service point. To complete this transfer, we need additional assumptions shown below.

(1) The containers can be transported to any location in the affected area.

(2) The offloading of containers has a constant cost.

(3) The service area of a container is sufficiently large (i.e. the drones in a container can reach every demand point in the service area of that container) and the shape of service area is circular.

We know that a CH-47D helicopter can carry 26,000 pounds externally such as 40-foot containers[9], which shows the 20-foot containers HELP Inc. used can be easily transported in this way. Although the ground condition may limit the offloading of containers, we think the additional assumption 1 is pretty valid.

In the real world, the establishment of a service point has cost that varies with the components of it. But the effect of this cost is very little and considering it causes much unnecessary work. Therefore, we just set the cost to a constant, and furthermore in the model shown below we set it to zero.

The additional assumption 3 seems to be wired, but it can simplify the problem on a great extent. In the sections afterwards, we will see that this additional assumption actually provides constraints to our other two models. Dasci et al.[10] showed that the shape of service area has little effect on the optimization result, thus we choose a simplest one: circular.

With all original and additional assumptions, we can find the optimal locations of containers in the way to solve typical facility location problems. We will use a heuristic method called Cooper-NB method developed by Levin et al.[7] to solve the general problem and apply it to the actual problem.

5.1.2 Cooper-NB Method

Cooper-NB method is a heuristic method to solve large-scale facility location problems. Generally speaking, this method consists of two steps: nearest center reclassification algorithm (see Fig. 3)and single center location optimization.

The first step is to divide all demand points into disjoint clusters using each clusters center. The center here is computed by the algorithm in next sector.

Nearest center reclassification algorithm (NCRA) runs iteratively. In the k th iteration, every demand point joins the cluster whose center is the nearest to the demand point, and a new partition is generated. If the partition no longer changes, the algorithm will stop and the last partition will be seen as the optimal partition. We use Euclidean distance to measure the "nearest" because of application targets.

When the amount of demand points are large, the effect of initial partition is limited while initialization matters with less demand points. Thus, for large amount of demand points, we use random initialization, otherwise we use manual initialization. In extremely large-scale case, we use k-means algorithm because it's much cheaper and efficient to locate m random centers than making a random partition.

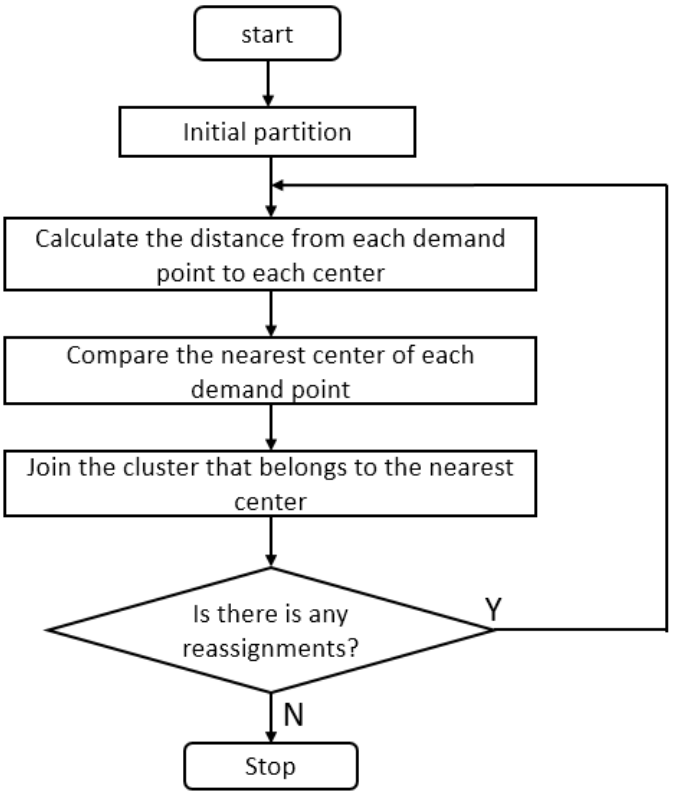

Figure 3: The flow chart of NCRA

5.1.3 Result

Given a cluster of demand points, we need to find the single optimal location for service point (i.e. the center). Let:

- $a_j = (a_j^1, a_j^2)$ be the location of demand point j;
- $a_j \in A$ and A be the set of all demand points;
- x be the current center of cluster;
- d_j be distance between the center and the demand point j;
- w_j be the cost weight of demand point j.

Here we use Euclidean distance, so

$$d_j = \sqrt{\sum_{k=1}^{2}(a_j^k - x^k)^2} = ||a_j - x||$$

The total transportation cost can be written as:

$$C(x) = \sum_{j=1}^{n} w_j ||a_j - x||$$

Note that n is the amount of demand points. Then the gradient of $C(x)$ is

$$\bigtriangledown C(x) = \sum_{j=1}^{n} w_j \frac{x - a_j}{||a_j - x||}.$$

The function is differentiable for all $x \notin \{a_1, a_2, a_3, \ldots, a_n\}$ and the optimal location x^* lies at:

$$\bigtriangledown C(x^*) = \sum_{j=1}^{n} w_j \frac{x^* - a_j}{||a_j - x^*||} = 0.$$

According to our calculus knowledge, x^* must be a convex combination of set A, that is

$$x^* = \sum_{j=1}^{n} \lambda_j(x^*) a_j,$$

where

$$\lambda_j(x) = \frac{w_j ||x - a_j||^{-1}}{\sum_{k=1}^{n} w_k ||x - a_k||^{-1}}.$$

But solving this function directly can be very hard when the scale of demand points increases. Therefore, the authors[7] proposed an iterative method called the Newton-Bracketing method. The key idea of this method is to improve the bounds of the minimum of $C(x)$ iteratively and find a satisfactory solution of x.

An iteration begins with an interval $[L^k, U^k]$ called a $bracket$. The $bracket$ contains the minimum value of $C(x)$, that is

$$L^k \leq C_{min}(x) \leq U^k,$$

and if for an acceptable tolerance ε we have

$$U^k - L^k < \varepsilon,$$

the current x^k is announced to be optimal. Otherwise, we choose a value M^k in the $bracket$ by

$$M^k = \alpha U^k + (1 - \alpha) L^k \quad for \quad some \quad \alpha \in (0, 1).$$

After that we do a one-directional Newtown iteration

$$x^{k+1} = x^k - \frac{C(x^k) - M^k}{|| \bigtriangledown C(x^k)||^2}$$

to get x^{k+1}. Then there are two possible cases here.

