IJARCCE

ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 8, August 2021

DOI 10.17148/IJARCCE.2021.10810

Cov-INS | Intelligent Navigation System to Avoid Infected Covid-19 Areas with Reinforcement Learning and Internet of Things

[1]Sudip Mitra, [2]Shree Sarkar

[1]Department of Computer Science and Engineering, Government College of Engineering and Leather Technology, Kolkata, India

[2]Department of Computer Science and Engineering, Budge Budge Institute of Technology, Kolkata, India

Abstract: In this paper, a Machine Learning-enabled intelligent navigation system is presented. It will recommend routes in a road network by minimizing source to destination distance by choosing right shortest path between source to destination , it also take care and avoids categorically marked COVID-19 hotspots. The Q-Learning based system takes the source and destination as inputs from the users and recommends a safe and shorter path for traveling. It reduces the risk of getting exposed to the contaminated zones and contracting the virus by bypassing the red covid19 hotspot zones.

Keywords: Reinforcement Learning, IoT, Intelligent Navigation System, Route Planning, Q-Learning, Covid19 Hotspot

I. INTRODUCTION

Coronavirus disease (COVID-19) is an infectious disease caused by a newly discovered coronavirus. Most people infected with the COVID-19 virus will experience mild to moderate respiratory illness and recover without requiring special treatment. Coronaviruses belong to the Coronaviridae family in the Nidovirales order. Corona represents crown-like spikes on the outer surface of the virus; thus, it was named as a coronavirus. Coronaviruses are minute in size (65–125 nm in diameter) and contain a single-stranded RNA as a nucleic material, size ranging from 26 to 32kbs in length. The small size and intangible nature of the virus have led to an uncontrollable spread across the world, resulting in creation of COVID-19 hotspots.

The best way to prevent and slow down transmission is to be well informed about the COVID-19 virus, the disease it causes and how it spreads. Protect yourself and others from infection by washing your hands using an alcohol based rub frequently, not touching your face and maintaining physical distance. Since contact tracing in these covid19 hotspot areas is challenging, the concerned authorities identify these hotspots and categorize them as red, orange, or green zones. These zones are classified based on the severity of the spread, and the restrictions in each zone vary accordingly.

Hotspot areas classified as Red Zone as it needs to have focused attention in these areas reporting a large number of Covid-19 cases and high growth rate.

The areas with a limited number of cases in the past and with no surge in positive cases recently would be included under the orange zone. For the Green Zone, the area that has not reported positive coronavirus cases can be marked under this zone.

Although the conditions are adverse, people need to commute from one place to the other regularly for work and basic amenities. Safe and Intelligent Transportation is a concern at this moment, and avoidance of hotspots is of paramount importance.

Here, shortest path and optimized path algorithms do not suffice in finding safe routes in COVID-19 environments. In such situations, intelligent system that recommend routes [1] that bypass hotspots or minimize passage through them is necessary for ensuring the safety of the people.

In this Reinforcement Learning-based intelligent navigation system avoids COVID- 19 hotspots according to the category of the zones for ensuring the safety of the users. It takes Source and Destination as inputs from the users and recommends a safe path that is optimal route and minimizes traveling through the hotspot areas.

A. Uniqueness of Intelligent Navigation System

Some of the most commonly used applications are Google Maps, Apple Maps, HERE WeGo Maps, and others. Apart from their uniqueness in features, such as online/offline, personalization, and visualization, these applications typically focus on road traffic, blockage, speed, and distance for finding the shortest/optimized route from source to destination. Such methods are not suitable for use in the current scenario of COVID-19. The commuters need to avoid

Vol. 10, Issue 8, August 2021

DOI 10.17148/IJARCCE.2021.10810

traveling through hotspots to reduce exposure. Our algorithm recommends paths that bypass traveling through hotspots. In unavoidable situations, it ensures minimal travel through them.

B. Related Work

Navigation and path planning has been an area of great interest among researchers.

