main

March 1, 2023

[ ]:
```
import pandas as pd
import traci
import gym
```

[ ]:
```
order_path = "../data/order.csv"
order = pd.read_csv(order_path,index_col = 0 )
order.columns =␣
 ↪["useless","order_id","arrive_time","departure_time","dest_lng","dest_lat","starting_lng","
order = order.drop(order[order.arrive_time == "0000-00-00 00:00:00"].index)
order.arrive_time = pd.to_datetime(order.arrive_time)
order.departure_time = pd.to_datetime(order.departure_time)
order["start_time"] = order.departure_time.min()
order["relative_time"] = order.apply(lambda x:x.departure_time - x.
 ↪start_time,axis = 1)
order["relative_seconds"] = order.relative_time.dt.total_seconds()
order = order[order.relative_seconds >=0]
order = order[(order.arrive_time - order.departure_time).dt.total_seconds() >=0]
order = order.sort_values("relative_seconds")
order.head(2)
```

[ ]:
```
       useless        order_id         arrive_time      departure_time  \
37166   459617  17592360594114 2017-05-01 08:00:19 2017-05-01 08:00:00
39560   462011  17592363527038 2017-05-01 08:00:44 2017-05-01 08:00:00

       dest_lng  dest_lat  starting_lng  starting_lat  year  month  day  \
37166  110.3433   19.9836      110.3743       20.0081  2017      5    1
39560  110.3533   19.9786      110.2897       20.0127  2017      5    1

              start_time relative_time  relative_seconds
37166 2017-05-01 08:00:00        0 days               0.0
39560 2017-05-01 08:00:00        0 days               0.0
```

[ ]:
```
env_args = {
    "driver_num" : 20,                    #
    "order_path" : '../data/order.csv',     #
    "GUI": True,
    "sumo_path" : 'C:/Program Files (x86)/Eclipse/Sumo/bin',  #
    "sumocfg_path" : '../network/haikou.sumocfg',
```

```
    "time_interval":10,
    "distance_threshold":100,
    'seed':42,
    "total_timesteps":1800,
    "delay": 0.01,

}
```

[ ]:
```
import os
import random
from tqdm.notebook import tqdm
import pandas as pd
import numpy as np
import traci
import time
import sumolib

COLOR_DICT = {
    "white":(255, 255, 255, 255),    #
    "yellow":(255, 255, 0, 255),     #      0 /
    "orange":(255, 165, 0, 255),     #         1 /        1 /        1
    "red":(255, 0, 0, 255),          #        2
}

class SUMO_env(gym.Env):
    def __init__(self,args):
        self.driver_num = env_args["driver_num"]
        self.time_interval = env_args["time_interval"]
        self.distance_threshold = env_args["distance_threshold"]
        self.seed_ = env_args["seed"]
        self.total_timesteps = env_args["total_timesteps"]
        self.delay = env_args["delay"]

        #   order
        order_path = env_args["order_path"]
        self.order = self.process_order(order_path)

        #
        if env_args["GUI"]:
            self.sumo_path = os.path.join(env_args["sumo_path"],"sumo-gui")
        else:
            self.sumo_path = os.path.join(env_args["sumo_path"],"sumo")
        self.sumocfg_path = env_args["sumocfg_path"]

    def process_order(self,order_path):
        order = pd.read_csv(order_path,index_col = 0 )
```

```
        order.columns =␣
↪["useless","order_id","arrive_time","departure_time","dest_lng","dest_lat","starting_lng","
        order = order.drop(order[order.arrive_time == "0000-00-00 00:00:00"].
↪index)
        order.arrive_time = pd.to_datetime(order.arrive_time)
        order.departure_time = pd.to_datetime(order.departure_time)
        order["start_time"] = order.departure_time.min()
        order["relative_time"] = order.apply(lambda x:x.departure_time - x.
↪start_time,axis = 1)
        order["relative_seconds"] = order.relative_time.dt.total_seconds()
        order = order[order.relative_seconds >=0]
        order = order[(order.arrive_time - order.departure_time).dt.
