Traffic Optimization Code

by Kashish Mamania

Submission date: 16-Apr-2025 10:32PM (UTC+0530)
Submission ID: 2648087815
File name: Traffic_Optimization_Code.docx (34.5K)
Word count: 1470
Character count: 11132

```
from __future__ import absolute_import, print_function

import os
import sys
import optparse
import subprocess
import random
import copy
import numpy as np

try:
    sys.path.append(
        os.path.join(os.path.dirname(__file__), '..', '..', '..', '..', "tools")
    )  # tutorial in tests
    sys.path.append(
        os.path.join(
            os.environ.get("SUMO_HOME", os.path.join(os.path.dirname(__file__), "..", "..", "..")),
            "tools"
        )
    )  # tutorial in docs
    from sumolib import checkBinary
except ImportError:
    sys.exit(
        "Please declare environment variable 'SUMO_HOME' as the root directory of "
        "your sumo installation (it should contain folders 'bin', 'tools' and 'docs')"
    )

import traci
from junction import Junction
from device import Device
```

```
from phaseConfig import setJunctionPhase

junction_U = Junction(
    _id='U',
    dev_a_dets=['0', '1', '2', '3', '4', '5'],
    dev_b_dets=['6', '7', '8', '9', '10', '11'],
    dev_c_dets=['54', '55', '56', '57', '58', '59'],
    dev_d_dets=['60', '61', '62', '63', '64', '65'],
    phaseMap={1: 1, 2: 2, 3: 4, 4: 3}
)

junction_L = Junction(
    _id='L',
    dev_a_dets=['18', '19', '20', '21', '22', '23'],
    dev_b_dets=['12', '13', '14', '15', '16', '17'],
    dev_c_dets=['66', '67', '68', '69', '70', '71'],
    dev_d_dets=['24', '25', '26', '27', '28', '29'],
    phaseMap={1: 1, 2: 2, 3: 4, 4: 3}
)

junction_R = Junction(
    _id='R',
    dev_a_dets=['30', '31', '32', '33', '34', '35'],
    dev_b_dets=['48', '49', '50', '51', '52', '53'],
    dev_c_dets=['36', '37', '38', '39', '40', '41'],
    dev_d_dets=['42', '43', '44', '45', '46', '47'],
    phaseMap={1: 3, 2: 1, 3: 2, 4: 4}
)

# Set neighbours
junction_U.neighbours = [
```

```
    {'junction': junction_L, 'connection': ('d', 'b'), 'data': 0}
    {'junction': junction_R, 'connection': ('c', 'b'), 'data': 0}
]
junction_L.neighbours = [
    {'junction': junction_R, 'connection': ('c', 'a'), 'data': 0},
    {'junction': junction_U, 'connection': ('b', 'd'), 'data': 0}
]
junction_R.neighbours = [
    {'junction': junction_L, 'connection': ('a', 'c'), 'data': 0},
    {'junction': junction_U, 'connection': ('b', 'c'), 'data': 0}
]

def run(i):
    print("\nIteration :", i, "\n")
    output.write("\n")
    output.write("Iteration:" + str(i) + "\n")
    output.write("--------------------------------\n")

    endSimTime = 16
    global steps, allWaitingTime, allTravelTime, allarrived, alldeparted
    global fitnessInd, phaseInd, AllFit, currentvechiles, AllPhase
    global BestPopInd, BestPopFit, fuelConsumption

    steps = 0
    step = 0
    allWaitingTime = []
    allTravelTime = []
    allarrived = []
    # alldeparted is declared global later in __main__
    fuelConsumption = []
```

```
initial = [
    [50, 23, 59], [23, 50, 45], [52, 7, 49], [22, 59, 38], [25, 7, 44],
    [32, 56, 59], [15, 46, 9], [10, 8, 44], [6, 57, 27], [40, 10, 33],
    [45, 44, 14], [59, 19, 56], [22, 12, 28], [8, 23, 26], [45, 16, 39],
    [56, 48, 57], [46, 51, 47], [17, 54, 51], [54, 30, 11], [36, 25, 30],
    [24, 41, 56], [47, 48, 39], [60, 28, 39], [52, 44, 18], [50, 58, 55],
    [8, 21, 9], [50, 24, 47], [32, 30, 19], [29, 38, 23], [13, 44, 45]
]
global x
x = 0
global updated


if i == 0:
    while step < endSimTime:
        print("\n------------------------------------------------------------\n")
        print("Steps :", step)
        # traci.simulationStep()
        GAphase = initial[x]
        x += 1
        phaseInd.append(GAphase)
        temp1 = []
        temp2 = []
        phase = int((GAphase[0] + GAphase[1] + GAphase[2]) / 3)
        runDeviceDetect(phase)


        # gets data from devices for junctions for "time" number of simulation steps