Zhou and Wang [2] designed a routing model by considering intersection signals and real-time velocity of the vehicle. Szucs [3] developed a model based on Dempster–Shafer theory to calculate the uncertainty (road conditions) in cost function while using Dijkstra's algorithm. Chen et al. [4] proposed a log C-means clustering algorithm (LFCM) to form clusters based on driving style. Then, they used the ant colony algorithm to calculate the shortest path. Sun et al. [5] proposed a graph-based method to minimize travel time by either recommending the shortest path or the best starting time using conventional and extended Bellman–Ford algorithms.

Lv et al. [6] demonstrated how IoT systems have the potential to monitor these routes and vehicles by using off-the-shelf sensors for communicating with street lights and other stationary units installed on the street. Xu et al. [7] proposed a crowd evacuation recommender system in case of disasters. Zhu et al. [8] exploited the features of Q-learning methods to schedule transmissions to ensure reliable exchange of data.

The existing solution techniques typically focus on finding the shortest/optimal paths between the source and destinations. Such methods are not suitable for use in the current scenario due to the threat of the COVID-19 virus. The commuters need to maintain social distancing and avoid traveling through zones that may be part of the optimized routes to reduce exposure.

II. INTELLIGENT NAVIGATION MODEL

A. Why Q- Learning is used to build the model?

The COVID-19 virus spreads primarily through droplets of saliva or discharge from the nose when an infected person coughs or sneezes, so virus is spreading rapidly and it got out of control. Due to this, the status of the hotspots keeps changing at specific intervals. The criteria for road routes need to change per the changing states of the hotspots.

Q-learning is an off-policy value-based Reinforcement Learning technique that does not depend on the conventional greedy methods and estimates its reward for the future. In summary, because of the feedback loop, Q-learning, changing status of the hotspots, and nongreedy selection technique, we use Reinforcement Learning instead of other possible solution techniques.

B. Analysing Routes and Generation of Graph

For a geographic region, we extract the road information and form a road network graph. We represent it as $G = < V, E >$ such that V is the set of vertices representing the intersection points, and E is the set of edges representing the roads. For the set of vertices $V = \{v1, v2,..., va\}$ and corresponding edges $E = \{e1, e2,..., eb\}$, the final vertex–edge pair representing the road route is $< V*, E* > \subseteq < V, E >$. The $< V*, E* >$ pair is user centric as it depends on the user's source and destination.

C. Intelligent Safety-aware Navigation Algorithm

We obtain G from the maps to generate a reward matrix and process it on C or F to produce the safety-aware route. For $G = < V, E >$, we represent the length of the ith edge as $e^{dist}i= l_i$ and the set of all distances as $L = \{l_1, l_2,..., l_c\}$.

It operates on the roads from G and the corresponding R matrix. The rewards for each edge in take care of assigning higher rewards to low containment factors and distances. The low containment factors ensure taking routes through low-risk regions. The distance parameter ensures minimal traversal through the containment zones if there is no safer alternative. Upon training the Q matrix based on these rewards, it considers the path that renders the maximum reward. For a road e^i, its containment factor is α^i and the set of all containment factors is $\alpha = \{\alpha^1, \alpha^2, \alpha^3,...,\alpha^b\}$ such that $|E|=|\alpha|$. For instance, the government has divided the regions into zones according to three categories: 1) red zone (α^r): regions that have a large number of COVID-19 positive cases; 2) orange zone (α^o): regions with a relatively fewer number of COVID-19 cases compared to the red zones; and 3) green zone (α^g): regions with no reported COVID-19 cases. In this work, we assign low α values to edges belonging to zones with lower severity, implying that $\alpha^g < \alpha^o < \alpha^r$. In context of the mentioned parameters, we formulate the reward of a path based on two major components—1) path length or distance and 2) intensity of containment.

ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 8, August 2021

DOI 10.17148/IJARCCE.2021.10810

```
Input: epochs = iterations ;        // Number of
iterations set according to number of
nodes
Result: All the possible paths are discovered and
        knowledge of the best path is attained by the
        trail and error approach.
for epochs do
    Select $c_s$ randomly from Q-matrix;
    Select $n_s$ from available states for $c_s$;
    Update the Q[$c_s$, $n_s$];
    // Intelligent Navigation System for Covid19
end
```

Algorithm: Intelligent Navigation Model Training

We used a random distribution process to allocate the categorical hotspot zones in the network map. We used Google Colab to assume the role of cloud and the device mentioned earlier. We scale our results here and presented them in its original form.