- $L^0 = min \quad d_{ij}$, in which case we update the lower bound of the $bracket$ using $L^{k+1} = C(x^{k+1})$.
- $C(x^{k+1}) \geq C(x^k)$, in which cases we update the upper bound of the $bracket$ using $U^{k+1} = C(x^{k+1})$.

The initial $bracket$ $[L^0, U^0]$ is defined as:

$$U^0 = C(x^0)$$

$$L^0 = min \quad d_{ij}$$

where d_{ij} is the distance between every pair of demand points.

When we apply the Cooper-NB method to the Puerto Rico case, we do some modifications on the original method. First, since the space of containers is significantly redundant for minimal drone fleet and medical packages, we set all cost weights equal to 1. Therefore the whole problem becomes a min-max distance problem. In Addition, we manually initialize the partition in case of bad performance resulted from limited data.

The picture Fig. 4 below shows a valid result for containers locations: one is placed at the center of the area where three hospitals locate and the other two faraway hospitals are covered by two container respectively.

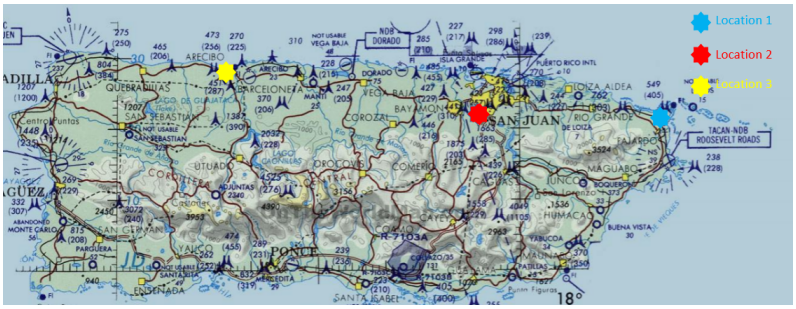

Figure 4: Containers' optimal locations for transportation

5.2 Considering Video Reconnaissance

5.2.1 Grid Approximation and Iterated Local Search

Now we consider the video reconnaissance task. With some primary computation, we find that its impossible to search the whole island from fixed three bases. So the goal becomes to decide the optimal locations of containers so that the drones can cover the most roads. Meanwhile, the selection of locations is bounded by the previous location problem of optimal transportation.

We introduce a grid approximation to model the road network on the island (see Fig. 5). Each cells size is set to 8km$\times$8km, and the drone with the largest flight range can cover three cells. In each cell of the grid, we compute the density of road (i.e. how many roads cross over the cell). Then we get a density picture of road-network in Puerto Rico (see Fig. 6). In Fig. 6, the brighter the area's color , the denser the road network is in this area.

Iterated local search is a simple but efficient algorithm to search local optima. When taking search task into consideration, the new solutions cannot be far from the original

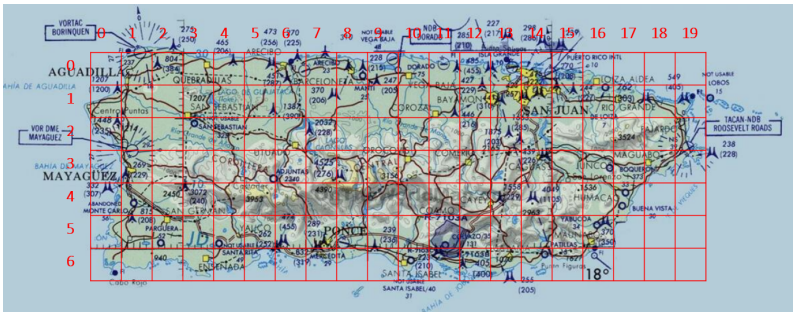

Figure 5: The grid approximation of Puerto Rico

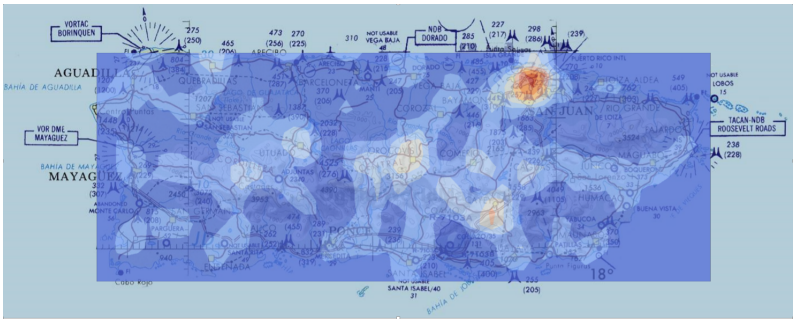

Figure 6: The density picture of road network

solutions since the demands of transportation must be satisfied. Therefore, in this case, we seek for local optima rather than global optima. That's why iterated local search is suitable for this task.

In iterated local search algorithm, the optimal solution is searched by iterating the possible solutions in neighbourhood. Thus the result is largely affected by the original solutions. To bound the final locations near to the optimal locations computed in the previous sector, we use those optimal locations as original solutions.

The goal function leads the direction of search and defines optimal solution. In this case, the goal function is the sum of the 3×3 grid centered on the current searched point. The 3×3 grid is the largest area that the drone can cover from the center point, so this goal function represents the most road that can be covered given a base's location.

In each iteration, a point in the neighbourhood of current solution is randomly selected and the value of the goal function at that point is computed. In this case, neighborhood is defined as the points that locates not beyond one cell. If the value is larger than the best value so far, the new point become the new solution. The process ends when the fluctuation of values of goal function is less than an acceptable tolerance e. Fig. 7 is the follow chart of iterated local search algorithm.

5.2.2 The Final Result of Locations

The new optimal locations are shown below. As we can see, the locations has some subtle change compared to the results in the previous sector. The container in the middle

stays where it used to be since the road-network around San Juan is the densest on the island. The east container moves towards south for some distance because the density of road-network southern area is higher while the west container remains unmoved and we suspect that neighborhood is too limited to allow the search goes beyond the original cell. The results fit our intuition and the actual condition, so we think our model is reasonably valid. Fig. 7 is the final result of locations. The final locations of three cargo containers are shown in Table 3.

