uOttawa

ELG 5142:
Ubiquitous Sensing / Smart Cities
Assignment 1

Dr. Burak Kantarci

GROUP 14

Table of Contents

Table of Figures

1. Generating tasks:

To obtain the required 2000 tasks, many parts in the code are modified:

- The for loop is modified to write only one text file instead of ten files.

```
    f1T = 0
    f1F = 0
    for run_num in range(1,2):
```

- The number of tasks is modified to be 2000 tasks.

```
num_tsk = 2000#int(num_usr/10)
```

- The code that is used to write the text and csv files for the generated tasks is modified to include the task value.

```
# task txt file
        task_file = open('./Inputs/Mobility/differentradius/50/'+str(run_num)+'/Tasks.txt', 'w')
        task_file.write("/ID-Task/  -/Lat/  -/Long/ -/Day/  -/Hour/ -/Minute/   -/Duration/ -/Remaining time/ -/Resources/    -/Coverage/ -/Ligitimacy/ -/on peak hour/ -/grid_number -/task_value\n")
        for i in range(0,len(tasks)):
            task_file.write("{}\t {}\t {}\t {}\t {}\t {}\t {}\t {}\t {}\t {}\t {}\t {}\t {}\t {}\n".format(tasks[i][0], tasks[i][1], tasks[i][2], tasks[i][3], tasks[i][4], tasks[i][5], tasks[i][6], tasks[i][7], tasks[i][8], tasks[i][9], tasks[i][10], tasks[i][11], tasks[i][12], tasks[i][13]))

        task_file.close()

        # task csv file
        csvfile = open('./Inputs/Mobility/differentradius/50/'+str(run_num)+'/Tasks.csv','w')
        with csvfile:
            titles = ['ID','Latitude','Longitude','Day','Hour','Minute','Duration','RemainingTime','Resources','Coverage','Ligitimacy','OnPeakHours','GridNumber','TaskValue']
            writer = csv.DictWriter(csvfile, fieldnames=titles)
            writer.writeheader()
            for i in range(0,len(tasks)):
                # if(tasks[i][3]==0):
                #     continue
                writer.writerow({'ID':tasks[i][0],'Latitude':tasks[i][1],'Longitude':tasks[i][2],'Day':tasks[i][3],'Hour':tasks[i][4],'Minute':tasks[i][5],'Duration':tasks[i][6],'RemainingTime':tasks[i][7],'Resources':tasks[i][8],'Coverage':tasks[i][9],'Ligitimacy':tasks[i][10],'OnPeakHours':tasks[i][11],'GridNumber':tasks[i][12],'TaskValue':tasks[i][13]})
```

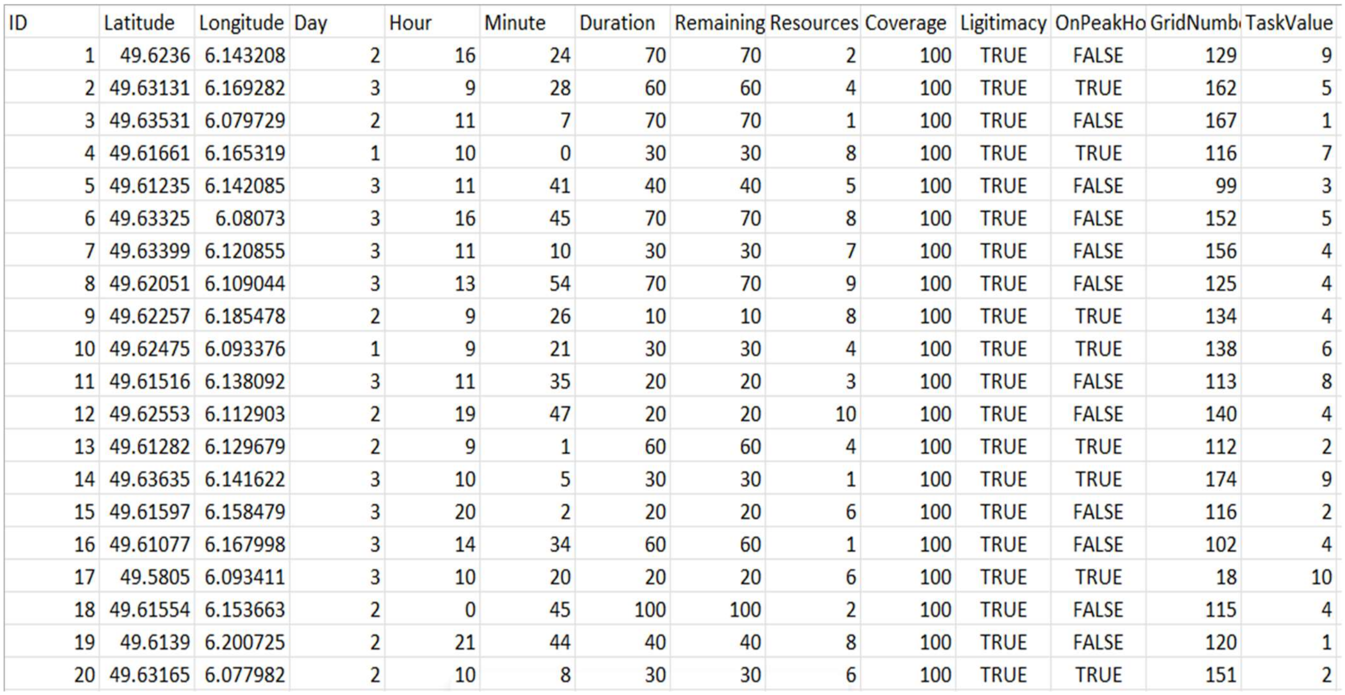
- The task_generator function is modified to satisfy the requirements of tasks. The function after removing unnecessary comments is shown below.

```python
def task_generator(big_graph, num_tasks, days, df, ligi, on_peak, attackLocations, num_att):
    attack_radius = 50
    tl = []
    l = []
    l = list(big_graph.nodes())
    for i in range(num_tasks):
        day = random.randint(1,days)
        hourrand = random.randint(1,100)
        if hourrand <51:
            h=random.randint(9,11)
        elif hourrand<76:
            h=random.randint(12,17)
        else:
            h=random.randint(0,14)
            if(h>8):
                h += 9
        m = random.randint(0,59)

        ligi = True
        durrand = random.randint(1,100)
        if durrand < 51:
            dur = random.randint(1,3)*20
        elif durrand < 81:
            dur = random.randint(1,3)*20 + 10
        else:
             dur = random.randint(1,3)*10
             if dur == 20:
                 dur = 80
             elif dur == 30:
                 dur = 100
        task_value = random.randint(1,10)
        r = random.randint(1,10)
        index = random.choice(range(len(l)))
        y=big_graph.node[l[index]]['y']
        x=big_graph.node[l[index]]['x']
        remaining_t = dur
        grid_num=convert_location(big_graph,y,x)
        if(h in range(7,11)):
            on_peak = True
        else:
            on_peak = False
        tl.append([i+1, float(y), float(x), day, h, m, dur, remaining_t, r,
df,ligi,on_peak,grid_num,task_value])

    return tl
```