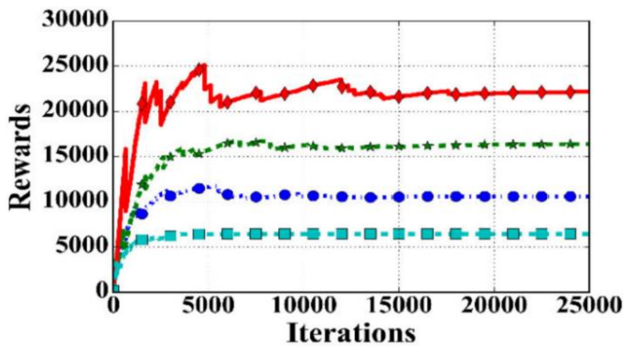


Fig 1: Reward values while training the Intelligent Navigation model with varying number of nodes in the map.

While training the model for Intelligent Navigation System, we fix the number of epochs/iterations sufficiently high to ensure that the model accurately explores the environment. We draw inferences, and safety-aware paths after the training is complete.

III. EXPERIMENTAL SETUP

We performed a series of experiments to evaluate the performance of our Intelligent Navigation Model. Toward this, we, used the road-network data set exported from OpenStreetMap and processed the data in a device with 1.8-GHz Dual-Core Intel Core i5 processor with Q Reinforcement Learning. We used different sets of longitudes and latitudes to vary the number of nodes and edges. We used a random distribution process to allocate the categorical hotspot zones in the network map.

Here we used osmnx 1.1.0, OSMnx is a Python package that lets us download spatial data and model, project, visualize, and analyze real-world street networks from OpenStreetMap's APIs. we can download and model walkable, drivable, or bikeable urban networks with a single line of Python code, and then easily analyze and visualize them. With OSMnx we can just as easily download and work with amenities/points of interest, building footprints, elevation data, street bearings/orientations, speed/travel time, and network routing.

```
#Install Required Libraries

!apt-get -qq install -y libspatialindex-dev

!pip install -q -U osmnx

!pip install matplotlib = = 3.4.1
```

We also used matplotlib 3.4.1, Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations. Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits.

Let us bound the range of the map: north, east, south, west = 22.5862, 88.3785, 22.5764, 88.3645. And decide node numbers of Orange and Red Zone node:

Orange = [1177655232, 1176976769, 663846649, 1202746627, 1176978122, 1202746673, 1177816130, 1195827529, 1177815769, 1194743066, 663940665, 1202168183, 1202168279]

Red = [1217614532, 663847554, 2281800874, 1177815903, 1217614604, 1217614501, 121761790, 1202332476, 1217614545, 1217614635, 1217614499, 1217614509, 663843644]

Vol. 10, Issue 8, August 2021
DOI 10.17148/IJARCCE.2021.10810

Let us assign the source node and destination node number:

```
initial_state = 21  #assign source node number
goal = 41                    #assign destination node number
```

```
import numpy as np
import pylab as plt
import networkx as nx
import time
import osmnx as ox
import matplotlib.pyplot as plt ox.config(use_cache=True,
log_console=True)
```

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NetworkX is a Python library for studying graphs & networks.

OSMnx is a Python package to retrieve, model, analyse, and visualize street networks from OpenStreetMap.

Matplotlib is a plotting library for creating static, animated, and interactive visualizations in Python.