↪total_seconds() >=0]
        order = order.sort_values("relative_seconds")

        return order

    def step(self,action):
        '''

        waiting -> pickup_p1 -> fail_carpooling,deliver_p1 -> trip_end ->␣
↪waiting

        waiting -> pickup_p1 -> pickup_p2 -> deliver_p1 -> deliver_p2 ->␣
↪trip_end -> waiting
        '''
        time.sleep(self.delay)

        #   timestep
        self.time += 1
        self.pbar.update(1)

        #
        self.terminate = self.time == self.total_timesteps
        success_d_p,us_drivers,us_passengers = action

        # process pickuping passengers
        for driver in self.drivers.keys():
            passenger_list, posi,edge,driver_condition,next_posi = self.
↪drivers[driver]

            if driver_condition == "pickup_p1":
                #
                passenger = passenger_list[0]
                dis = self.cal_driver_passenger_distance(driver,passenger)
```

```
                #
                if dis < self.distance_threshold:
                    if len(next_posi) >0:   #       pickup_p1 -> pickup_p2
                        # log
                        self.log(driver,"pickup_p2","pickup_p1", p1 = passenger)

                        #
                        self.arrange_route(driver,next_posi)

                        #
                        traci.vehicle.setColor(str(driver),COLOR_DICT["orange"])
                        driver_condition = "pickup_p2"
                        next_posi = ()

                        #
                        traci.person.remove(str(passenger))

                    else:     # pickup_p1 -> fail_carpooling.deliver_p1
                        # log
                        self.log(driver,"fail_carpooling.
↪deliver_p1","pickup_p1",p1 = passenger)

                        #
                        tmp_order = self.order[self.order.order_id == passenger]
                        x1,y1 = tmp_order.dest_lng.item(),tmp_order.dest_lat.
↪item()


                        #
                        self.arrange_route(driver,(x1,y1))

                        #
                        traci.vehicle.setColor(str(driver),COLOR_DICT["yellow"])
                        driver_condition = "fail_carpooling.deliver_p1"
                        next_posi = ()

                        #
                        traci.person.remove(str(passenger))

            elif driver_condition == "fail_carpooling.deliver_p1":
                #
                passenger = passenger_list[0]
                dis = self.cal_driver_destination_distance(driver,passenger)

                #
                if dis < self.distance_threshold:   #     fail_carpooling.
↪deliver_p1 -> waiting
                    # log
```

```
                    self.log(driver,"waiting","fail_carpooling.deliver_p1")

                    #
                    next_posi = self.random_destination(driver)

                    #
                    traci.vehicle.setColor(str(driver),COLOR_DICT["white"])
                    passenger_list = []
                    driver_condition = "waiting"

                    #
                    driver_list, posi, passenger_condition = self.
↪passengers[passenger]
                    passenger_condition = "finish"
                    self.passengers[passenger] = (driver_list, posi,␣
↪passenger_condition)

            elif driver_condition == "waiting":
                if len(next_posi) >0:   #
                    #
                    dis = self.cal_driver_coordinate_distance(driver,next_posi)

                    if dis < self.distance_threshold:   #     waiting -> waiting
                        # log
                        self.log(driver,"waiting","waiting")

                        #
                        next_posi = self.random_destination(driver)

                        #
                        driver_condition = "waiting"
                else:
                    raise ValueError(f"{driver} - Random trip must have a␣
↪destination")

            elif driver_condition == "pickup_p2":
                #
                passenger = passenger_list[1]
                dis = self.cal_driver_passenger_distance(driver,passenger)


                #
                if dis < self.distance_threshold:   # pickup_p2 -> deliver_p1 /␣
↪deliver_p2

                    #
```

```
                    (x_s,y_s),(x_l,y_l),index = self.
