Model II. Assignment problem to define delivery method - trucks or drones

Basic Introduction:

- Target: Assign each order an appropriate deliverty method in daily routine. There are two options, truck and drones.
- Input: Result from model 1(appropricate locations of warehouses), Order list(200 sample size)Order ID, initial weight of the other parcel in same truck
- Output: truck or drone(0 or 1 - 1 means choose this method)
- Target: minimize the operating cost while satisfying customer's requirement of waiting time

Special Function:

- Automatically select closest warehouse to order, which could help to select optimal warehouse and platform firstly.
- Automatically calculate cost of delivery per order, which incorporate NLP to seperate the conditions that trucks with different number of packages.
- Automatically draw driving direction or flight route by using folium.

Wrote by NI(Mani) MAN, Nov 11, 2017

```
- Import basic library - sys, docplex
```

In [1]:
```
import sys
try:
    import docplex.mp
except:
    if hasattr(sys, 'real_prefix'):
        #we are in a virtual env.
        !pip install docplex
    else:
        !pip install --user docplex
```

```
    - Set up docplex engine
    In this case, we use cplex cloud to solve optimization problem. Please be noted that this ca
se could also be solved by Cplex community with preinstalled IBM Cplex software.
```

In [2]:
```
url = "https://api-oaas.docloud.ibmcloud.com/job_manager/rest/v1/"
key = "api_3d568d67-19d8-4b62-894b-d31981859867"
```

- Step 1 - prepare data

```
- Input order dataset
#0.ID
#1.order address
#2.order_weight
#3.service time remaining

- Define class of order including required property
```

In [3]:
```
class order():
    def __init__(self, ID, address, weight, time_remaining):
        self.id = ID
        self.address = address
        self.weight = weight
        self.time = time_remaining
    def __str__(self):
        return self.id
```

```
- Load Excel dataset
```

In [4]:
```
import xlrd
mydataset=xlrd.open_workbook('Order_target_address.xlsx')
mytable = mydataset.sheets()[0]

ID_l = []
order_l = []
weight_l = []
time_l = []

pos = 0
ID_l.extend(mytable.col_values(pos)[1:])

pos = 1
order_l.extend(mytable.col_values(pos)[1:])

pos = 2
weight_l.extend(mytable.col_values(pos)[1:])

pos = 3
time_l.extend(mytable.col_values(pos)[1:])
```

    - collect order with its property

In [5]:
```
order_property = []

for i in range(len(ID_l)):
    a = ID_l[i]
    b = order_l[i]
    c = weight_l[i]
    d = time_l[i]
    op = order(a, b, c, d)
    order_property.append(op)
```

    - Define Function 1:
    #For one specific order and the closer warehouse assigned, we use google map api get the dri
ving distance and driving time data.
    #According to the google's guideline, this API will call the function simultaneously when yo

u run the code with prediction of traffic situation at that time.
#So, the driving time result might be different when users apply this function to same order at different time according to the real time traffic situation.

In [6]:
```
import googlemaps
import json
def get_driving_distance(origin_address, destination_address):
    gmap = googlemaps.Client(key='AIzaSyBxm210dQEvQ3JBJHL8-A97GggzsX-9pbA')
    distance = gmap.distance_matrix(origin_address, destination_address)
    my_kilometer = distance['rows'][0]['elements'][0]['distance']['value']/1000
    my_min = distance['rows'][0]['elements'][0]['duration']['value']/60
    my_time = distance['rows'][0]['elements'][0]['duration']['text']
    distance_list = []
    distance_list.append(my_kilometer)
    distance_list.append(my_min)
    distance_list.append(my_time)
    return distance_list

#distance_list - 0: km, 1: min, 2: str time
```

- Import warehouse dataset from model 1
- create warehouse list and collect it, these 7 warehouse address is coming from model 1.

In [7]:
```
warehouse_list = []
new_warehouse_1 = '1035 Park Ave New York, NY  10028'
new_warehouse_2 = '207 E 15th St, New York, NY 10003'
new_warehouse_3 = '2552 Holland Ave Bronx, NY 10467'
new_warehouse_4 = '1370 E 18th St Brooklyn, NY 11230'
new_warehouse_5 = '546 W 146th St, New York, NY 10031'
new_warehouse_6 = '32-23 100th St Flushing, NY 11369'
new_warehouse_7 = '88-35 212th Pl Queens Village, NY 11427'
warehouse_list.append(new_warehouse_1)
warehouse_list.append(new_warehouse_2)
warehouse_list.append(new_warehouse_3)
warehouse_list.append(new_warehouse_4)
warehouse_list.append(new_warehouse_5)
warehouse_list.append(new_warehouse_6)
warehouse_list.append(new_warehouse_7)
```

- create warehouse dictionary and collect info.