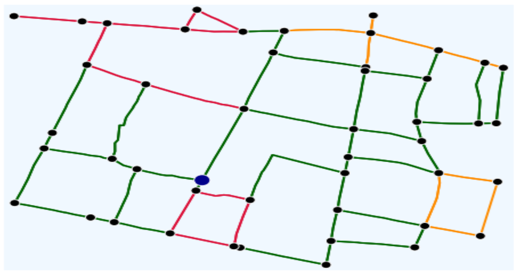
```
# Defining the map boundaries
north, east, south, west = 22.5862, 88.3785, 22.5764,
88.3645
# Downloading the map as a graph object
G = ox.graph_from_bbox(north, south, east, west,
network_type = 'drive')

red = [1177655232, 1176976769, 663846649, 1202746627,
1176978122, 1202746673, 1177816130, 1195827529,
1177815769, 1194743066, 663940665, 1202168183,
1202168279]  #assign node numbers of edges in red zone

orange = [1217614532, 663847554, 2281800874,
1177815903, 1217614604, 1217614501, 121761790,
1202332476, 1217614545, 1217614635, 1217614499,
1217614509, 663843644]     #assign node numbers of edges
in orange zone
```

```
nodes = []
 node_color = []
node_size = []
initial_state =  21          #assign source node number
goal = 41                    #assign destination node number

for i in range(len(list(G.nodes(data = True)))):
    temp = list(G.nodes(data = True))[i]
    nodes.append(temp[0])
    print(i," - ",temp[0])
    if i == initial_state:
        node_color.append('darkblue')
        node_size.append(200)
    else:
        node_color.append('k')
        node_size.append(80)edges = []

edge_color = []
max_path_len = 0
min_path_len = 1000
avg_path_len = 0
```

```
for i in range(len(list(G.edges(data = True)))):
    temp1 = list(G.edges(data = True))[i]
    length = temp1[2]['length']

    if min_path_len>length:
        min_path_len = length
    if max_path_len<length:
        max_path_len = length
    avg_path_len += length

    alpha = 0;
    node1 = temp1[0]
    node2 = temp1[1]
    if node1 in red and node2 in red:
        edge_color.append('crimson')
        alpha = 0.3 # assign contamination factor for red
    elif node1 in orange and node2 in orange:
        edge_color.append('darkorange')
        alpha = 0.2  # assign contamination factor for orange
    else:
        edge_color.append('darkgreen')
        alpha = 0.1  # assign contamination factor for green
    temp2 = (nodes.index(node1), nodes.index(node2), length,
alpha)
    edges.append(temp2)
```

```
avg_path_len = avg_path_len/len(edges)
print("average path length : ",avg_path_len)
print("maximum path length : ",max_path_len)
print("minimum path length : ",min_path_len)

# Plotting the map graph
# ox.plot_graph(G,node_zorder=3,annotate =
True,node_edgecolor = 'w',node_color = node_color, node_size
= node_size, edge_color = edge_color, edge_linewidth=2,
use_geom = True, bgcolor = "aliceblue")

ox.plot_graph(G,node_zorder=3,node_edgecolor =
'w',node_color = node_color, node_size = node_size,
edge_color = edge_color, edge_linewidth=2, bgcolor =
```

0 - 663843644	16 - 1177816130	32 - 1217614499
1 - 663846649	17 - 1194742715	33 - 1217614501
2 - 663847548	18 - 1194742977	34 - 1217614509
3 - 663847554	19 - 1194743014	35 - 1217614510
4 - 663940656	20 - 1194743066	36 - 1217614532
5 - 663940665	21 - 1194743076	37 - 1217614545
6 - 663940666	22 - 1195827529	38 - 1217614574
7 - 664446482	23 - 1196059624	39 - 1217614579
8 - 1176976769	24 - 1200782099	40 - 1217614580
9 - 1176978122	25 - 1202168057	41 - 1217614581
10 - 1177655232	26 - 1202168183	42 - 1217614604
11 - 1177815213	27 - 1202168279	43 - 1217614635
12 - 1177815361	28 - 1202332476	44 - 1217614783
13 - 1177815769	29 - 1202746627	45 - 1217614790
14 - 1177815854	30 - 1202746673	46 - 2281800874
15 - 1177815903	31 - 1217614493	



```
average path length :  176.0204108527131
maximum path length :  416.677
minimum path length :  15.602

(<Figure size 576x576 with 1 Axes>,
<matplotlib.axes._subplots.AxesSubplot at 0x7f4284b36050>)
```

We keep track of the rewards while training the Intelligent Navigation Model and present the results. We vary the number of nodes on the map and start training. We observe convergence in each case. Additionally, as we increase the number of nodes, the value of the rewards keeps increasing.

