Supplementary Materials

File 1. Main location-allocation models used in the literature and their implications for use in this study

Applied model (reference)	Author (year)	Model description	Implication for use in this study
P-median	Church (1990)	Optimally locates p facilities to minimize the average travel time	NP-hard; ignores remote demands
P-center	Revelle (1989)	Optimally locates p facilities to minimize the maximum travel time	NP-hard; Not optimize the average travel time
Location Set Covering Problem (LSCP)	Toregas (1971)	to find the minimum number of facilities and their locations to cover all of the demand points in a pre-defined standard	Requires a lot of facilities to cover all demands
Maximal Covering Location Problem (MCLP)	Church and ReVelle (1974)	to locate a fixed number of facilities to maximize the total demand covered by at least one facility	The capacity of ambulances is ignored.
Dynamic Double Standard Model (DDSM)	Gendreau et al. (2001)	Relocate ambulances optimally at time (t) when a request is registered	Real time data is needed.
Capacitated MCLP	Current and Storbeck (1988)	Adds maximum capacity constraint to MCLP formulation	Some remote demands could be ignored

File 2. Location allocation model script

```
import pandas as pd
import numpy as np
import csv

try:
    import docplex.mp
except:
    raise Exception('Please install docplex. See https://pypi.org/project/docplex/')

#load datasets
#load excel file of stations with including ID and Number of ambulance vehicles
def read_candidate_sites(candidates_csv_path):
    df = pd.read_csv(candidates_csv_path)
    df["NumberOfAmbulances"] = df["NumberOfAmbulances"].fillna(value=0)
    return df

#load excel file of demand points including ID, Call frequancy, and location (urban/rural)
def read_demands(demand_csv_path):
```

```
    df = pd.read_csv(demand_csv_path)
    df["CallFreq"] = df["CallFreq"].fillna(value=0)
    return df


#load cost matrix including FacilityID, DemandID, DriveTime
def create_OD_matrix(OD_csv_path, time_threshold):
    df = pd.read_csv(OD_csv_path)
    # determine the cover relationship based on the travel time and the service standard
    df["Covered"] = np.where(df["DriveTime"] < time_threshold, 1, 0)
    # create pivot table for OD matrix
    pivot = df.pivot("StationID", "DemandID", "Covered")
    return pivot


# Scenario 0: Current distribution
def calculate_objective_with_current_car_distribution(demand_csv, candidates_csv,
ODMatrix_csv, time_threshold,
                                                      unit_car_capacity):
    # read input data for the model
    demands = read_demands(demand_csv)
    candidates = read_candidate_sites(candidates_csv)
```

```
    coverage_matrix = create_OD_matrix(ODMatrix_csv, time_threshold)


    from docplex.mp.environment import Environment
    env = Environment()
    env.print_information()


    from docplex.mp.model import Model
    mdl = Model("EMS vehicles")
    # Define the decision variables
    # percentage of demand i covered by facility j
    y_i_j_vars = mdl.continuous_var_matrix(demands["DemandID"], candidates["StationID"],
ub=1, name="y")


    # add constraints
    # ct1: the allocated demand should not exceed the capacity of the facility
    for j in candidates["StationID"]:
        mdl.add_constraint(mdl.scal_prod([y_i_j_vars[i, j] for i in demands["DemandID"]],
demands["CallFreq"])
                           <= unit_car_capacity * candidates.loc[
                               candidates["StationID"] == j, "NumberOfAmbulances"].item())
```

```
    # ct2: The allocated demand at i should not exceed 100%
    for i in demands["DemandID"]:
        mdl.add_constraint(mdl.sum(y_i_j_vars[i, j] for j in candidates["StationID"]) == 1)

    # express the objective
    total_covered_demand = mdl.sum(y_i_j_vars[i, j] * demands.loc[demands["DemandID"]
== i, "CallFreq"].item() *
                                   coverage_matrix.loc[j, i]
                                   for i in demands["DemandID"]
                                   for j in candidates["StationID"])
    mdl.maximize(total_covered_demand)
    mdl.print_information()
    # solve the model
    mdl.solve()
    # print the solution
    print("Total covered demand = %g" % mdl.objective_value)

#Scenaro 1 and 2: Relocatoin and Allocation model
```

```
def mcmclp_additive_model(demand_csv, candidates_csv, ODMatrix_csv, time_threshold, unit_car_capacity,
                          maximal_cars_per_site,Output_csv, total_added_cars, additive_mode=0):
    # read input data for the model
    demands = read_demands(demand_csv)
    candidates = read_candidate_sites(candidates_csv)
    coverage_matrix = create_OD_matrix(ODMatrix_csv, time_threshold)
    max_matrix_rural = create_OD_matrix(ODMatrix_csv, max_time_rural)  #Create an upper bound coverage matrix for rural demands
    max_matrix_urban = create_OD_matrix(ODMatrix_csv, max_time_urban)  #Create an upper bound coverage matrix for urban demands

    from docplex.mp.environment import Environment
    env = Environment()

    # create a model
    from docplex.mp.model import Model
    mdl = Model("EMS vehicles")

    # define the decision variables
```

```
    # dv1: the percentage of demand i covered by facility j
    y_i_j_vars = mdl.continuous_var_matrix(demands["DemandID"], candidates["StationID"],
ub=1, name="y")

    # dv2: the number of cars added at a station
    x_j_vars = mdl.integer_var_dict(candidates["StationID"], name="x")

    # add constraints
    # ct1: the allocated demand should not exceed the capacity of the facility
    # ct2: The total number of cars at each site should not exceed maximal_cars_per_site
    for j in candidates["StationID"]:
        num_existing_cars = candidates.loc[candidates["StationID"] == j,
"NumberOfAmbulances"].item()
        if additive_mode == 0:
            num_existing_cars = 0
        num_total_cars = num_existing_cars + x_j_vars[j]
        mdl.add_constraint(num_total_cars <= maximal_cars_per_site)
        mdl.add_constraint(mdl.scal_prod([y_i_j_vars[i, j] for i in demands["DemandID"]],
demands["CallFreq"])
                           <= unit_car_capacity * num_total_cars)
```

```
    # ct3: the total number of EMS vehicles added should be equal to total_added_cars
    mdl.add_constraint(mdl.sum(x_j_vars[j] for j in candidates["StationID"]) == total_added_cars)

    # ct4: The allocated demand at i should not exceed 100%
    for i in demands["DemandID"]:
        mdl.add_constraint(mdl.sum(y_i_j_vars[i, j] for j in candidates["StationID"]) == 1)

    # ct5: Upper bound for urban and rural demands                #NEW
    for i in demands.loc[demands["Urban"] == 1,"DemandID"]:
        mdl.add_constraint(mdl.sum(max_matrix_urban.loc[j,i]* x_j_vars[j] for j in candidates["StationID"]) >= 1)
    for i in demands.loc[demands["Urban"] == 0,"DemandID"]:
        mdl.add_constraint(mdl.sum(max_matrix_rural.loc[j,i]* x_j_vars[j] for j in candidates["StationID"]) >= 1)

    # express the objective
    total_covered_demand = mdl.sum(y_i_j_vars[i, j] * demands.loc[demands["DemandID"] == i, "CallFreq"].item() *
                                   coverage_matrix.loc[j, i]
```

```
                        for i in demands["DemandID"]
                        for j in candidates["StationID"])
mdl.maximize(total_covered_demand)
mdl.print_information()


# solve the model
mdl.solve()


# print the solution
print("Total covered demand = %g" % mdl.objective_value)
total_cars = 0
for j in candidates["StationID"]:
    num_added_cars = round(x_j_vars[j].solution_value)
    total_cars += num_added_cars
    if (num_added_cars>0):
        print("{0} #vehicles added: {1!s}".format(j, num_added_cars))
#     print([y_i_j_vars[i, j].solution_value for i in demands["DemandID"]])


print("total cars added: {0!s}".format(total_cars))
```