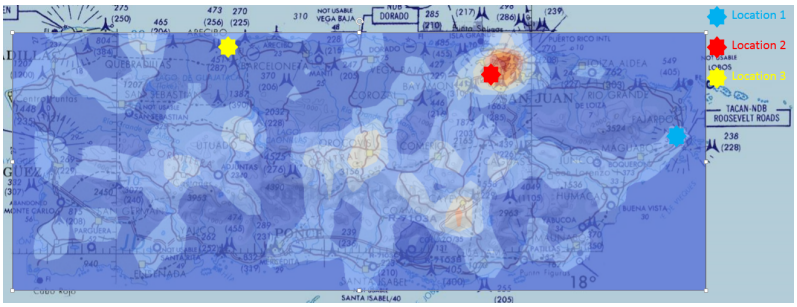

Figure 7: The final result of location

Table 3: The Locations of Three Cargo Containers

	Location 1	Location 2	Location 3
Latitude	18.3	18.4	18.5
Longitude	-65.8	-66.1	-66.7

6 Model of the Dispatch for Drones

6.1 Classification of Drones

In the assumption, we have assumed that all drones have the only amount of electricity for one flight. And charging is not available on the island. So, all drones can only fly once. The drones that carry medical packages and the drones that do video reconnaissance complete their tasks separately. In order to ensure that the drones loaded in the cargo container can complete the transportation of medical packages and video reconnaissance task, we divide the drones into two groups according to the packing capacity and flight distance. First, we calculate the packing capacity of each type of cargo bay(see Table 4). Note that when calculate the maximum loading quantity of each cargo bay, we only consider a simple situation. That is, only the medical packages of the same type are loaded in a cargo bay.

From Table 3, we can see that cargo bay of type 1 can only carry the medical packages of MED2. And cargo bay of type 2 can carry any type of medical packages. Then, we calculate the packing capacity and flight distance of each type of drones(see Table 5) based on Table 4 and attachments.

Table 4: Packing Capacity of Each Type of Cargo Bay

Cargo Bay	Medical Package	Maximum Loading Quantity	Quality(lbs.)
Type 1	MED1	0	0
	MED2	4	8
	MED3	0	0
Type 2	MED1	8	16
	MED2	12	24
	MED3	8	24

Table 5: Packing Capacity and Flight Distance of Each Type of Drones

Type of Drones	Optimal Packing Configurations			Flight Distance
	Only MED1	Only MED2	Only MED3	
A		1		11.6km
B		4		26.3km
C	7	7	4	18.6km
D		4		9.0km
E	7	7	5	7.5km
F	8	11	7	15.8km
G	8	10	6	8.5km

We select the type-B drones for the video reconnaissance task and drones of type E, G to complete the transportation mission.

6.2 Transportation of Medical Packages

Now that we have classified the drones into two types to complete video reconnaissance and transportation mission. Next we have to determine the payload packing configurations, delivery routes and schedule. In this section, the only optimization objective is transportation time. That is, complete the transportation mission in the shortest time. So, we have to consider the speed of drones selected. In model 2, a grid approximation table of Puerto Rico had been established. Now we establish our transportation model according to the grid approximation table.

The input of the model is packing configurations of cargo containers and the locations of each container. The optimization objective is that the drones complete the transportation mission in the shortest time. The constraints are the maximum payload capability and the length of each dimensions. The output are the payload packing configurations and destinations of each drone.

We give each delivery location a number for concise representation(see Table 6). The packing configurations, amount and destinations of contain 1, 2, 3 are shown in Table 7, 8, 9. The delivery routes are straight lines from containers to destinations. Since all drones can only fly once, they can set out at any time after the containers'landing.

Table 6: Location Name and Number of Delivery Locations

Location name	Number
Caribbean Medical Center	1
Hospital HIMA	2
Hospital Pavia Santurce	3
Puerto Rico Children's Hospital	4
Hospital Pavia Arecibo	5

Table 7: Packing Configurations and Destinations of Container 1

Number of Drones	Amount	Packing Configurations			Destinations
		MED1	MED2	MED3	
E1	14	7	0	0	1
E2	19	0	0	5	1
G1	10	0	0	6	1
G2	8	8	0	0	1

Table 8: Packing Configurations and Destinations of Container 2

Number of Drones	Amount	Packing Configurations			Destinations
		MED1	MED2	MED3	
E1	15	7	0	0	2,3,4
E2	18	0	0	5	2,4
G	18	0	10	0	3,4

Table 9: Packing Configurations and Destinations of Container 3

Number of Drones	Amount	Packing Configurations			Destinations
		MED1	MED2	MED3	
E	33	7	0	0	5
G	18	8	0	0	5

6.3 Video Reconnaissance

Only B-type drones complete the video reconnaissance task, so we just send B-type drones to the areas that have the densest road network. Consider that the flight distance of B-type drone is 26.3km, and the side length of each cell grid is 8km$\times$ 8km. So, the flight distance of B-type drone is about 3 cell grids. Fig. 8 is the grid approximation of density. The yellow cell is location 1, the red cell is location 2 and the blue cell is location 3. The numbers in each cell are the road density of the area. Now we search for the best flight plan for the drones with the delivery position of each container as the center.

The input of the model are the positions of containers and the amount of drones. The output of the model are the routes of each drone. The optimization objective is the maximum coverage rate of video reconnaissance. The constraint is that each drone can only move within 3 grid cells. Let: p_{xy} be the position of (x, y),$x \in 0 \sim 19, y \in 0 \sim 6$, n be the amount of drones, d_{xy} be the density value of (x, y), (x_i, y_i) be the position of

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	68	57	72	53	52	35	32	32	34	37	49	49	88	119	114	78	24	24	14	20
1	48	38	50	43	50	52	47	40	27	42	45	49	59	85	76	51	14	13	11	15
2	54	45	54	51	62	68	57	51	46	69	73	81	88	103	92	67	38	36	28	35
3	37	28	35	44	55	75	58	58	50	80	83	97	86	79	57	42	39	35	28	25
4	48	39	53	58	58	59	42	56	64	88	82	94	90	82	64	52	49	41	23	20
5	34	26	34	36	31	32	25	39	41	54	54	64	68	58	46	28	27	16	6	0
6	43	36	49	46	38	26	23	38	44	55	39	46	47	51	49	36	27	13	0	0

Figure 8: The grid approximation of density

container $i(i = 1, 2, 3)$. Then, our model is

$$st.max \sum_{i=1}^{n} d_{xy}\delta_{xy}$$

$$\begin{cases} \delta_{xy} = 1, |x - x_i| \leq 2 and |y - y_i| \leq 2 \\ \delta_{xy} = 0, |x - x_i| > 2 and |y - y_i| > 2 \end{cases}.$$