We measure the reward values by the sum of all the entries in the Q-matrix of the model. We notice the jumps and downs as the model's training phase uses the trial and error approach. It learns from its mistakes and gains knowledge of the safer path. So, when the number of nodes is more, the number of edges is usually more. The model needs to explore all the possibilities and requires more number of iterations.

```
MATRIX_SIZE = len(nodes)
print(MATRIX_SIZE)

# create matrix x*y
R = np.matrix(np.ones(shape=(MATRIX_SIZE,
MATRIX_SIZE)))
R *= -100

def sigmoid(x):
   return 1/(1+np.exp(-x))

gamma = 0.5            #assign gamma value

for point in edges:
   if point[1] == goal:
      R[point[0],point[1]] = 1800
   else:
      R[point[0],point[1]] =
max_path_len*point[3]*sigmoid((min_path_len-
point[2])/avg_path_len)

   if point[0] == goal:
      R[point[1],point[0]] = 1800
   else:
      R[point[1],point[0]] =
max_path_len*point[3]*sigmoid((min_path_len-
point[2])/avg_path_len)

R[goal,goal] = 1800
```

```
def available_actions(state):
   current_state_row = R[state,]
   av_act =np.where(current_state_row >=0)[1]
   return av_act

def sample_next_action(available_actions_range,state):
   next_action = int(np.random.choice(available_act,1))
   return next_action

def update(current_state, action):
   max_index = np.where(Q[action,] ==
np.max(Q[action,]))[1]

   if max_index.shape[0] > 1:
      max_index = int(np.random.choice(max_index, size =
1))
   else:
      max_index = int(max_index)
   max_value = Q[action, max_index]

   Q[current_state,action] = R[current_state, action] +
gamma * max_value

   if (np.max(Q) > 0):
      return(np.sum(Q/np.max(Q)*100))
```

ISSN (O) 2278-1021, ISSN (P) 2319-5940

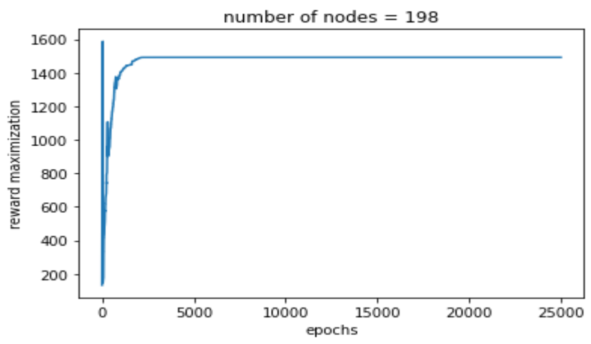
International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 8, August 2021

DOI 10.17148/IJARCCE.2021.10810

```
MATRIX_SIZE = len(nodes)
print(MATRIX_SIZE)
Q = np.matrix(np.zeros([MATRIX_SIZE,MATRIX_SIZE]))

available_act = available_actions(initial_state)
action = sample_next_action(available_act,initial_state)
update(initial_state, action)
scores = []
for k in range(25000):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_act = available_actions(current_state)
    action = sample_next_action(available_act,current_state)
    score = update(current_state,action)
    scores.append(score)

plt.xlabel("epochs")
plt.ylabel("reward maximization")
plt.title("number of nodes = 198")
plt.plot(scores)
plt.show()
```

O/P 47



```
current_state = initial_state
steps = [current_state]
prev = current_state;

def avail(state):
    ttt = []
    for y in range(MATRIX_SIZE):
        if Q[state,y] >0:
            ttt.append(y)
        return ttt

ach = 0
while current_state != goal:
    indices = avail(current_state)
    next_step = []
    for ee in range(len(indices)):
        if indices[ee]==goal:
            ach = 10
            next_step.append((Q[current_state,indices[ee]],
indices[ee]))
            break
        elif indices[ee] not in steps:
            next_step.append((Q[current_state,indices[ee]],
indices[ee]))

    if ach!=10:
        next_step.sort()
```

Commonly available applications, such as Google Maps rely on Dijkstra's algorithm and recommend shortest paths based on parameters. Without loss of generality, we refer to the available methods as shortest path algorithms. Although the Intelligent Navigation model finds the safest route possible, it increases the travel distance, implying a trade-off between safety and distance. On setting arbitrary source–destination pairs, we study the increase in travel distance.