In [8]:
```
warehouse_dictionary = {}
new_warehouse_1 = '1035 Park Ave New York, NY  10028'
new_warehouse_2 = '207 E 15th St, New York, NY 10003'
new_warehouse_3 = '2552 Holland Ave Bronx, NY 10467'
new_warehouse_4 = '1370 E 18th St Brooklyn, NY 11230'
new_warehouse_5 = '546 W 146th St, New York, NY 10031'
new_warehouse_6 = '32-23 100th St Flushing, NY 11369'
new_warehouse_7 = '88-35 212th Pl Queens Village, NY 11427'
warehouse_dictionary['warehouse_1']= new_warehouse_1
warehouse_dictionary['warehouse_2']= new_warehouse_2
warehouse_dictionary['warehouse_3']= new_warehouse_3
warehouse_dictionary['warehouse_4']= new_warehouse_4
warehouse_dictionary['warehouse_5']= new_warehouse_5
warehouse_dictionary['warehouse_6']= new_warehouse_6
warehouse_dictionary['warehouse_7']= new_warehouse_7
```

- Input information about trucks
  #1.speed - ignore it, since we will use google map api to issue driving distance and time directly.
  #2.driving distance - with using the google map api
  #3.cost per time - cost per kg
    - Trucks input equation
      #cost per kg = (driving distance(km) * gas cost per km(l)(NLP) * gas price ($)(fixed) + maintanence fee per time($)(fixed))/parcel total weight (kg)
      #gas cost per km - linear chart to generate y=ax+b
  #4.capacity(weight/per time)(fixed)(constraint)

-Define Function 2: Define an appropriate warehouse address as origin address with minimal driving distance between itself and order address.

In [9]:
```
def min_driving_origin(order_address):
    origin = {}
    warehouse_1 = get_driving_distance(warehouse_list[0], order_address)[0]
    warehouse_2 = get_driving_distance(warehouse_list[1], order_address)[0]
    warehouse_3 = get_driving_distance(warehouse_list[2], order_address)[0]
    warehouse_4 = get_driving_distance(warehouse_list[3], order_address)[0]
    warehouse_5 = get_driving_distance(warehouse_list[4], order_address)[0]
    warehouse_6 = get_driving_distance(warehouse_list[5], order_address)[0]
    warehouse_7 = get_driving_distance(warehouse_list[6], order_address)[0]
    origin['warehouse_1']= warehouse_1
    origin['warehouse_2']= warehouse_2
    origin['warehouse_3']= warehouse_3
    origin['warehouse_4']= warehouse_4
    origin['warehouse_5']= warehouse_5
    origin['warehouse_6']= warehouse_6
    origin['warehouse_7']= warehouse_7
    min_origin = min(origin.items(), key=lambda x: x[1])[0]
    return warehouse_dictionary[min_origin]
```

- Define function 3: Get driving distance, driving time and driving cost per kg between an appropriate warehouse and order address with a defined arguement, truck_weight_exist.
- In real world, truck_weight_exist depend on total weight of the other parcels. In this model, we simply assigned a value to it.

In [11]:
```
import math
def get_driving_distance_per_order (order_property_i, truck_weight_exist):
    driving_distance_per_order = []
    origin = min_driving_origin(order_property_i.address)
    destination= order_property_i.address
    driving_distance = get_driving_distance(origin, destination)[0]
    truck_time = get_driving_distance(origin, destination)[1]
    truck_total_weight = truck_weight_exist + order_property_i.weight
    gas_cost_per_km = 0.7072*exp(truck_total_weight*0.2985)
    truck_cost_kg = (driving_distance * gas_cost_per_km * 1.1 + 200)/truck_total_weight
    driving_distance_per_order.append(driving_distance)
    driving_distance_per_order.append(truck_time)
    driving_distance_per_order.append(truck_cost_kg)
    print ('For order ' + order_property[i].id + ', driving distance is ' + str(driving_distance) + 'km
    return driving_distance_per_order
```

```
    - Define Function 4:
    #We use google map api get the earth distance and transfer it into km or miles.
    #Please be noted that in this simple model we will not include the simultaneously weather da
ta, so the drone could fly at any time you run this code.
```