The input is

$$PI_i = \{(x_i, y_i), n_i\},$$

and the output is

$$PO_i = \{p_{xy}^1, p_{xy}^2, \cdots, p_{xy}^n\}.$$

We solve the model by heuristic search algorithm. Obviously, we can only search the cells within 3 grid cells as the center. The routes of different positions of containers are shown in Fig 9.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	68	57	72	53	52	35	32	32	34	37	49	49	88	119	114	78	24	24	14	20
1	48	38	50	43	50	52	47	40	27	42	45	49	59	85	76	51	14	13	11	15
2	54	45	54	51	62	68	57	51	46	69	73	81	88	103	92	67	38	36	28	35
3	37	28	35	44	55	75	58	58	50	80	83	97	86	79	57	42	39	35	28	25
4	48	39	53	58	58	59	42	56	64	88	82	94	90	82	64	52	49	41	23	20
5	34	26	34	36	31	32	25	39	41	54	54	64	68	58	46	28	27	16	6	0
6	43	36	49	46	38	26	23	38	44	55	39	46	47	51	49	36	27	13	0	0

Figure 9: The flight plan of drones

7 Model Testing and Sensitivity Analysis

If we consider the distance between containers and hospitals, the model becomes a little more complicated. Assume that the largest distance between a container and a hospital is d, we add a module to shrink the solution space by excluding those drones that cannot reach the hospital from the container. Then with the effect of distance d, a series of solution corresponding to the distance are proposed.

Table 10: The relationship between container's configurations and the range of d

	Drone		Medical Package		
Range of d	Type	Quantity	Type	Quantity	Drone Type
$0 < d <= 15km$	B	9	MED1	108	E
	E	33	MED2	176	G
	G	18	MED3	84	E
$15km < d <= 17km$	B	9	MED1	130	G
	G	48	MED2	128	G
			MED3	105	G
$17km < d <= 18km$	B	9	MED1	42	F
	D	25	MED2	99	D
	F	13	MED3	49	F
$18km < d <= 37.3km$	B	9	MED1	39	F
	F	21	MED2	99	F
			MED3	49	F
$37.3km < d$	B	9	MED1	0	None
			MED2	0	None
			MED3	0	None

Firstly, when $d \leq 15km$, all types of drones can reach the hospital, thus the model produces the same solution as ones without distance constraint. But once the time when $17km \leq d > 15km$, since the type-E drones can not reach the destination, our model chooses type-G as the only kind of drone to deliver the medical packages. 48 drones of type G will be used for delivering 130 MED1s, 128 MED2s, and 105 MED3s. Then, when $18km \leq d > 17km$, neither type-E drones nor type-G ones can not reach the hospital. Therefore the model selects type-D and type-F drones. 42 MED1s, 99 MED2s and 49 MED3s will be delivered by 25 type-D drones and 13 type-F drones. Next, at the time when $37.3km \leq d > 18km$, type-D drones cannot arrive at the hospital, but type-F ones can still reach it. Here the model chooses 21 type-F drones to deliver 39 MED1s, 99 MED2s, 49 MED3s. Furthermore, if $d > 37.3km$, obviously, drone B will be the only choose for delivering packets. However, drone B, with the cargo bay of type 1, can only deliver MED2, so in this condition, there is no way to deliver all 3 types of medical package by given drones (In all conditions above, 9 drones of type-B will be put into each container to assess the major highways and roads). Table 10 is the sensitivity analysis of our model.

8 Strengths and Weaknesses

1.Strengths

(1) We introduce some valid assumptions to divide the original problem into three parts and reduce the complexity significantly.

(2) Our models are general, so the extendibility of them are good and they can be easily applied to future cases.

(3) Since we use heuristic methods, our models are good at handling large-scale problem.

2.Weakness

(1) Some useful information is lost due to the partition on the original problem because all sub-problems are with imperfect information under our partition.

(2) We ignore the difference on flight between drones with and without cargo, which is counterfactual.

(3) Our models are static and we don't consider the dynamic change of environment.

(4) Due to using heuristic methods, the solutions may be trapped in local optima.

9 Conclusions

In this paper, we design a post-disaster response system and the result is valid intuitively. In the case of Puerto Rico, our model meets the requirements of medical package delivery and searches the largest area of road network. Moreover, the optimal loading plan balances the amount of medical packages and the load capabilities of drone fleet, which makes it possible to perform efficient transportation and video reconnaissance.

Our model is extensible since we build general models first and then apply it to specific cases. And we believe that our model can produce satisfactory solutions in most cases even though the model may produce wired results sometimes. The result of sensitivity analysis also supports this conclusion. This situation is caused by the use of heuristic methods and the interaction of parts of the model.

References

[1] Bortfeldt A, Gehring H. A hybrid genetic algorithm for the container loading problem[J]. European Journal of Operational Research, 2001, 131(1):143-161.

[2] Tiao Y U , Zhun J , Qiangbeng H . Model and Algorithm for Multi-Objective Joint Optimization of Three-dimensional loading and CVRP[J]. Control & Decision, 2016.

[3] Crainic T G , Perboli G , Tadei R . TS2PACK: A two-level tabu search for the three-dimensional bin packing problem[J]. European Journal of Operational Research, 2009, 195(3):744-760.

[4] Owen S H , Daskin M S . Strategic facility location: A review[J]. European Journal of Operational Research, 1998, 111(3):423-447.

[5] Drezner Z, Wesolowsky G O. A new method for the multifacility minimax location problem.[J]. Journal of the Operational Research Society, 1978, 29(11):1095-1101.

[6] Hassin R, Morad D. Lexicographic local search and the -center problem[J]. European Journal of Operational Research, 2003, 151(2):265-279.

[7] Levin Y , Ben-Israel A . A heuristic method for large-scale multi-facility location problems[J]. Computers & Operations Research, 2004, 31(2):257-272.

[8] Zhang D F, Peng Y, Zhu W X, et al. A Hybrid Simulated Annealing Algorithm for the Three-Dimensional Packing Problem[J]. Chinese Journal of Computers, 2009, 32(11):2147-2156.

[9] `BoeingCH-47DmodelChinookhelicopters.chinook-helicopter.com`

[10] Dasci A , Verter V . A continuous model for productiondistribution system design[J]. European Journal of Operational Research, 2001, 129(2):287-298.