We observe higher travel distances by our model in all cases. On average, we see almost 15% additional travel distance in the case of Intelligent Navigation Model as compared to the shortest path. However, we analyse the travel distance through hotspots (red/orange/green) and present the percentage with respect to each case's total recommended path. In both red and orange zones, we observe that our model takes paths as small as 2% (15 m). The shortest path algorithm takes 65% on the same source–destination pair. In the case of green zones, we observe that our model travels mostly through them compared to the shortest path model.

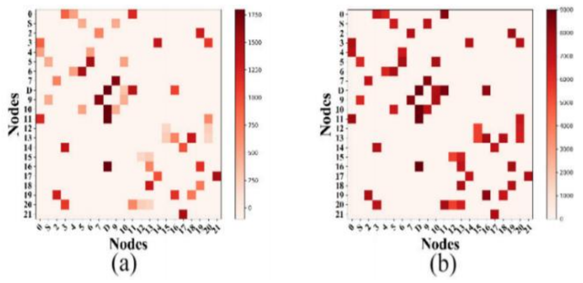


Fig 2: R and Q matrices while training the Intelligent Navigation Model

We consider a map with 22 nodes and 55 edges. We demarcate some areas as red and orange zones to show the confusion matrices. We set the rewards according to (1) and populate the R matrix of size 22×22. R is a symmetric matrix as we consider an undirected map. The destination node has the highest reward, and hence we observe dark patches in Fig. 2(a). The lightest shade, which covers the majority of the matrix, represents non existing edges among the nodes. The other shades represent the rewards for each edge. The Intelligent Recognition System trains the model and populates the Q matrix in Fig. 2(b). The patch with the darkest shade in the row represents the best possible next state from the current state (row index). The

```
if len(next_step)==0:
        print('path not possible')
        steps = nx.shortest_path(G,nodes[1],nodes[goal],weight
= 'length')
        for p in range(len(steps)):
            steps[p] = nodes.index(steps[p])
        break

    next_step_index = next_step[len(next_step)-1][1]
    steps.append(next_step_index)
    prev = current_state
    current_state = next_step_index
```

ISSN (O) 2278-1021, ISSN (P) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 10, Issue 8, August 2021
DOI 10.17148/IJARCCE.2021.10810

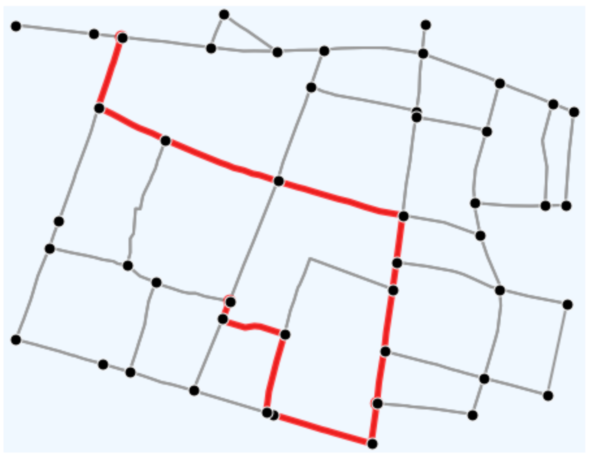
values of the Q-matrix updates according to the Bellman equation.

We safely comment that our model ensures safe to travel in all source–destination pairs. However, it increases the travel distance, which is not a concerning factor as it reduces the risk of contracting the virus.

```
route1 = []
for i in range(len(steps)):
    route1.append(nodes[steps[i]])
print(steps)
print(route1)

steps2=[]
route2 =
nx.shortest_path(G,nodes[initial_state],nodes[goal],weight =
'length')
for i in range(len(route2)):
    steps2.append(nodes.index(route2[i]))

print(steps2)
print(route2)

c1 = 'green'     #assign colour to our model
c2 = 'magenta'      #assign colour to Shortest
rc1 = [c1] * (len(route1) - 1)
rc2 = [c2] * (len(route2) - 1)
rc = rc1 + rc2
nc = [c1, c1, c2, c2]

# plot the routes
# fig, ax = ox.plot_graph_routes(G, [route1,route2],
route_color=rc, route_linewidth=5,route_alpha=
0.8,edge_linewidth = 2 ,node_color = 'k',node_zorder = 3,
# orig_dest_node_color='maroon',
node_edgecolor='w',orig_dest_node_alpha = 0.8,
node_size=80,orig_dest_node_size=200,bgcolor='aliceblue')

fig, ax = ox.plot_graph_routes(G, [route1,route2],
route_color=rc, route_linewidth=5,route_alpha=
0.8,edge_linewidth = 2 ,node_color = 'k',node_zorder = 3,
node_edgecolor='w', node_size=80,bgcolor='aliceblue')
```

```
[1, 16, 22, 13, 2, 31, 25, 38, 41]

[663846649, 1177816130, 1195827529, 1177815769,
663847548, 1217614493, 1202168057, 1217614574,
1217614581]
[21, 20, 27, 26, 24, 6, 41]
[1194743076, 1194743066, 1202168279,
1202168183, 1200782099, 663940666, 1217614581]
```