In [12]:
```
import googlemaps
import json
try:
    import geopy.distance
except:
    if hasattr(sys, 'real_prefix'):
        #we are in a virtual env.
        !pip install geopy
    else:
        !pip install --user geopy
import geopy.distance
# Simple distance computation between 2 locations.
from geopy.distance import great_circle

def get_flight_distance(origin_address, destination_address):
    gmap = googlemaps.Client(key='AIzaSyD2fUATZAJtzmVCTQi5Fe6xpboAgR5-7J4')
    origin_earth_location = gmap.geocode(origin_address)
    origin_loc=origin_earth_location[0]['geometry']['location']
    destination_earth_location = gmap.geocode(destination_address)
    dest_loc=destination_earth_location[0]['geometry']['location']
    flight_distance = great_circle((origin_loc['lat'], origin_loc['lng']), (dest_loc['lat'], dest_loc['l
    return(flight_distance)
#flight_distance(km)
```

```
    - information about drones
      #1.speed (km/hr)(fixed)
      #2.cost per kg = (distance(km) * electric (kj) per km * electric charge per kj + maintanen
ce fee per time)/parcel weight
      #3.capacity(weight/per time) (fixed)(constraint)
      #4. flight distance - with using the google map api + transfer earth distance to km/mile
```

```
    -Define Function 5: Define an appropriate warehouse address as origin address with minimal f
light distance between itself and order address.
```

In [13]:

```
def get_flight_origin(order_address):
    origin_flight = {}
    warehouse_1 = get_flight_distance(warehouse_list[0], order_address)
    warehouse_2 = get_flight_distance(warehouse_list[1], order_address)
    warehouse_3 = get_flight_distance(warehouse_list[2], order_address)
    warehouse_4 = get_flight_distance(warehouse_list[3], order_address)
    warehouse_5 = get_flight_distance(warehouse_list[4], order_address)
    warehouse_6 = get_flight_distance(warehouse_list[5], order_address)
    warehouse_7 = get_flight_distance(warehouse_list[6], order_address)
    origin_flight['warehouse_1']= warehouse_1
    origin_flight['warehouse_2']= warehouse_2
    origin_flight['warehouse_3']= warehouse_3
    origin_flight['warehouse_4']= warehouse_4
    origin_flight['warehouse_5']= warehouse_5
    origin_flight['warehouse_6']= warehouse_6
    origin_flight['warehouse_7']= warehouse_7
    min_origin = min(origin_flight.items(), key=lambda x: x[1])[0]
    return warehouse_dictionary[min_origin]
```

```
-Define function 6: Get flight distance, flight time and flight cost per kg between an appro
priate warehouse and order address.
```

In [14]:

```
def get_flight_distance_per_order(order_property_i):
    flight_distance_per_order = []
    origin = get_flight_origin(order_property_i.address)
    flight_distance = get_flight_distance(origin, order_property_i.address)
    drone_speed = 60
    flight_time = flight_distance/drone_speed*60
    order_weight = 6
    drone_cost_kg = (flight_distance * (8.2/3600) * 10.19 + 1)/order_weight
    flight_distance_per_order.append(flight_distance)
    flight_distance_per_order.append(flight_time)
    flight_distance_per_order.append(drone_cost_kg)
    print ('For order ' + order_property[i].id + ', Flight distance is ' + str(flight_distance) + 'km an
    return flight_distance_per_order
```

- Step 2: Set up optimization model

```
- Set up the prescriptive model
- create docplex model
```

In [15]:

```
from docplex.mp.environment import Environment
env = Environment()
env.print_information()
from docplex.mp.model import Model
mdl = Model("Delivery Method")
```

```
* system is: Darwin 64bit
* Python is present, version is 3.5.4
* docplex is present, version is (2, 3, 44)
* CPLEX wrapper is present, version is 12.7.0.0, located at: /Applications/anaconda3/envs/cplexenv/li
b/python3.5/site-packages
```

```
    - Before start optimizing, one more thing we should know is that the total weight of parcel
 require an extra input the weight of existed parcels in the truck.
        - In real world, we could input a real number here.
        - In this model, we will randomly deside weight of parcel with using defined weight of e
xisted parcels.
    - the order_pos could help us call one specific order in order_property list.
```

In [16]:

```
truck_weight_exist = 9000
order_pos = 5
```

- Step 3: Define Decision variables

```
- bin_var: truck or drone
```

In [17]:

```
truck = mdl.integer_var(name="truck")
drone = mdl.integer_var(name="drone")
```

- Step4: Define constraints

```
#constraint 1: Choose truck or drone
```

In [18]:
```
mdl.add_constraint(truck + drone == 1)
mdl.print_information()
```

```
Model: Delivery Method
 - number of variables: 2
   - binary=0, integer=2, continuous=0
 - number of constraints: 1
   - linear=1
 - parameters: defaults
```

#constraint 2: drone parcel weight <= drone capacity = 6 kg

In [19]:
```
mdl.add_constraint(drone * order_property[order_pos].weight <= 6)
```

Out[19]:
```
docplex.mp.linear.LinearConstraint[](3drone,LE,6)
```

#constraint 3: customer requirement represented by service time remaining(min), which means one specific order have to delivered in remaining time to satisfy customers' demand.