Appendices

Appendix A Python Source Code

```
import cv2
import numpy as np
import math
img = cv2.imread('pic.png', cv2.IMREAD_COLOR)
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

xcenter,ycenter=(525,401)
dx,dy=(70,66)
xrange=(-6,1)
yrange=(-3,17)

scale=1
dx=math.floor(dx/scale)
dy=math.floor(dy/scale)
xrange=tuple([xr*scale for xr in xrange])
yrange=tuple([yr*scale for yr in yrange])

img[xcenter,ycenter,:]=[0,0,255]
img[xcenter,:,:]=[0,0,255]
img[:,ycenter,:]=[0,0,255]

img[xcenter+dx*xrange[0]:xcenter:dx,:,:]=[0,0,255]
img[xcenter:xcenter+dx*xrange[1]+dx:dx,:,:]=[0,0,255]

img[:,ycenter+dy*yrange[0]:ycenter:dy,:]=[0,0,255]
img[:,ycenter:ycenter+dy*yrange[1]+dy:dy,:]=[0,0,255]

cv2.imwrite('grid.png',img)
```

```
import cv2

from grid import xcenter,ycenter,dx,dy,xrange,yrange,img
cv2.imshow("hhhh",img)

from grid_roadfill import roadfill,roads
for i in range(1,roads.size):
    roadfill(i)

import numpy as np
from grid_roadfill import roads
print(np.floor(roads*1000))

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np

X = np.arange(0, roads.shape[1])
```

```
Y = np.arange(roads.shape[0],0,-1)
X, Y = np.meshgrid(X, Y)
Z = roads

cset = plt.contourf(X, Y, Z, zdir='z', cmap=cm.coolwarm)

plt.show()
```

```
import cv2
import numpy as np
from grid import xcenter,ycenter,dx,dy,xrange,yrange,img

img=img[xcenter+dx*xrange[0]:xcenter+dx*xrange[1],
        ycenter+dy*yrange[0]:ycenter+dy*yrange[1]]
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

Lower = np.array([0, 30, 30])
Upper = np.array([20, 200, 200])
mask = cv2.inRange(hsv, Lower, Upper)
Lower[0] = 160
Upper[0] = 180
mask += cv2.inRange(hsv, Lower, Upper)
Lower = np.array([20, 100, 100])
Upper = np.array([40, 255, 255])
mask += cv2.inRange(hsv, Lower, Upper)

print(mask.shape)

nx=xrange[1]-xrange[0]
ny=yrange[1]-yrange[0]
roads=np.zeros([nx,ny])
for i in range(dx):
    for j in range(dy):
        roads+=mask[i::dx,j::dy]
roads/=dx*dy*255

kernel=np.ones([3,3])
roads=cv2.filter2D(roads,-1,kernel)
roads/=9

def roadfill(how_much):
    top_thr=np.sort(roads.reshape(-1))[::-1][how_much]
    print(roads.shape)
    top_locs=np.argwhere(roads>top_thr)
    print(top_locs)

    for top_loc in top_locs:
        print((top_loc[0]*dx,top_loc[1]*dy),
              (top_loc[0]*dx+dx,top_loc[1]*dy+dy))
        cv2.rectangle(img,
                      (top_loc[1]*dy,top_loc[0]*dx),
                      (top_loc[1]*dy+dy,top_loc[0]*dx+dx),
                      (0, 255, 0),
                      -1)
    print(img.shape)
```

```
    cv2.imwrite('roadfill'+str(how_much)+'.png',img)
```

```
import math
import matplotlib.pyplot as plt

class Point:
    def __init__(self,lat,lon):
        self.lat=lat*math.pi/180
        self.lon=lon*math.pi/180

    def __str__(self):
        return 'Lat is '+str(self.lat/math.pi*180)+\
               'Lon is'+str(self.lon/math.pi*180)

    def copy(self):
        return Point(self.lat/math.pi*180,self.lon/math.pi*180)

class Network:
    def __init__(self,sources=[],destinations=[]):
        for p in sources:
            if not isinstance(p,Point):
                raise BaseException('Must be a Point')
        for p in destinations:
            if not isinstance(p,Point):
                raise BaseException('Must be a Point')
        self.sources=sources
        self.destinations=destinations

    def add_source(self,p):
        if not isinstance(p,Point):
            raise BaseException('Must be a Point')
        self.sources.append(p.copy())

    def add_destination(self,p):
        if not isinstance(p,Point):
            raise BaseException('Must be a Point')
        self.destinations.append(p.copy())

    @staticmethod
    def length(p1,p2):
        R=6371004
        C = math.sin(p1.lat)*\
            math.sin(p2.lat)*\
            math.cos(p1.lon-p2.lon)\
            + math.cos(p1.lat)*math.cos(p2.lat)
        return R*math.acos(C)

    @staticmethod
    def cal_minlength_single(p0,ps):
        lmin=float('inf')
        pmin=None
        for p in ps:
            l=Network.length(p0,p)
            if l<lmin:
                lmin=l
```

```
                pmin=p.copy()
        return lmin,pmin

    def minlength(self):
        self.lmin=0
        self.pmins=[]
        for destination in self.destinations:
            lmin,pmin=Network.cal_minlength_single(destination,
                                                   self.sources)
            self.lmin+=lmin
            self.pmins.append(pmin.copy())
        return self.lmin,self.pmins

nw=Network()
nw.add_destination(Point(18.33,-65.65))
nw.add_destination(Point(18.22,-66.03))
nw.add_destination(Point(18.44,-66.07))
nw.add_destination(Point(18.40,-66.16))
nw.add_destination(Point(18.47,-66.73))
nw.add_source(Point(18.22,-66.03))
nw.add_source(Point(18.44,-66.07))
nw.add_source(Point(18.40,-66.16))
for p in nw.destinations:
    print(p)
lmin,pmins=nw.minlength()
print(lmin)
for p in pmins:
    print(p)

cur=0.1
search_points=[]
for lat in range(math.floor(18.22/cur),
                 math.ceil(18.47/cur)+1):
    for lon in range(math.floor(-66.73/cur),
                     math.ceil(-65.65/cur)+1):
        search_points.append(Point(lat*cur,lon*cur))
len(search_points)
for search_point in search_points:
    print(search_point)

hospitals=[Point(18.33,-65.65),
           Point(18.22,-66.03),
           Point(18.44,-66.07),
           Point(18.40,-66.16),
           Point(18.47,-66.73)]
lmin3=float('inf')
pmins3=[]
for p1 in search_points:
    for p2 in search_points:
        for p3 in search_points:
            containers=[p1,p2,p3]
            nw=Network(sources=containers,
                       destinations=hospitals)
            lmin,pmins=nw.minlength()
            if lmin<lmin3:
```

```
                lmin3=lmin
                pmins3=pmins

print(lmin3)
for pmin in pmins3:
    print(pmin)
```

```
import math
class Box:
    def __init__(self,size):
        for s in size:
            if s<=0:
                raise BaseException('box length can not be0')
        self.x,self.y,self.z=tuple(size)

    def turn(self):
        return Box([self.y,self.x,self.z])

    def transpose(self,trans_type):
        size=[self.x,self.y,self.z]
        x=size[0]
        size[0]=size[trans_type%3]
        size[trans_type%3]=x
        if trans_type%2==1:
            y=size[1]
            size[1]=size[2]
            size[2]=y
        return Box(size)

    def reverse_transpose(self,trans_type):
        size=[self.x,self.y,self.z]
        return Box(self.tuple_reverse_transpose(size,trans_type))

    @staticmethod
    def tuple_reverse_transpose(size,trans_type):
        size=list(size)
        if trans_type%2==1:
            y=size[1]
            size[1]=size[2]
            size[2]=y
        x=size[0]
        size[0]=size[trans_type%3]
        size[trans_type%3]=x
        return tuple(size)

    def volume(self):
        return self.x*self.y*self.z

    def size(self):
        return (self.x,self.y,self.z)

    def __sub__(self,box):
        xr=self.x-box.x
        yr=self.y-box.y
        zr=self.z-box.z
```

```
        if xr<0 or yr<0 or zr<0:
            raise BaseException("Can not contain")
        boxrs=[]
        if zr!=0:
            boxrs.append(Box([self.x,self.y,zr]))
        if yr!=0:
            boxrs.append(Box([box.x,yr,box.z]))
        if xr!=0:
            boxrs.append(Box([xr,box.y,box.z]))
        if xr!=0 and yr!=0:
            boxrs.append(Box([xr,yr,box.z]))
        return boxrs

    def __add__(self,box):
        return 0

    def __str__(self):
        return ('L'+str(self.x)+
                ',W'+str(self.y)+
                ',H'+str(self.z))


class Cargo(Box):
    def __init__(self,size,n,m=0):
        super(Cargo,self).__init__(size)
        self.n=n
        self.m=m

    def turn(self):
        cargo=super(Cargo,self).turn()
        return Cargo(cargo.size(),self.n)

    def transpose(self,trans_type):
        if not (trans_type in [0,2,3]):
            raise BaseException(
                'Emergency Medical Package can not upturn')
        cargo=super(Cargo,self).transpose(trans_type)
        return Cargo(cargo.size(),self.n)

    def reverse_transpose(self,trans_type):
        cargo=super(Cargo,self).reverse_transpose(trans_type)
        return Cargo(cargo.size(),self.n)

    def __str__(self):
        return 'Cargo-'+\
               super(Cargo,self).__str__()+\
               ',num'+str(self.n)+\
               ',m'+\
               str(self.m)

def load_meds(box,cargo):
    if(not isinstance(box,Box) or not isinstance(cargo,Cargo)):
        raise BaseException("Class duplicated")
    nx_max=math.floor(box.x/cargo.x)
    ny_max=math.floor(box.y/cargo.y)
```

```
    nz_max=math.floor(box.z/cargo.z)
    n_max=nx_max*ny_max*nz_max
    nz=nz_max\
        if n_max<=cargo.n\
        else math.ceil(cargo.n/(nx_max*ny_max))
    if n_max>0:
        return (nx_max,ny_max,nz),\
               box-Box([nx_max*cargo.x,ny_max*cargo.y,nz*cargo.z])
    else:
        return (nx_max,ny_max,nz),\
               [Box([box.x,box.y,box.z])]

cs,boxrs=load_meds(Box([231,92,94]),Cargo([12,7,4],60))
print(cs)
for boxr in boxrs:
    print(boxr)
def load_meds_trans(box,cargo,trans_type):
    n_size,boxrs=load_meds(box.transpose(trans_type),
                           cargo.transpose(trans_type))
    n_size=Box.tuple_reverse_transpose(n_size,trans_type)
    boxrs=[b.reverse_transpose(trans_type) for b in boxrs]
    return n_size,boxrs

cs,boxrs=load_meds_trans(Box([231,92,94]),Cargo([12,7,4],60),3)
print(cs)
for boxr in boxrs:
    print(boxr)
def best_boxload(box,cargos):
    if cargos==[]:
        return [box],[]
    n_min=float('inf')
    n_size_min=''
    r_box_min=''
    turn=0
    for trans_type in [0,2,3]:
        for i in range(2):
            n_size,r_box=load_meds_trans(
                box,
                cargos[0].turn() if i==1 else cargos[0],
                trans_type)
            n=n_size[0]*n_size[1]*n_size[2]
            if n<n_min:
                n_min=n
                n_size_min=n_size
                r_box_min=r_box
                turn=i==1
    r_box,r_load_plan=best_boxload(r_box_min[0],
                                   cargos[1:])
    return r_box_min[1:]+r_box,[(turn,n_size_min)]+r_load_plan

box=Box([231,92,94])
meds=[]
meds.append(Cargo([14,7,5],60))
meds.append(Cargo([5,8,5],30))
meds.append(Cargo([12,7,4],60))
```

```
meds.append(Cargo([30,30,22],9,3))
r_box,load_plan=best_boxload(box,meds)
print(r_box[-1])
print(load_plan)
def max_loads(box,cargos,max_m):
    load_lists=[[0]*len(cargos)]
    for t in range(len(cargos)):
        load_lists_t=load_lists
        load_lists=[]
        for load_list in list(load_lists_t):
            for n in range(cargos[t].n+1):
                load_list=list(load_list)
                load_list[t]=n
                load_lists.append(load_list)
    load_plans=[]
    for load_list in load_lists:
        m=0
        for t in range(len(cargos)):
            m+=cargos[t].m*load_list[t]
        if m<=max_m:
            cargos_t={}
            for t in range(len(cargos)):
                if load_list[t]>0:
                    cargos_t[t]=Cargo(cargos[t].size(),load_list[t])
            if sum(load_list)>0:
                r_box,load_plan=best_boxload(
                    box,
                    list(cargos_t.values()))
                load_plan_t={}
                for v in zip(list(cargos_t.keys()),load_plan):
                    load_plan_t[v[0]]=v[1]
                load_plans.append(load_plan_t)
    return load_plans
box=Box([8,10,14])
meds=[]
meds.append(Cargo([14,7,5],n=10,m=2))
meds.append(Cargo([5,8,5],n=10,m=2))
meds.append(Cargo([12,7,4],n=10,m=3))
max_loads(box,meds,float('inf'))
class Drone(Box):
    def __init__(self,size,
                 max_load,speed,flight_time,video,
                 load_dict={}):
        super(Drone,self).__init__(size)