Final Directed Path by Avoiding Covid19 Hotspots:



IV. NETWORK ARCHITECTURE WITH IOT

We consider an IoT-based network architecture, as shown in Fig. 3 for realizing Intelligent Navigation System. In this work, we consider a scenario with a remote cloud server C and a set of Fog Layer Nodes F = {f_1, f_2,..., f_n}. The cloud is responsible for computing the paths for a wide area (cities/states) and for areas that are devoid of Fog Nodes. On the other hand, we propose the use of location-aware Fog Nodes for Intelligent Navigation System. In other words, for a set of geographical regions G = {g_1, g_2,..., g_m} each of the Fog Nodes is responsible for their own geographical area. This strategy reduces the size of the map on the Fog Nodes, which is suitable for the resource-constrained nature of the Fog Nodes. It may be noted that we consider networking devices, such as switches, routers, and others for assuming the role of Fog Nodes.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 8, August 2021

DOI 10.17148/IJARCCE.2021.10810

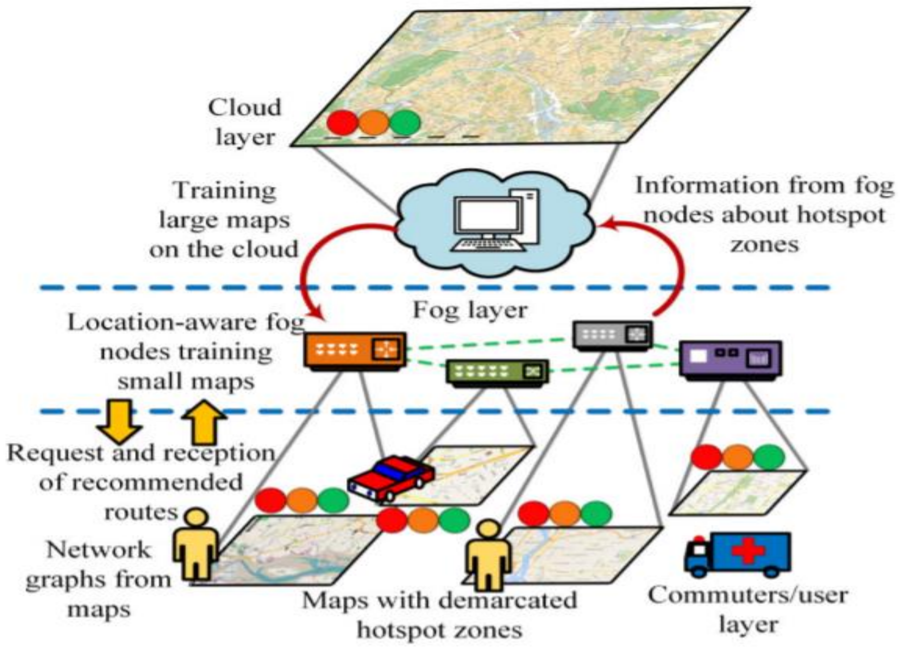

Cloud layer
Training large maps on the cloud
Information from fog nodes about hotspot zones
Fog layer
Location-aware fog nodes training small maps
Request and reception of recommended routes
Network graphs from maps
Maps with demarcated hotspot zones
Commuters/user layer

Fig 3: Network Architecture for Intelligent Navigation System

As Cloud Server tries to train for a large region, and the FNs feed granular details with respect to the hotspots, the separation of the tasks among the fog and cloud helps distribute the load along with easy data management.