↪choose_from_p1_p2(str(driver),passenger_list)

                    #
                    self.arrange_route(driver,(x_s,y_s))
                    # new_edge,dis,_ = traci.simulation.
↪convertRoad(x_s,y_s,isGeo=True)
                    # traci.vehicle.changeTarget(str(driver),new_edge)

                    #
                    traci.vehicle.setColor(str(driver),COLOR_DICT["red"])
                    next_posi = (x_l,y_l)
                    if index == 0:
                        driver_condition = "deliver_p1"
                    else:
                        driver_condition = "deliver_p2"

                    # log
                    self.log(driver, driver_condition,"pickup_p2",p1 =␣
↪passenger_list[0],p2 = passenger_list[1])

                    #
                    traci.person.remove(str(passenger))

                    #
                    passenger = passenger_list[0]
                    driver_list, posi, passenger_condition = self.
↪passengers[passenger]
                    passenger_condition = "picked"
                    self.passengers[passenger] = (driver_list, posi,␣
↪passenger_condition)

            elif driver_condition == "deliver_p1" or driver_condition ==␣
↪"deliver_p2":
                #   next_posi
                if len(next_posi) > 0:      # deliver_p2/1 -> deliver_p1/2

                    #
                    p1,p2 = passenger_list
                    tmp_order_1 = self.order[self.order.order_id == p1]
                    x1,y1 = tmp_order_1.dest_lng.item(),tmp_order_1.dest_lat.
↪item()
                    tmp_order_2 = self.order[self.order.order_id == p2]
                    x2,y2 = tmp_order_2.dest_lng.item(),tmp_order_2.dest_lat.
↪item()
```

```
                    if driver_condition == "deliver_p2":   #
                        #   driver   destination
                        dis = self.cal_driver_destination_distance(driver,p2)


                        #
                        if dis < self.distance_threshold:   # deliver_p2 ->␣
↪deliver_p1
                            # log
                            self.log(driver,"deliver_p1","deliver_p2",p1 = p1)

                            #
                            tmp_order_1 = self.order[self.order.order_id == p1]
                            x1,y1 = tmp_order_1.dest_lng.item(),tmp_order_1.
↪dest_lat.item()
                            self.arrange_route(driver,(x1,y1))

                            #
                            traci.vehicle.
↪setColor(str(driver),COLOR_DICT["orange"])
                            driver_condition = "deliver_p1"
                            next_posi = ()

                            # update passenger
                            passenger = passenger_list[1]
                            driver_list, posi, passenger_condition = self.
↪passengers[passenger]
                            passenger_condition = "finish"
                            self.passengers[passenger] =␣
↪(driver_list,posi,passenger_condition)

                            # passenger = passenger_list[0]
                            # driver_list, posi, passenger_condition = self.
↪passengers[passenger]
                            # passenger_condition = "picked"
                            # self.passengers[passenger] = (driver_list, posi,␣
↪passenger_condition)

                            # # remove passenger 2 for driver
                            # passenger_list = [passenger_list[0]]

                    else:  #
                        #   driver   destination
                        dis = self.cal_driver_destination_distance(driver,p1)

                        #
```

```
                    if dis < self.distance_threshold:   # deliver_p1 ->␣
↪deliver_p2
                        # log
                        self.log(driver,"deliver_p2","deliver_p1",p2 = p2)

                        #
                        tmp_order_2 = self.order[self.order.order_id == p2]
                        x2,y2 = tmp_order_2.dest_lng.item(),tmp_order_2.
↪dest_lat.item()
                        self.arrange_route(driver,(x2,y2))

                        #
                        traci.vehicle.
↪setColor(str(driver),COLOR_DICT["orange"])
                        driver_condition = "deliver_p2"
                        next_posi = ()

                        # update passenger
                        passenger = passenger_list[0]
                        driver_list, posi, passenger_condition = self.