        edgeIDs = traci.edge.getIDList()
        for j in edgeIDs:
            temp1.append(traci.edge.getTraveltime(j))
            temp2.append(traci.edge.getWaitingTime(j))
```

```
        if sum(alldeparted) == 0:
            allWaitingTime.append(sum(temp1))
        else:
            allWaitingTime.append(sum(temp1) / sum(alldeparted))
        allTravelTime.append(sum(temp2))

        Cr = phase * (7 / 21)
        if sum(allarrived) == 0:
            fitness = (
                sum(temp2) + sum(temp1) + (sum(alldeparted) - sum(allarrived)) *
traci.simulation.getTime()
            ) / 1 + Cr
        else:
            fitness = (
                sum(temp2) + sum(temp1) + (sum(alldeparted) - sum(allarrived)) *
traci.simulation.getTime()
            ) / (sum(allarrived)) ** 2 + Cr

        fitnessInd.append(fitness)
        AllFit.append(fitnessInd)
        AllPhase.append(phaseInd)

        currentvechiles = traci.vehicle.getIDList()
        for v in currentvechiles:
            Fue = traci.vehicle.getFuelConsumption(str(v))
            fuelConsumption.append(Fue)

        print("Phase Average: ", phase, "   ", "Fitness:", fitness)
        print("Arrived Cars : ", sum(allarrived))
        print("Departed Cars : ", sum(alldeparted))
        print("Current Simulation time : ", traci.simulation.getTime())
        print("Avg Waiting time: ", sum(allWaitingTime) / len(allWaitingTime))
```

```
            print("Avg Travel time: ", sum(allTravelTime) / len(allTravelTime))
            print("Avg Fuel Consumption: ", sum(fuelConsumption) / len(fuelConsumption))

            output.write("Step:" + str(step) + "\n")
            output.write("Phases:" + str(GAphase) + "\n")
            output.write("Phase Average: " + str(phase) + "\n")
            output.write("Fitness:" + str(fitness) + "\n")
            output.write("Arrived Cars:" + str(sum(allarrived)) + "\n")
            output.write("Departed Cars:" + str(sum(alldeparted)) + "\n")
            output.write("Avg Waiting time: " + str(sum(allWaitingTime) / len(allWaitingTime)) + "\n")
            output.write("Avg travel time: " + str(sum(allTravelTime) / len(allTravelTime)) + "\n")
            output.write("Avg Fuel Consumption:" + str(sum(fuelConsumption) / len(fuelConsumption)) +
"\n")
            output.write("------------------------------------------------------------------------\n")

            step += 1
            useAlgoAndSetPhase()
            """
            Use an algorithm to set the phase for the junctions
            """
            prepareJunctionVectArrs()
            """
            Prepare the vehicle vector array for junctions
            """
            setJunctionPhasesInSUMO()
            """
            Set the junction's phases in the SUMO simulator
            """

        Bstfit = min(fitnessInd)
        idx = fitnessInd.index(Bstfit)
```

```
        BstInd = phaseInd[idx]
        BestPopFit.append(Bstfit)
        BestPopInd.append(BstInd)

    else:
        step = 0
        allWaitingTime = []
        allTravelTime = []
        allarrived = []
        alldeparted = []
        fuelConsumption = []
        steps = 0

        ga = GA(AllPhase[-1], AllFit[-1], pcross=0.6, pmuta=0.05)
        updated = ga[0]
        BestPopInd.append(ga[1])
        BestPopFit.append(ga[2])
        AllPhase.append(updated)

        h = 0
        phaseInd = []
        fitnessInd = []

        while step < len(updated):
            temp1 = []
            temp2 = []
            Ind = updated[h]
            h += 1
            phase = int((Ind[0] + Ind[1] + Ind[2]) / 3)
            runDeviceDetect(phase)
```

```
        # Gets data from devices for junctions for "time" number of simulation steps
        edgeIDs = traci.edge.getIDList()
        for j in edgeIDs:
            temp1.append(traci.edge.getTraveltime(j))
            temp2.append(traci.edge.getWaitingTime(j))

        if sum(alldeparted) == 0:
            allWaitingTime.append(sum(temp1))
        else:
            allWaitingTime.append(sum(temp1) / sum(alldeparted))
        allTravelTime.append(sum(temp2))

        Cr = phase * (7 / 21)
        if sum(allarrived) == 0:
            fitness = (
                sum(temp2) + sum(temp1) + (sum(alldeparted) - sum(allarrived)) *
traci.simulation.getTime()
            ) / 1 + Cr
        else:
            fitness = (
                sum(temp2) + sum(temp1) + (sum(alldeparted) - sum(allarrived)) *
traci.simulation.getTime()
            ) / (sum(allarrived)) ** 2 + Cr

        fitnessInd.append(fitness)
        fuelConsumption = []
        currentvechiles = traci.vehicle.getIDList()
        for v in currentvechiles:
            Fue = traci.vehicle.getFuelConsumption(str(v))
            fuelConsumption.append(Fue)

        print("\n----------------------------------------------------------\n")
```

```
            print("Steps: ", step)
            print("Phase Average: ", phase, "   ", "Fitness:", fitness)
            print("Arrived Cars : ", sum(allarrived))
            print("Departed Cars: ", sum(alldeparted))
            print("Current Simulation time : ", traci.simulation.getTime())
            print("Waiting time: ", sum(allWaitingTime) / len(allWaitingTime))
            print("Travel time: ", sum(allTravelTime) / len(allTravelTime))
            print("Fuel Consumption: ", sum(fuelConsumption) / len(fuelConsumption))

            output.write("Step:" + str(step) + "\n")
            output.write("Phase Average: " + str(Ind) + "\n")
            output.write("Phase Average: " + str(phase) + "\n")
            output.write("Fitness:" + str(fitness) + "\n")
            output.write("Arrived Cars:" + str(sum(allarrived)) + "\n")
            output.write("Departed Cars:" + str(sum(alldeparted)) + "\n")
            output.write("Avg Waiting time: " + str(sum(allWaitingTime) / len(allWaitingTime)) + "\n")
            output.write("Avg travel time: " + str(sum(allTravelTime) / len(allTravelTime)) + "\n")
            output.write("Avg Fuel Consumption:" + str(sum(fuelConsumption) / len(fuelConsumption)) +
"\n")
            output.write("------------------------------------------------------------------------------\n")

            useAlgoAndSetPhase()
            prepareJunctionVectArrs()
            setJunctionPhasesInSUMO()
            step += 1

    AllFit.append(fitnessInd)
    return BestPopFit, BestPopInd

def setJunctionPhasesInSUMO():
    setJunctionPhase(junction_U, setAllRed=False)
```

```
    setJunctionPhase(junction_L, setAllRed=False)
    setJunctionPhase(junction_R, setAllRed=False)
    return

def useAlgoAndSetPhase():
    junction_U.update()
    junction_L.update()
    junction_R.update()
    return

def runDeviceDetect(time):
    global steps
    for _ in range(time):
        junction_U.checkDevices()
        junction_L.checkDevices()
        junction_R.checkDevices()
        allarrived.append(traci.simulation.getArrivedNumber())
        alldeparted.append(traci.simulation.getDepartedNumber())
        traci.simulationStep()
        steps += 1
    return

def prepareJunctionVectArrs():
    junction_U.prepareVehVectarr()
    junction_L.prepareVehVectarr()
    junction_R.prepareVehVectarr()
    return

def individual(indSiz):
    return [random.randint(5, 60) for _ in range(indSiz)]
```

```
def Elitism(pop, popFit):
    elit = []
    elit_fit = []

    bestFit = min(popFit)
    idx = popFit.index(bestFit)
    elit.append(pop[idx])
    elit_fit.append(bestFit)
    pop.pop(idx)
    popFit.pop(idx)

    bestFit2 = min(popFit)
    idx = popFit.index(bestFit2)
    elit.append(pop[idx])
    elit_fit.append(bestFit2)
    pop.pop(idx)
    popFit.pop(idx)

    return elit, pop, popFit, elit_fit

def Fit_Calculations(popFit):
    total = sum(popFit)
    return [fit / total for fit in popFit]

def Calc_Commulative_fit(fit):
    CommulativeFit = []
    temp = 0.0
    for f in fit:
        temp += f