        self.max_load=max_load
        self.speed=speed
        self.flight_time=flight_time
        self.video=video
        self.load_dict={}
        for ld in load_dict:
            self.load_dict[ld]=load_dict[ld]

    def turn(self):
        drone=super(Drone,self).turn()
        return Drone(drone.size(),
```

```
                    self.max_load,
                    self.speed,
                    self.flight_time,
                    self.video)

    def transpose(self,trans_type):
        drone=super(Drone,self).transpose(trans_type)
        return Drone(drone.size(),
                    self.max_load,
                    self.speed,
                    self.flight_time,
                    self.video)

    def reverse_transpose(self,trans_type):
        drone=super(Drone,self).reverse_transpose(trans_type)
        return Drone(drone.size(),
                    self.max_load,
                    self.speed,
                    self.flight_time,
                    self.video)

    def __str__(self):
        s='Drone-'+super(Drone,self).__str__()
        s+=',Maxload'+str(self.max_load)
        s+=',Speed'+str(self.speed)
        s+=',Flight time'+str(self.flight_time)
        s+=',Camera'+str(self.video)
        s+=',Load list'+str(self.load_dict)
        return s

cg=Drone([1,2,3],5,3,4,True)
print(cg.turn())
print(cg)
print(cg.size())
print(cg.transpose(2))
print(cg.volume())
drones=[]
load_dict={1:2}
drones.append(Drone([30,30,22],8,79,40,True,load_dict))
load_dict={0:7,1:7,2:4}
drones.append(Drone([60,50,30],14,64,35,True,load_dict))
load_dict={1:2}
drones.append(Drone([25,20,25],11,60,18,True,load_dict))
load_dict={0:7,1:7,2:5}
drones.append(Drone([25,20,27],15,60,15,True,load_dict))
load_dict={0:8,1:11,2:7}
drones.append(Drone([40,40,25],22,79,24,False,load_dict))
load_dict={0:8,1:10,2:6}
drones.append(Drone([32,32,17],20,64,16,True,load_dict))
for drone in drones:
    print(drone)
class Deliver:
    def __init__(self,
                 box,
                 cargos_dict,
```

```
                drones_dict,
                distance,
                box_for_cargo=None):
        self.drones_dict=drones_dict
        if box_for_cargo==None:
            self.cargo_plans,\
            self.cargos_dict,\
            self.r_box=Deliver.get_cargos_load_plan(
                box,
                cargos_dict)
        else:
            self.r_boxs=box-box_for_cargo
            vol_max=0
            for r_box in self.r_boxs:
                if r_box.volume()>vol_max:
                    vol_max=r_box.volume()
                    self.r_box=r_box
            self.cargo_plans,\
            self.cargos_dict,\
            _=Deliver.get_cargos_load_plan(
                box_for_cargo,
                cargos_dict)
        self.drones_plan,\
        self.trans_plan=Deliver.modify_drones(self.r_box,
                                              self.cargos_dict,
                                              drones_dict,
                                              distance)

    @staticmethod
    def get_cargos_load_plan(box,cargos_dict):
        r_box,load_plans=best_boxload(
            box,
            list(cargos_dict.values()))
        _load_plans={}
        _cargos_dict={}
        for load_plan,cargoi in zip(load_plans,cargos_dict):
            cargon=load_plan[1][0]*\
                   load_plan[1][1]*\
                   load_plan[1][2]
            if cargon>0:
                _cargos_dict[cargoi]=Cargo(
                    cargos_dict[cargoi].size(),
                    cargon,
                    cargos_dict[cargoi].m)
                _load_plans[cargoi]=load_plan
        return _load_plans,_cargos_dict,r_box[-1]

    @staticmethod
    def modify_drones(box,cargos_dict,drones_dict,distance):
        drone_plan={}
        for i in drones_dict:
            drone_plan[i]=0
        trans_plan={}
        for cargoi in cargos_dict:
            drone_best=None
```

```
            ndrone_min=0
            vol_min=float('inf')
            for dronei in drones_dict:
                drone=drones_dict[dronei]
                if drone.speed*drone.flight_time>=distance:
                    if cargoi in drone.load_dict:
                        ndrone=math.ceil(
                            cargos_dict[cargoi].n
                            /drone.load_dict[cargoi])
                        vol=ndrone*drone.volume()
                        if vol<vol_min:
                            vol_min=vol
                            ndrone_min=ndrone
                            drone_best=dronei
            trans_plan[cargoi]=drone_best
            if drone_best!=None:
                drone_plan[drone_best]+=ndrone_min
        return drone_plan,trans_plan

    def __str__(self):
        s='Deliver:\n'
        s+='r_box'
        s+=str(self.r_box)+'\n'
        s+='cargos_dict and cargo_plans:\n'
        for i in self.cargos_dict:
            s+=str(i)+':'+str(self.cargos_dict[i])+\
               str(self.cargo_plans[i])+'\n'
        s+='drones_dict:\n'
        for drone in self.drones_dict:
            s+=drone+':'+str(self.drones_dict[drone])+'\n'
        s+='drones_plan:\n'+str(self.drones_plan)+'\n'
        s+='trans_plan:\n'+str(self.trans_plan)+'\n'
        return s

box=Box([231,92,94])

cargos={}
cargos['MED1']=Cargo([14,7,5],60,2)
cargos['MED2']=Cargo([5,8,5],30,2)
cargos['MED3']=Cargo([12,7,4],60,3)
cargos['B']=Cargo([30,30,22],9)

drones={}
load_dict={'MED2':2}
drones['B']=Drone([30,30,22],8,79,40,True,load_dict)
load_dict={'MED1':7,'MED2':7,'MED3':4}
drones['C']=Drone([60,50,30],14,64,35,True,load_dict)
load_dict={'MED2':2}
drones['D']=Drone([25,20,25],11,60,18,True,load_dict)
load_dict={'MED1':7,'MED2':7,'MED3':5}
drones['E']=Drone([25,20,27],15,60,15,True,load_dict)
load_dict={'MED1':8,'MED2':11,'MED3':7}
drones['F']=Drone([40,40,25],22,79,24,False,load_dict)
load_dict={'MED1':8,'MED2':10,'MED3':6}
drones['G']=Drone([32,32,17],20,64,16,True,load_dict)