↪passengers[passenger]
                        passenger_condition = "finish"
                        self.passengers[passenger] =␣
↪(driver_list,posi,passenger_condition)

                        # passenger = passenger_list[1]
                        # driver_list, posi, passenger_condition = self.
↪passengers[passenger]
                        # passenger_condition = "picked"
                        # self.passengers[passenger] = (driver_list, posi,␣
↪passenger_condition)

                        # remove passenger for driver
                        # passenger_list = [passenger_list[1]]

                else:          # deliver_p1/2 -> trip_end
                    # check
                    if driver_condition == "deliver_p1":   #
                        passenger = passenger_list[0]
                    else:                                  #
                        passenger = passenger_list[1]
                    #   driver  destination
                    dis = self.cal_driver_destination_distance(driver,passenger)

                    #
                    if dis < self.distance_threshold:   #
```

```
                        # log
                        self.log(driver,"waiting",driver_condition)

                        #
                        next_posi = self.random_destination(driver)

                        #
                        traci.vehicle.setColor(str(driver),COLOR_DICT["white"])
                        passenger_list = []
                        driver_condition = "waiting"

                        #
                        driver_list, posi, passenger_condition = self.
↪passengers[passenger]
                        passenger_condition = "finish"
                        self.passengers[passenger] = (driver_list, posi,␣
↪passenger_condition)

            self.drivers[driver] = passenger_list,␣
↪posi,edge,driver_condition,next_posi

        # process actions
        for driver in success_d_p.keys():
            passenger = success_d_p[driver]

            #
            passenger_list, posi,edge,driver_condition,next_posi = self.
↪drivers[driver]
            driver_list, posi, passenger_condition = self.passengers[passenger]

            #   action
            tmp_order = self.order[self.order.order_id == passenger]
            x1,y1 = tmp_order.starting_lng.item(),tmp_order.starting_lat.item()

            #
            # try:
            if driver_condition == "waiting":                    # waiting ->␣
↪pickup_p1
                #
                self.arrange_route(driver,(x1,y1))

                #
                traci.vehicle.setColor(str(driver),COLOR_DICT["yellow"])
                driver_condition = "pickup_p1"
                # log
                self.log(driver,"pickup_p1","waiting")
```

```
            elif driver_condition == "pickup_p1":                    # pickup_p1␣
↪-> pickup_p2
                if len(passenger_list) >=2:
                    print(f"{driver} passenger > 2 {passenger_list},␣
↪{passenger}")
                next_posi = (x1,y1)

            # update
            passenger_list.append(passenger)
            driver_list.append(driver)
            passenger_condition = "picked"
            self.drivers[driver] = (passenger_list,␣
↪posi,edge,driver_condition,next_posi)
            self.passengers[passenger] = (driver_list, posi,␣
↪passenger_condition)

            # except Exception as e:
            #     print(f"Action: {driver} -> {passenger} failed")

        traci.simulationStep()
        # process new orders

        self.current_order = self.order[self.order.relative_seconds == self.
↪time]

        for passenger in self.current_order.itertuples():
            x1,y1 = passenger.starting_lng,passenger.starting_lat
            self.passengers[passenger.order_id] = ([],(x1,y1),"waiting")
            edge,dis,_  = traci.simulation.convertRoad(x1,y1,isGeo=True)
            traci.person.add(str(passenger.order_id),edge,pos =0.0)
            traci.person.appendWaitingStage(str(passenger.order_id),duration =␣
↪float(10000000),)

        self.update_posi()

        return self.make_observation(),{},self.terminate,{}

    def choose_from_p1_p2(self,driver,passenger_list):
        p1,p2 = passenger_list
        #
        tmp_order_1 = self.order[self.order.order_id == p1]
        x1,y1 = tmp_order_1.dest_lng.item(),tmp_order_1.dest_lat.item()
        tmp_order_2 = self.order[self.order.order_id == p2]
        x2,y2 = tmp_order_2.dest_lng.item(),tmp_order_2.dest_lat.item()
        #
        xd,yd = traci.vehicle.getPosition(driver)
```

```
        xd,yd = traci.simulation.convertGeo(xd,yd,fromGeo = False)
        #    1   2
        dis_d_p1 = self.getDistance((xd,yd),(x1,y1))
        dis_p1_p2 = self.getDistance((x1,y1),(x2,y2))
        #    2   1
        dis_d_p2 = self.getDistance((xd,yd),(x2,y2))
        dis_p2_p1 = self.getDistance((x2,y2),(x1,y1))
        dis_1 = dis_d_p1 + dis_p1_p2
        dis_2 = dis_d_p2 + dis_p2_p1
        if dis_1 <= dis_2:
            return (x1,y1),(x2,y2),0

        else:
            return (x2,y2),(x1,y1),1

    def reset(self):
        # start_sumo
        if traci.isLoaded():
            traci.close()
        traci.start([self.sumo_path,'-c',self.sumocfg_path,"--start"])

        # init
        self.terminate = False
        self.time = 0

        self.drivers = {}           #[[  ], (   ) ( & )]
        self.passengers = {}        #[[ ], (   ),  ]
        self.logger = pd.DataFrame()

        self.vehicle_id = 0
        self.route_id = 0


        # init drivers
        self.all_edges = traci.edge.getIDList()
        random.seed(self.seed_)
        selected_seeds = random.choices([i for i in range(100000)],k = self.