'''

Run the models

'''

Model parameters

input_data_folder = r"Folder_Path"

demand_csv = input_data_folder + "\Demand.csv"

candidates_csv = input_data_folder + "\Stations.csv"

ODMatrix_csv = input_data_folder + "\ODMatrix.csv"

Output_csv= input_data_folder + "\output.csv"

time_threshold = 5 # minutes

unit_car_capacity = 2387 # 224355/94

maximal_cars_per_site = 3

max_time_rural= 16 #minutes

max_time_urban= 48 #minutes

use scenario variable to control which model will run

0 - assess the covered demand based on the current distribution of EMS vehicles

1 - scenario 1 where all cars can be relocated for optimization

2 - scenario 2 where existing cars remain and added cars are optimally located

```
scenario = 2


# run the model


if scenario == 1:  # run the model for scenario 1 and 2
    mcmclp_additive_model(demand_csv, candidates_csv, ODMatrix_csv, time_threshold, unit_car_capacity,
                          maximal_cars_per_site,Output_csv,
                          total_added_cars=94, additive_mode=0)
elif scenario == 2:  # run the model for scenario 2
    for i in range (0,11): #adding zero to 10 new vehicles
        mcmclp_additive_model(demand_csv, candidates_csv, ODMatrix_csv, time_threshold, unit_car_capacity,
                          maximal_cars_per_site, Output_csv,
                          total_added_cars=i, additive_mode=1)
elif scenario == 0:  # assess the covered demand based on the current distribution of EMS vehicles
    calculate_objective_with_current_car_distribution(demand_csv, candidates_csv, ODMatrix_csv, time_threshold,
                                                      unit_car_capacity)
```

File 3. Comparison of current distribution (scenario 0) vs. relocated distribution (scenario 1) of ambulance vehicles in Mashhad, Iran.