        CommulativeFit.append(temp)
    return CommulativeFit
```

```
def Roullette_Selection(CommulativeFit, pop):
    selected_ind = []
    for _ in range(len(CommulativeFit)):
        selection_Prob = np.random.rand()
        selected = pop[0]
        for j in range(len(CommulativeFit)):
            if CommulativeFit[j] >= selection_Prob:
                selected = pop[j]
                selected_ind.append(selected)
                break
    return selected_ind


def crossOver(popSiz, indSiz, selected_ind, pCross=0.6):
    newpop = []
    while len(newpop) != popSiz - 2:
        for x in range(0, len(selected_ind), 2):
            cutPoint = int(np.round(np.random.rand() * (indSiz - 1)))
            p1 = selected_ind[x]
            p2 = selected_ind[x + 1]
            offspring1 = p1[:cutPoint] + p2[cutPoint:]
            offspring2 = p2[:cutPoint] + p1[cutPoint:]
            crossProb = np.random.rand()
            if crossProb > pCross:
                newpop.extend([p1, p2])
            else:
                newpop.extend([offspring1, offspring2])
    return newpop


def mutation(pop, indSiz, elit, pmuta=0.05):
    fine_pop = []
```

```
    mut = np.random.randint(5, 10)
    for p in pop:
        for x in range(indSiz):
            if np.random.rand() < pmuta:
                p[x] = abs(p[x] - mut)
        fine_pop.append(p)
    fine_pop.extend(elit)
    return fine_pop


def GA(pop, popFit, pcross=0.6, pmuta=0.05):
    popSiz = len(pop)
    elit, remaining_pop, remaining_fit, elit_fit = Elitism(pop, popFit)
    fit = Fit_Calculations(elit_fit)
    CommulativeFit = Calc_Commulative_fit(fit)
    selected_ind = Roullette_Selection(CommulativeFit, remaining_pop)
    newpop = crossOver(popSiz, 3, selected_ind, pcross)
    pop_final = mutation(newpop, 3, elit, pmuta)
    fittest = min(elit_fit)
    idx = elit_fit.index(fittest)
    best_individual = elit[idx]
    return pop_final, best_individual, fittest


def get_options():
    optParser = optparse.OptionParser()
    optParser.add_option(
        "--nogui", action="store_true", default=False,
        help="run the commandline version of sumo"
    )
    options, _ = optParser.parse_args()
    return options
```

```
if __name__ == "__main__":
    options = get_options()

    global alldeparted, fitnessInd, phaseInd, AllFit, AllPhase, BestPopInd, BestPopFit
    alldeparted = []
    fitnessInd = []
    phaseInd = []
    AllFit = []
    AllPhase = []
    BestPopInd = []
    BestPopFit = []

    # this script has been called from the command line.
    # It will start sumo as a server, then connect and run
    if options.nogui:
        sumoBinary = checkBinary('sumo')
    else:
        sumoBinary = checkBinary('sumo-gui')

    # first, generate the route file for this simulation
    # generate_routefile()

    # this is the normal way of using traci.
    # sumo is started as a subprocess and then the python script connects and runs
    for i in range(10):
        traci.start([sumoBinary, "-c", "../city.sumocfg",
                     "--tripinfo-output", "../tripinfo.xml"])
        output = open("output.txt", "a")
        Result = run(i)
        print("\n----------------------------------------------------------")
        print("End of iteration : ", i)
```

```
    print("Best Phases and its fitness for the entire iteration : ")
    print(Result[1][-1], ",", Result[0][-1])
    print("------------------------------------------------------------")
    output.write("End of iteration: " + str(i) + "\n")
    output.write("Best phases:" + str(Result[1][-1]) + "\n")
    output.write("Best fitness:" + str(Result[0][-1]) + "\n")
traci.close()
```

Traffic Optimization Code

ORIGINALITY REPORT

11%	11%	5%	11%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	intelaligent.github.io Internet Source	8%
2	www.mathworks.com Internet Source	2%
3	Submitted to Asian Institute of Technology Student Paper	1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		