↪driver_num)
        drivers_on_road = {}
        for seed in tqdm(selected_seeds,desc = "Add init driver random route"):
            #
            init_edge_id,end_edge_id = self.init_destination(seed)

            traci.vehicle.setColor(str(self.vehicle_id),COLOR_DICT["white"])
            drivers_on_road[self.vehicle_id] = (self.
↪all_edges[init_edge_id],self.all_edges[end_edge_id])
            self.vehicle_id += 1
```

```
            self.route_id += 1

        # process obs
        self.current_order = self.order[self.order.relative_seconds == self.
↪time]

        for driver in tqdm(drivers_on_road.keys(),desc = "Get driver current␣
↪position"):
            init_edge,end_edge = drivers_on_road[driver]
            x3,y3 = traci.simulation.convert2D(end_edge , 0, toGeo= True)
            next_posi = (x3,y3)
            x1,y1 = traci.simulation.convert2D(init_edge, 0, toGeo = True)
            self.drivers[driver] =␣
↪([],(x1,y1),(init_edge,end_edge),"waiting",next_posi)


        # passenger at 0 time
        for passenger in self.current_order.itertuples():
            x1,y1 = passenger.starting_lng,passenger.starting_lat
            self.passengers[passenger.order_id] = ([],(x1,y1),"waiting")
            edge,dis,_ = traci.simulation.convertRoad(x1,y1,isGeo=True)
            traci.person.add(str(passenger.order_id),edge,pos =0)
            traci.person.appendWaitingStage(str(passenger.order_id),duration =␣
↪float(10000000),)

        # step
        traci.simulationStep()

        # log
        for driver in self.drivers.keys():
            self.log(driver,"waiting","init")

        # tqdm
        self.pbar = tqdm(total = self.total_timesteps)

        return self.make_observation()

    def cal_driver_passenger_distance(self,driver,passenger):
        try:
            d_rid = traci.vehicle.getRoadID(str(driver))
            d_posi = traci.vehicle.getLanePosition(str(driver))
            tmp_order = self.order[self.order.order_id == passenger]
            x1,y1 = tmp_order.starting_lng.item(),tmp_order.starting_lat.item()
            p_edge, p_posi,_ = traci.simulation.convertRoad(x1,y1, isGeo=True)

            return traci.simulation.getDistanceRoad(d_rid, d_posi,␣
↪p_edge,p_posi,isDriving = True)
```

```
        except:
            return 100000000

    def cal_driver_destination_distance(self,driver,passenger):
        #   passenger
        tmp_order = self.order[self.order.order_id == passenger]
        x1,y1 = tmp_order.dest_lng.item(),tmp_order.dest_lat.item()
        p_edge, p_posi,_ = traci.simulation.convertRoad(x1,y1, isGeo=True)
        #
        d_edge = traci.vehicle.getRoadID(str(driver))
        d_posi = traci.vehicle.getLanePosition(str(driver))
        #
        dis =  traci.simulation.getDistanceRoad(d_edge, d_posi, p_edge, p_posi,␣
↪isDriving = True)

        return dis

    def cal_driver_coordinate_distance(self,driver,p1):
        #
        x1,y1 = p1
        p_edge, p_posi,_ = traci.simulation.convertRoad(x1,y1, isGeo=True)
        #
        d_edge = traci.vehicle.getRoadID(str(driver))
        d_posi = traci.vehicle.getLanePosition(str(driver))
        #
        dis =  traci.simulation.getDistanceRoad(d_edge, d_posi, p_edge, p_posi,␣
↪isDriving = True)

        return dis

    def arrange_route(self,driver,p1):
        #

        #
        x1,y1 = p1
        new_edge,dis,_ = traci.simulation.convertRoad(x1,y1,isGeo=True)
        all_edges = list(self.all_edges)
        all_edges.remove(new_edge)

        try:
            #  1
            traci.vehicle.changeTarget(str(driver),new_edge)
        except:
            #  2
            init_edge = traci.vehicle.getRoadID(str(driver))
            init_posi = traci.vehicle.getLanePosition(str(driver))
            success = False
```

```
            while not success:
                try:

                    route = traci.simulation.findIntermodalRoute( init_edge,␣
↪new_edge )
                    assert len(route) > 0
                    success = True
                except:
                    print("Reroute again")
                    all_edges_lenth = [traci.simulation.