Functions will be used and their examples:
- get_driving_distance_per_order (order_property_i, truck_weight_exist)
- get_flight_distance_per_order(order_property_i)

In [20]:
```
a= get_driving_distance_per_order(order_property[order_pos], truck_weight_exist)[1]
b= get_flight_distance_per_order(order_property[order_pos])[1]
mdl.add_constraint(truck * a + drone * b <= order_property[order_pos].time)
```

```
For order 0000200, driving distance is 1.006km and driving time is 5.616666666666666mins. The average
cost per kg is 0.022263983116738866dollars.
For order 0000200, Flight distance is 0.741901207796773km and flight time is 0.741901207796773mins. Th
e average cost per kg is 0.16953665653338346dollars.
```

Out[20]:
```
docplex.mp.linear.LinearConstraint[](5.617truck+0.742drone,LE,10)
```

- Step 5: Objective - minimize operating cost

```
Functions will be used and examples:
- get_driving_distance_per_order (order_property_i, truck_weight_exist)

- get_flight_distance_per_order(order_property_i)
```

In [21]:
```
c = get_driving_distance_per_order(order_property[order_pos], truck_weight_exist)[2]
d = get_flight_distance_per_order(order_property[order_pos])[2]
mdl.minimize((truck * c + drone *d)*order_property[order_pos].weight)
```

```
For order 0000200, driving distance is 1.006km and driving time is 5.616666666666666mins. The average
cost per kg is 0.022263983116738866dollars.
For order 0000200, Flight distance is 0.741901207796773km and flight time is 0.741901207796773mins. Th
e average cost per kg is 0.16953665653338346dollars.
```

- Step 6: Print info. + Print results

In [22]:
```
mdl.print_information()
s = mdl.solve(url = url, key=key)
```

```
Model: Delivery Method
 - number of variables: 2
   - binary=0, integer=2, continuous=0
 - number of constraints: 3
   - linear=3
 - parameters: defaults
```

In [23]:
```
mdl.print_solution()
```

```
objective: 0.067
  truck=1
```

- Step 7: Visualization of result

```
- Get the cplex optimization result
    -Decision variable result
```

In [24]:
```
if_drone = drone.solution_value
if_truck = truck.solution_value
print(if_drone, if_truck)
```

```
0 1.0
```