Scenario:			0 : Current situation	1 : Relocation model
Total covered demands (%)			155,617 (69.36%)	168,700 (75.19%)
Number of EMS stations (number of ambulance vehicles)			55(1) + 18(2) + 1(3)	12(0) + 37(1) + 18(2) + 7(3)
Station ID	location	(longitude, latitude)	No. of vehicles	No. of vehicles
1	Urban	(59.5806,36.2919)	2	2
2	Urban	(59.6258,36.2861)	2	0
3	Urban	(59.5819,36.3308)	2	3
4	Urban	(59.5103,36.3300)	3	3
5	Urban	(59.6428,36.3008)	2	3
6	Urban	(59.4944,36.3650)	1	1
7	Urban	(59.5406,36.2842)	1	1
8	Urban	(59.6036,36.2478)	2	3
9	Urban	(59.5481,36.3292)	2	1
10	Urban	(59.6578,36.3150)	1	2
11	Urban	(59.6131,36.3144)	1	2
12	Urban	(59.6628,36.2708)	2	1
13	Urban	(59.5911,36.2747)	1	0
14	Urban	(59.5956,36.3431)	2	1
15	Urban	(59.5150,36.3594)	1	2
16	Urban	(59.6111,36.2864)	1	3
17	Urban	(59.6486,36.3300)	1	1
18	Urban	(59.5908,36.2858)	1	2
19	Urban	(59.5964,36.3042)	1	1
20	Urban	(59.6222,36.2978)	1	0
21	Urban	(59.6033,36.2814)	1	0
22	Urban	(59.6519,36.2642)	1	1
23	Urban	(59.5172,36.3025)	2	2
24	Urban	(59.6072,36.2700)	2	1
25	Urban	(59.5639,36.3167)	1	1
26	Urban	(59.6344,36.2683)	2	2
27	Urban	(59.6172,36.3147)	1	2
28	Urban	(59.6256,36.2197)	2	0
29	Urban	(59.5431,36.3725)	2	1
30	Urban	(59.6194,36.3289)	2	2
31	Urban	(59.4975,36.3150)	2	3
32	Urban	(59.6214,36.2678)	1	2
33	Urban	(59.5917,36.2611)	1	1
34	Urban	(59.6739,36.2997)	1	2
35	Urban	(59.5989,36.2906)	1	2
36	Urban	(59.6519,36.1139)	1	0
37	Urban	(59.5431,36.2869)	1	1
38	Urban	(59.4733,36.3756)	1	1

39	Urban	(59.6789,36.3147)	2	0
40	Urban	(59.5383,36.3583)	1	1
41	Urban	(59.4681,36.3642)	1	1
42	Urban	(59.4981,36.3217)	1	1
43	Urban	(59.6011,36.2944)	1	2
44	Urban	(59.5428,36.34)	1	1
45	Urban	(59.6247,36.3492)	1	1
46	Urban	(59.4858,36.3606)	1	0
47	Urban	(59.6414,36.3156)	1	2
48	Urban	(59.4767,36.3364)	1	2
49	Urban	(59.6781,36.3214)	2	2
50	Urban	(59.5353,36.3153)	1	1
51	Urban	(59.5125,36.3611)	1	1
52	Urban	(59.605,36.3328)	1	2
53	Urban	(59.6417,36.2669)	1	0
54	Urban	(59.5083,36.3964)	1	3
55	Rural	(59.7167,36.4483)	1	1
56	Urban	(59.6425,36.2272)	1	0
57	Urban	(59.6644,36.2133)	1	1
58	Rural	(59.7283,36.2372)	1	1
59	Rural	(59.5136,36.4844)	1	1
60	Rural	(59.6725,36.4328)	1	1
61	Rural	(59.6501,36.4633)	1	0
62	Urban	(59.6939,36.2753)	1	1
63	Rural	(59.4892,36.4189)	2	1
64	Urban	(59.5172,36.37)	1	1
65	Urban	(5961404,362904)	1	2
66	Rural	(59.8375,36.5856)	1	1
67	Rural	(59.5836,36.9889)	1	1
68	Rural	(59.9603,36.0783)	1	1
69	Rural	(59.3911,36.7858)	1	1
70	Rural	(59.7367,36.0947)	1	1
71	Rural	(59.3581,36.985)	1	1
72	Rural	(59.4225,36.8886)	1	1
73	Rural	(59.1278,36.2586)	1	0
74	Rural	(59.6694,36.6539)	1	1