```

```
dll=Deliver(box,cargos,drones,0,box_for_cargo=Box([30,92,94]))

print(dll)
class ModifiedDeliver(Deliver):
    def __init__(self,box,
                 cargos_dict,drones_dict,distance,
                 except_cargos=[]):
        bxs=[box.transpose(i).x for i in range(6)]
        max_bxlen=max(bxs)
        tran_type=bxs.index(max_bxlen)
        if tran_type!=0:
            raise BaseException('box.x be max')
        box_for_cargo=Box(box.size())
        self.r_vol_min=float('inf')
        for x in range(1,max_bxlen):
            box_for_cargo.x=x
            deliver=Deliver(box,
                            cargos_dict,
                            drones_dict,
                            distance,
                            box_for_cargo)
            if not ModifiedDeliver.cargos_verify(
                    deliver.cargos_dict,
                    cargos_dict):
                continue
            r_box,drones_plan=ModifiedDeliver.load_drone(
                deliver.r_box,
                drones_dict,
                deliver.drones_plan)
            if not ModifiedDeliver.drones_verify(
                    drones_plan,
                    deliver.drones_plan):
                continue
            super(ModifiedDeliver,self).__init__(box,
                                                 cargos_dict,
                                                 drones_dict,
                                                 distance,
                                                 box_for_cargo)
            r_vol=sum([rb.volume() for rb in r_box])
            if r_vol<self.r_vol_min:
                self.drones_plan=drones_plan
                self.r_vol_min=r_vol
                self.r_box_drone=r_box
                super(ModifiedDeliver,self).__init__(
                    box,
                    cargos_dict,
                    drones_dict,
                    distance,
                    box_for_cargo)

    @staticmethod
    def cargos_verify(cargos_dict,min_cargos_dict):
        for ci in min_cargos_dict:
            if not ((ci in cargos_dict)
```

```
                    and
                    (min_cargos_dict[ci].n<=cargos_dict[ci].n)):
                return False
        return True

    @staticmethod
    def load_drone(box,drones_dict,drones_plan):
        drones=[]
        cargos=[]
        for drone in drones_plan:
            if drones_plan[drone]>0:
                cargos.append(Cargo(drones_dict[drone].size(),
                                    drones_plan[drone]))
                drones.append(drone)
        r_box,load_plans=best_boxload(box,cargos)
        drones_plan={drone:0 for drone in drones_plan}
        for drone,load_plan in zip(drones,load_plans):
            drones_plan[drone]=load_plan[1][0]*\
                               load_plan[1][1]*\
                               load_plan[1][2]
        return r_box,drones_plan

    @staticmethod
    def drones_verify(drones_plan,min_drones_plan):
        for di in min_drones_plan:
            if not ((di in drones_plan)
                    and
                    (min_drones_plan[di]<=drones_plan[di])):
                return False
        return True

    def __str__(self):
        s=super(ModifiedDeliver,self).__str__()
        s+='ModifiedDeliver best r_box_drone:\n'
        for box in self.r_box_drone:
            s+=str(box)+'\n'
        s+='ModifiedDeliver best r_vol_min:\n'
        s+=str(self.r_vol_min)
        return s

box=Box([231,92,94])

drones={}
load_dict={'MED2':2}
drones['B']=Drone([30,30,22],8,79,40,True,load_dict)
load_dict={'MED1':7,'MED2':7,'MED3':4}
drones['C']=Drone([60,50,30],14,64,35,True,load_dict)
load_dict={'MED2':2}
drones['D']=Drone([25,20,25],11,60,18,True,load_dict)
load_dict={'MED1':7,'MED2':7,'MED3':5}
drones['E']=Drone([25,20,27],15,60,15,True,load_dict)
load_dict={'MED1':8,'MED2':11,'MED3':7}
drones['F']=Drone([40,40,25],22,79,24,False,load_dict)
load_dict={'MED1':8,'MED2':10,'MED3':6}
drones['G']=Drone([32,32,17],20,64,16,True,load_dict)
```

```
best_mdl=[]
r_vol_min=float('inf')
for mul in range(1,100):
    cargos={}
    cargos['MED1']=Cargo([14,7,5],7*mul,2)
    cargos['MED2']=Cargo([5,8,5],2*mul,2)
    cargos['MED3']=Cargo([12,7,4],4*mul,3)
    cargos['B']=Cargo([30,30,22],9)
    try:
        mdl=ModifiedDeliver(box,
                            cargos,
                            drones,
                            0,
                            except_cargos=['B'])
        if mdl.r_vol_min<r_vol_min:
            r_vol_min=mdl.r_vol_min
            best_mdl=mdl
    except:
        break
print(best_mdl)

def best_deliver(distance):
    best_mdl=[]
    r_vol_min=float('inf')
    for mul in range(1,100):
        cargos={}
        cargos['MED1']=Cargo([14,7,5],7*mul,2)
        cargos['MED2']=Cargo([5,8,5],2*mul,2)
        cargos['MED3']=Cargo([12,7,4],4*mul,3)
        cargos['B']=Cargo([30,30,22],9)
        try:
            mdl=ModifiedDeliver(box,
                                cargos,
                                drones,
                                distance,
                                except_cargos=['B'])
            if mdl.r_vol_min<r_vol_min:
                r_vol_min=mdl.r_vol_min
                best_mdl=mdl
        except:
            break
    return best_mdl

best_delivers={}
for distance in range(0:100:10):
    best_delivers[distance]=best_deliver(distance)

r_box=Box([182,92,94])

#r_drones=[Cargo([30,30,22],100)]

r_drones=[Cargo([60,50,30],100)]
rr_box,load_plan=best_boxload(r_box,r_drones)
print(rr_box[-1])
```

```
print(load_plan)

#r_drones=[Cargo([25,20,25],100)]

r_drones=[Cargo([25,20,27],100)]
rr_box,load_plan=best_boxload(r_box,r_drones)
print(rr_box[-1])
print(load_plan)

r_drones=[Cargo([40,40,25],100)]
rr_box,load_plan=best_boxload(r_box,r_drones)
print(rr_box[-1])
print(load_plan)

r_drones=[Cargo([32,32,17],100)]
rr_box,load_plan=best_boxload(r_box,r_drones)
print(rr_box[-1])
print(load_plan)
```