It may be noted that the red zones are converted to orange when there are no COVID-19 positive cases for 14 continuous days, and its conversion to the green zone needs at least 28 continuous days with no positive reports. In this work, we depend on some user intervention in this regard and allow only concerned authorities to update the databases.

V. CONCLUSION

In this article, we proposed and developed an Reinforcement Learning based intelligent navigation model that recommends road routes with ensured safety from interaction with covid19 viruses. The recommended path avoids traveling categorical hotspots, ensuring safety, and reducing exposure risk for the users. To facilitate real-time results, we proposed an IoT-based network architecture. We formulate the reward function for the reinforcement learning model by imposing zone-based penalties and demonstrate that Intelligent Navigation System achieves convergence under all conditions. To ensure real-time results, we propose an Internet of Things (IoT) based architecture by incorporating the cloud and fog computing paradigms. The fog computing platform allows partitioning of the maps and operations on small portions rather than the entire map, which is time-consuming. We also performed extensive experiments on Intelligent Navigation System using real data sets from OpenStreetMap, Google Maps and presented results. We also performed a detailed comparative analysis of the recommended paths by Intelligent Navigation System with Dijkstra’s shortest path algorithm. We observed an average 14% increase in travel distances by our navigation system. Here we also described the total procedure of Q-Learning with Python step by step. We plan to extend this work by considering additional factors, such as real-time traffic, multilane road system, and speed control.

REFERENCES

[1] G. Andonov and B. Yang, “Stochastic shortest path finding in pathcentric uncertain road networks,” in Proc. 19th IEEE Int. Conf. Mobile Data Manag. (MDM), Jul. 2018, pp. 40–45.
[2] W. Zhou and L. Wang, “The energy-efficient dynamic route planning for electric vehicles,” J. Adv. Transp., vol. 2019, pp. 1–16, Aug. 2019
[3] I G. Szucs, “Decision support for route search and optimum finding in transport networks under uncertainty,” J. Appl. Res. Technol., vol. 13, pp. 125–134, Feb. 2015
[4] P. Chen, X. Zhang, X. Chen, and M. Liu, “Path planning strategy for vehicle navigation based on user habits,” Appl. Sci., vol. 8, p. 407, Mar. 2018
[5] Y. Sun, X. Yu, R. Bie, and H. Song, “Discovering time-dependent shortest path on traffic graph for drivers towards green driving,” J. Netw. Comput. Appl., vol. 83, pp. 204–212, Apr. 2017.
[6] Z. Lv, B. Hu, and H. Lv, “Infrastructure monitoring and operation for smart cities based on IoT system,” IEEE Trans. Ind. Informat., vol. 16, no. 3, pp. 1957–1962, Mar. 2020
[7] X. Xu, L. Zhang, S. Sotiriadis, E. Asimakopoulou, M. Li, and N. Bessis, “CLOTHO: A large-scale Internet of Things-based crowd evacuation planning system for disaster management,” IEEE Internet Things J., vol. 5, no. 5, pp. 3559–3568, Oct. 2018.
[8] J. Zhu, Y. Song, D. Jiang, and H. Song, “A new deep-Q-learning-based transmission scheduling mechanism for the cognitive Internet of Things,” IEEE Internet Things J., vol. 5, no. 4, pp. 2375–2385, Aug. 2018

BIOGRAPHY

Sudip Mitra is a Bachelor of Technology final year candidate in the Department of Computer Science and Engineering at Government College of Engineering and Leather Technology, Kolkata. He received Diploma in Engineering in Computer Science and Technology from Budge Budge Institute of Technology, Kolkata at 2019. He has been recognized as Google Information Technology Support Professional at 2020. He worked in two product based companies as a Software Developer and Research & Mobile App Developer. His primary research interest in Systems, Networks and Artificial Intelligence. He is the member of National Society of Professional Engineers (US) since 2021.

Shree Sarkar is a Bachelor of Technology final year candidate in the Department of Computer Science and Engineering at Budge Budge Institute of Technology, Kolkata. She received Diploma in Engineering in Computer Science and Technology from Budge Budge Institute of Technology, WB at 2019.