↪getDistanceRoad(init_edge, init_posi, edge, 0, isDriving = True) for edge in␣
↪all_edges]
                    shortest_edge_idx = np.argmax(all_edges_lenth)
                    new_edge = all_edges[shortest_edge_idx]
                    all_edges.remove(new_edge)

            print("Reroute success")

            route = route[0]
            traci.vehicle.remove(str(driver))
            traci.route.add(str(self.route_id),edges = route.edges)
            traci.vehicle.add(
                vehID = str(driver),
                routeID = str(self.route_id),
                personNumber = 0
                )
            self.route_id += 1

    def random_destination(self,driver):
        success = False
        while not success:
            try:
                end_edge_id = random.choices([i for i in range(len(self.
↪all_edges))],k = 1)[0]
                end_edge = self.all_edges[end_edge_id]
                x3,y3 = traci.simulation.convert2D(end_edge, 0, toGeo= True)
                next_posi = (x3,y3)
                traci.vehicle.changeTarget(str(driver),end_edge)
                success = True
            except Exception as e:
                # print(e,end= " ")
                pass

        return next_posi

    def init_destination(self,seed = 42):
        success = False
```

```
        while not success:
            try:
                random.seed(seed)
                init_edge_id = random.choices([i for i in range(len(self.
↪all_edges))],k = 1)[0]
                init_edge = self.all_edges[init_edge_id]
                random.seed(seed+1)
                end_edge_id = random.choices([i for i in range(len(self.
↪all_edges))],k = 1)[0]
                end_edge = self.all_edges[end_edge_id]
                route = traci.simulation.findRoute( init_edge, end_edge )
                dis = traci.simulation.getDistanceRoad(init_edge,0,end_edge,0,)

                assert dis > 1000
                traci.route.add(str(self.route_id),edges = route.edges)
                traci.vehicle.add(
                    vehID = str(self.vehicle_id),
                    routeID = str(self.route_id),
                    personNumber = 0
                        )
                # d_edge = traci.vehicle.getRoadID(str(self.vehicle_id))
                # d_posi = traci.vehicle.getLanePosition(str(self.vehicle_id))
                # x1,y1 = traci.simulation.convert2D(end_edge , 0, toGeo= True)
                # p_edge, p_posi,_ = traci.simulation.convertRoad(x1,y1,␣
↪isGeo=True)
                # dis =  traci.simulation.getDistanceRoad(d_edge, d_posi,␣
↪p_edge, p_posi, isDriving = True)
                success = True
            except Exception as e:
                # print(e)
                seed += 1

        return init_edge_id,end_edge_id

    @staticmethod
    def cal_distence(p1,p2,isGeo = True):
        x1,y1 = p1
        x2,y2 = p2
        if (x2,y2) == (-1073741824.0,-1073741824.0):
            return 1e7
        # try:
        if isGeo:
            x1,y1 = traci.simulation.convertGeo(x1,y1,fromGeo= True)
            x2,y2 = traci.simulation.convertGeo(x2,y2,fromGeo= True)
        return traci.simulation.getDistance2D(x1,y1,x2,y2,isGeo=False,␣
↪isDriving = False)
        # except:
```

```
            # return 1e7

    @staticmethod
    def get_path(p1,p2,isGeo=True):
        x1,y1 = p1
        x2,y2 = p2
        edge_1,dis,_ = traci.simulation.convertRoad(x1,y1,isGeo=isGeo)
        edge_2,dis,_ = traci.simulation.convertRoad(x2,y2,isGeo=isGeo)
        res = traci.simulation.findIntermodalRoute(edge_1,edge_2)[0]
        edges = res.edges
        length = res.length
        return edges,length

    def find_route(self,edge_id):
        traci.route.add(str(self.route_id),edges = self.all_edges[edge_id:
↪edge_id+1])    #

    def update_posi(self):
        for driver in self.drivers.keys():
            passenger_list, posi, edge, condition, next_posi = self.