A sample of the generated tasks is shown in the following figure:



ID	Latitude	Longitude	Day	Hour	Minute	Duration	Remaining	Resources	Coverage	Ligitimacy	OnPeakHo	GridNumb	TaskValue
1	49.6236	6.143208	2	16	24	70	70	2	100	TRUE	FALSE	129	9
2	49.63131	6.169282	3	9	28	60	60	4	100	TRUE	TRUE	162	5
3	49.63531	6.079729	2	11	7	70	70	1	100	TRUE	FALSE	167	1
4	49.61661	6.165319	1	10	0	30	30	8	100	TRUE	TRUE	116	7
5	49.61235	6.142085	3	11	41	40	40	5	100	TRUE	FALSE	99	3
6	49.63325	6.08073	3	16	45	70	70	8	100	TRUE	FALSE	152	5
7	49.63399	6.120855	3	11	10	30	30	7	100	TRUE	FALSE	156	4
8	49.62051	6.109044	3	13	54	70	70	9	100	TRUE	FALSE	125	4
9	49.62257	6.185478	2	9	26	10	10	8	100	TRUE	TRUE	134	4
10	49.62475	6.093376	1	9	21	30	30	4	100	TRUE	TRUE	138	6
11	49.61516	6.138092	3	11	35	20	20	3	100	TRUE	FALSE	113	8
12	49.62553	6.112903	2	19	47	20	20	10	100	TRUE	FALSE	140	4
13	49.61282	6.129679	2	9	1	60	60	4	100	TRUE	TRUE	112	2
14	49.63635	6.141622	3	10	5	30	30	1	100	TRUE	TRUE	174	9
15	49.61597	6.158479	3	20	2	20	20	6	100	TRUE	FALSE	116	2
16	49.61077	6.167998	3	14	34	60	60	1	100	TRUE	FALSE	102	4
17	49.5805	6.093411	3	10	20	20	20	6	100	TRUE	TRUE	18	10
18	49.61554	6.153663	2	0	45	100	100	2	100	TRUE	FALSE	115	4
19	49.6139	6.200725	2	21	44	40	40	8	100	TRUE	FALSE	120	1
20	49.63165	6.077982	2	10	8	30	30	6	100	TRUE	TRUE	151	2

Figure 1 A sample from the generated tasks

Excel is used to check if the generated tasks are satisfying the requirements or not.

2. Obtaining user movement event:

1. Uniform distribution

Choose the last value greater than cut

```
for l in length:
  if length[l]>cut:
    idr=l
```

If not find values greater than the cut choose a larger value less than cut

```
    if idr==0:
        idr=max(length, key=length.get)
```

2. stochastic algorithm

Using length and path generated by Dijkstra function to get all paths from origin node they less than cutadded

```
(length, path)= nx.single_source_dijkstra(G_old, origin_node,target=None, cutoff=cutadded, weight='length')
```

Now get all paths they are greater than cut and put them in newpath dictionary

```
newpath={}
for l in length:
    if length[l]>cut:
        newpath[l]=length[l]
```

Generate random probability based on the length of each pass longer paths have more probability than shorter paths

```
weight_probs = np.random.dirichlet(list(newpath.values()))
```

choose path stochastically but give more probability based on the weight that gets in the previous step

```
idr=np.random.choice(list(newpath.keys()),p=weight_probs)
```

Get a path that chooses based on a stochastic algorithm

```
route=path[idr]
```