If result is choosing truck:
- Maping truck driving route

If result is choosing drone:
- Maping drone flight route

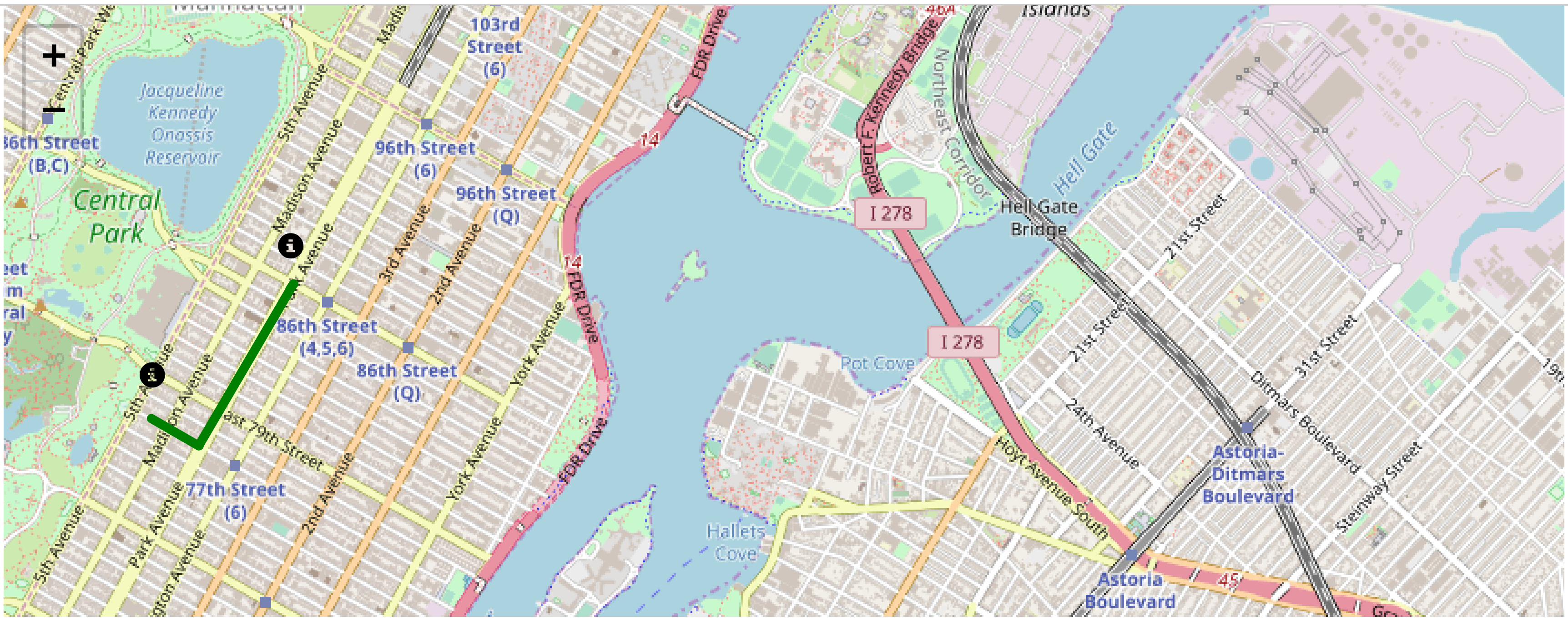
In [25]:
```python
import folium

origin_address = get_flight_origin(order_property[order_pos].address)
destination_address = order_property[order_pos].address
f_map = googlemaps.Client(key='AIzaSyD2fUATZAJtzmVCTQi5Fe6xpboAgR5-7J4')
o_loc = f_map.geocode(origin_address)
lat = o_loc[0]['geometry']['location']['lat']
lng = o_loc[0]['geometry']['location']['lng']
map_osm = folium.Map(location=[lat, lng], zoom_start=14)

if if_truck == 1:
    origin_address = min_driving_origin(order_property[order_pos].address)
    destination_address = order_property[order_pos].address
    t_map = googlemaps.Client(key='AIzaSyAT04oE2KhMD4nHgHDLDW-5L4Ip18tYlSs')
    direction = t_map.directions(origin_address, destination_address)
    loc = direction[0]['legs'][0]['steps']
    my_direction = []
    for i in range(len(loc)):
        my_lat_1 = loc[i]['start_location']['lat']
        my_lng_1 = loc[i]['start_location']['lng']
        my_lat_2 = loc[i]['end_location']['lat']
        my_lng_2 = loc[i]['end_location']['lng']
        my_direction.append([my_lat_1, my_lng_1])
        my_direction.append([my_lat_2, my_lng_2])
    for j in range(len(my_direction)):
        if j < (len(my_direction)-1):
            coordinates = [my_direction[j], my_direction[j+1]]
            map_osm.add_child(folium.PolyLine(coordinates, color='green', weight=5))

    folium.Marker(my_direction[0], icon=folium.Icon(color='red',icon='info-sign')).add_to(map_osm)
    folium.Marker(my_direction[-1], icon=folium.Icon(color='blue',icon='info-sign')).add_to(map_osm)
elif if_drone == 1:
    origin_address = get_flight_origin(order_property[order_pos].address)
    destination_address = order_property[order_pos].address
    f_map = googlemaps.Client(key='AIzaSyD2fUATZAJtzmVCTQi5Fe6xpboAgR5-7J4')
    o_loc = f_map.geocode(origin_address)
    d_loc = f_map.geocode(destination_address)
    o_lat = o_loc[0]['geometry']['location']['lat']
    o_lng = o_loc[0]['geometry']['location']['lng']
    d_lat = d_loc[0]['geometry']['location']['lat']
    d_lng = d_loc[0]['geometry']['location']['lng']
    folium.Marker([o_lat, o_lng], icon=folium.Icon(color='red',icon='info-sign')).add_to(map_osm)
```

```
    folium.Marker([d_lat, d_lng], icon=folium.Icon(color='blue',icon='info-sign')).add_to(map_osm)
    coordinates_f = [[o_lat, o_lng], [d_lat, d_lng]]
    map_osm.add_child(folium.PolyLine(coordinates_f, color='red', weight=5))
```

- Draw map

In [26]:
```
map_osm
```

Out[26]:



End

In [ ]:
```
#End
```