↪drivers[driver]
            x1,y1 = traci.vehicle.getPosition(str(driver))
            x1,y1 = traci.simulation.convertGeo(x1,y1,fromGeo = False)
            posi = (x1,y1)
            self.drivers[driver] = (passenger_list, posi, edge, condition,␣
↪next_posi)

    def getDistance(self,p1,p2):
        x1,y1 = p1
        x2,y2 = p2
        edgeid_1, posi_1,_ = traci.simulation.convertRoad(x1,y1, isGeo=True)
        edgeid_2, posi_2,_ = traci.simulation.convertRoad(x2,y2, isGeo=True)
        res = traci.simulation.getDistanceRoad(edgeid_1, posi_1,␣
↪edgeid_2,posi_2,isDriving = True)
        return res

    def make_observation(self):
        no_passenger_driver = {}
        one_passenger_driver = {}
        two_passenger_driver = {}
        arranged_passenger = {}
        not_arranged_passenger = {}

        for driver in self.drivers.keys():
            passenger_list, posi,edge,condition,next_posi = self.drivers[driver]
            if len(passenger_list) == 0:
                no_passenger_driver[driver] = self.drivers[driver]
```

```
            elif len(passenger_list) == 1:
                one_passenger_driver[driver] = self.drivers[driver]
            elif len(passenger_list) == 2:
                two_passenger_driver[driver] = self.drivers[driver]
            else:
                raise ValueError(f"The max number of passengers should be 2,␣
↪but you passenger list is {passenger_list}")

        for passenger in self.passengers.keys():
            driver_list, posi, condition = self.passengers[passenger]
            if condition != "finish":
                if len(driver_list) == 0:
                    not_arranged_passenger[passenger] = self.
↪passengers[passenger]
                elif len(driver_list) == 1:
                    arranged_passenger[passenger] = self.passengers[passenger]


        return (no_passenger_driver,
                one_passenger_driver,
                two_passenger_driver,
                arranged_passenger,
                not_arranged_passenger)

    def log(self,driver,condition,last_condition,p1 = np.nan,p2= np.nan):
        driver = driver                                                    # driver_id
        timestep = self.time                                               #
        condition = condition                                              #
        last_condition = last_condition                                    #
        x1,y1 = traci.vehicle.getPosition(str(driver))                     #
        p1,p2 = p1,p2                                                      #
        x1,y1 = traci.simulation.convertGeo(x1,y1,fromGeo = False)
        tmp_df = pd.
↪DataFrame([timestep,driver,condition,last_condition,x1,y1,p1,p2]).T
        tmp_df.columns =␣
↪["timestep","driver","condition","last_condition","x1","y1","p1","p2"]
        self.logger = pd.concat([self.logger,tmp_df])
```

[ ]:
```
# fake agent
def cal_action(obs):
    no_passenger_driver, one_passenger_driver, two_passenger_driver,␣
↪arranged_passenger, not_arranged_passenger = obs
    action_dict = {}
    for passenger in not_arranged_passenger.keys():
        driver_list, posi_passenger, passenger_condition =␣
↪not_arranged_passenger[passenger]
        shortest_dis = 1000000000
```

```
        for driver in no_passenger_driver.keys():
            passenger_list, posi_driver, edge, driver_condition, next_posi =␣
↪no_passenger_driver[driver]
            dis = env.cal_distence(posi_passenger, posi_driver)
            if dis > 0:
                if dis < shortest_dis:
                    shortest_dis = dis
                    shortest_driver = driver

        for driver in one_passenger_driver.keys():
            passenger_list, posi_driver, edge, driver_condition, next_posi =␣
↪one_passenger_driver[driver]
            dis = env.cal_distence(posi_passenger, posi_driver)
            if dis > 0:
                if dis < shortest_dis:
                    shortest_dis = dis
                    shortest_driver = driver
        if len(no_passenger_driver)!=0 or len(one_passenger_driver)!=0:
            action_dict[shortest_driver] = passenger

    return [action_dict,[],[]]
```

[ ]:
```
env = SUMO_env(env_args)
obs = env.reset()
terminate = False
step = 0
while not terminate:
    # print(step,end = " ",flush=True)
    step += 1
    # if step == 10:
    #     break
    action = cal_action(obs)
    obs,_,terminate,_= env.step(action)

    obs_rem = obs
    action_rem = action
```

```
Add init driver random route:   0%|          | 0/20 [00:00<?, ?it/s]
Get driver current position:   0%|          | 0/20 [00:00<?, ?it/s]
  0%|          | 0/1800 [00:00<?, ?it/s]
Reroute success
Reroute success
Reroute success
Reroute success
Reroute success
```

```
Reroute success
Reroute success
```

[ ]:
```
df = env.logger
df[df.driver == 1]
# df
```

[ ]:
```
   timestep driver   condition last_condition          x1         y1  \
0         0      1     waiting           init  110.347041   19.99826
0         1      1   pickup_p1        waiting  110.347041   19.99826
0       120      1   pickup_p2      pickup_p1  110.332976  19.993174
0       393      1  deliver_p2      pickup_p2  110.373423  20.008917
0       738      1  deliver_p1     deliver_p2  110.344013   19.98047
0      1125      1     waiting     deliver_p1  110.461072  19.943233
0      1126      1   pickup_p1        waiting   110.46159  19.943288
0      1692      1   pickup_p2      pickup_p1  110.349039  20.050488
0      1734      1  deliver_p2      pickup_p2         inf        inf

                p1              p2
0              NaN             NaN
0              NaN             NaN
0  17592362531154             NaN
0  17592362531154  17592360594114
0  17592362531154             NaN
0              NaN             NaN
0              NaN             NaN
0  17592367860207             NaN
0  17592367860207  17592367795926
```

[ ]:
```
df = env.logger
df[df.driver == 3]
```

[ ]:
```
   timestep driver   condition last_condition          x1         y1  \
0         0      3     waiting           init  110.344706  20.058233
0        27      3   pickup_p1        waiting  110.341385  20.058297
0       230      3   pickup_p2      pickup_p1  110.322855  20.027187
0       374      3  deliver_p1      pickup_p2  110.348439  20.038385
0       461      3  deliver_p2     deliver_p1  110.343345   20.02339
0       687      3     waiting     deliver_p2  110.320954  19.992836
0       688      3   pickup_p1        waiting  110.320979  19.992829
0       925      3   pickup_p2      pickup_p1   110.37442  19.990677
0      1031      3  deliver_p2      pickup_p2  110.363067  19.980202
0      1220      3  deliver_p1     deliver_p2  110.339582  20.003247
0      1349      3     waiting     deliver_p1  110.333319  20.027548
0      1350      3   pickup_p1        waiting  110.333423  20.027651
0      1461      3   pickup_p2      pickup_p1  110.354898  20.028394
0      1782      3  deliver_p1      pickup_p2  110.355553  20.074555
```

	p1	p2
0	NaN	NaN
0	NaN	NaN
0	17592367673708	NaN
0	17592367673708	17592367487081
0	NaN	17592367487081
0	NaN	NaN
0	NaN	NaN
0	17592367620723	NaN
0	17592367620723	17592367715698
0	17592367620723	NaN
0	NaN	NaN
0	NaN	NaN
0	17592367720933	NaN
0	17592367720933	17592367875538