智慧型汽車導論 Homework 2

1. MILP Linearization

Q1.

prove $\alpha + \beta + \gamma \neq 2 \Longleftrightarrow \alpha + \beta - \gamma \le 1 \wedge \alpha - \beta + \gamma \le 1 \wedge -\alpha + \beta + \gamma \le 1$

α	**β**	**γ**	**LHS**	**α + β − γ ≤ 1**	**α − β + γ ≤ 1**	**−α + β + γ ≤ 1**	**RHS**	**LHS=RHS?**
0	0	0	T	T	T	T	T	T
0	0	1	T	T	T	T	T	T
0	1	0	T	T	T	T	T	T
0	1	1	F	T	T	F	F	T
1	0	0	T	T	T	T	T	T
1	0	1	F	T	F	T	F	T
1	1	0	F	F	T	T	F	T
1	1	1	T	T	T	T	T	T

$\alpha + \beta + \gamma$ = 0, 1, 1, 2, 1, 2, 2, 3
$\alpha + \beta - \gamma$ = 0, -1, 1, 0, 1, 0, 2, 1
$\alpha - \beta + \gamma$ = 0, 1, -1, 0, 1, 2, 0, 1
$-\alpha + \beta + \gamma$ = 0, 1, 1, 2, -1, 0, 0, 1

Q2.

prove $\alpha\beta = \gamma \Longleftrightarrow \alpha + \beta - 1 \le \gamma \wedge \gamma \le \alpha \wedge \gamma \le \beta$

α	**β**	**γ**	**LHS**	**α + β − 1 ≤ γ**	**γ ≤ α**	**γ ≤ β**	**RHS**	**LHS=RHS?**
0	0	0	T	T	T	T	T	T
0	0	1	F	T	F	F	F	T
0	1	0	T	T	T	T	T	T
0	1	1	F	T	F	T	F	T
1	0	0	T	T	T	T	T	T
1	0	1	F	T	T	F	F	T
1	1	0	F	F	T	T	F	T
1	1	1	T	T	T	T	T	T

$\alpha\beta$ = 0, 0, 0, 0, 0, 0, 1, 1
$\alpha + \beta - 1$ = -1, -1, 0, 0, 0, 0, 1, 1

Q3.

Given β which is a binary variable, x, y which are non-negative real variables, and a constraint $x \le 2021$, select a value of M to guarantee $\beta x = y \Longleftrightarrow 0 \le y \le x \wedge x - M(1 - \beta) \le y \wedge y \le M\beta$.

LHS: $0 = y$
RHS: $x - M \le y = 0 \le x$
constraint: $0 \le x \le 2021$
$x - M \le 0 \rightarrow x \le M \rightarrow M = 2021$

LHS: $x = y$
RHS: $0 \le y = x \le M$
constraint: $0 \le x \le 2021$
$0 \le y = x \le M \rightarrow 0 \le x \le M \rightarrow M = 2021$

ANS: M = 2021

2. Signal Packing

Q1.

[Original design]
length of μ0: 8+44+3 = 55
length of μ1: 16+44+3 = 63
Total length: 55+63 = 118

[Redesign]
length of μ'0: 16+44+3 = 63
Total length: 63

The new design is better, since the number of bits that need to be transmitted are reduced by 55 bits. New design cuts down the duplicate transmition of headers and other fields.

Q2.

No, the μ2 and μ'0 are on different ECUs. Messages from different senders can not be merged together.

Q3.

According to Q2, μ2 can not be merged into μ'0.
μ3 has the same sender ECU as μ'0, but their periods are not the same. If μ3 is to be merged into μ'0, the period will be 50ms, because it is not allowed to have less frequent (period=100ms) messages.

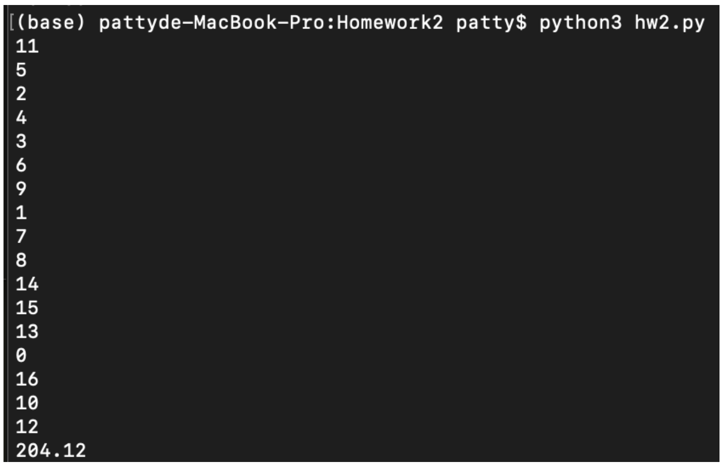
[Original design]
length of μ3: 16+44+3 = 63
Every 100ms, μ'0 and μ3 transmit (63x2)+63 = 189 bits

[Redesign]
length of new μ'0: 32+44+3 = 79
Every 100ms, new μ'0 transmits 79x2 = 158 bits

By merging the μ'0 and μ3 and changing the period to 50ms, the number of transmitted bits can be reduced by 31 bit every 100ms.

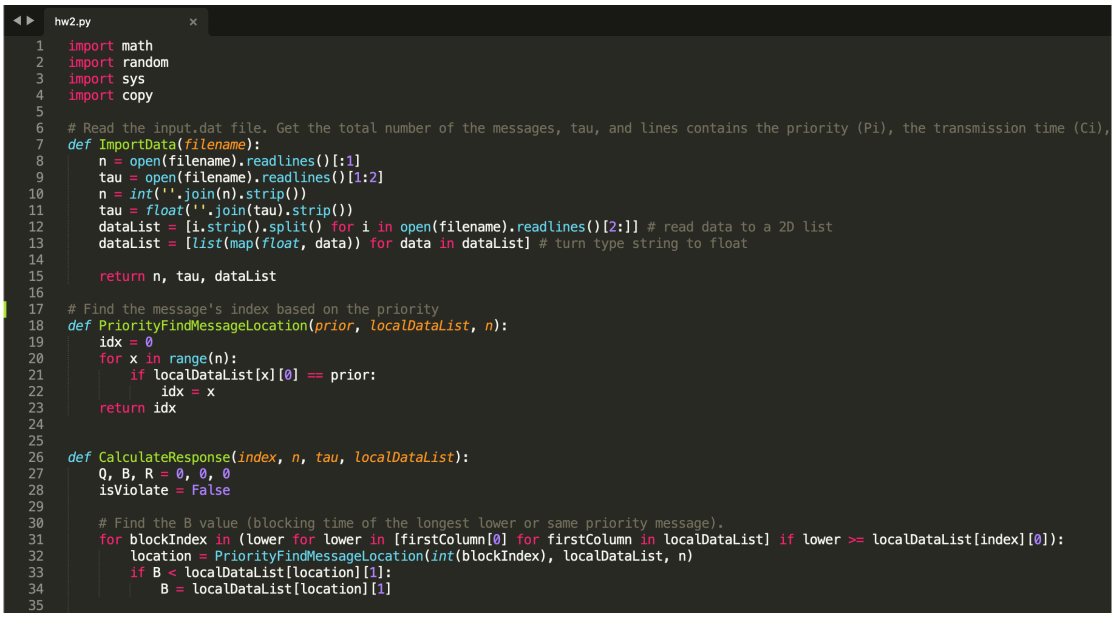
3. Simulated Annealing for Priority Assignment

Result

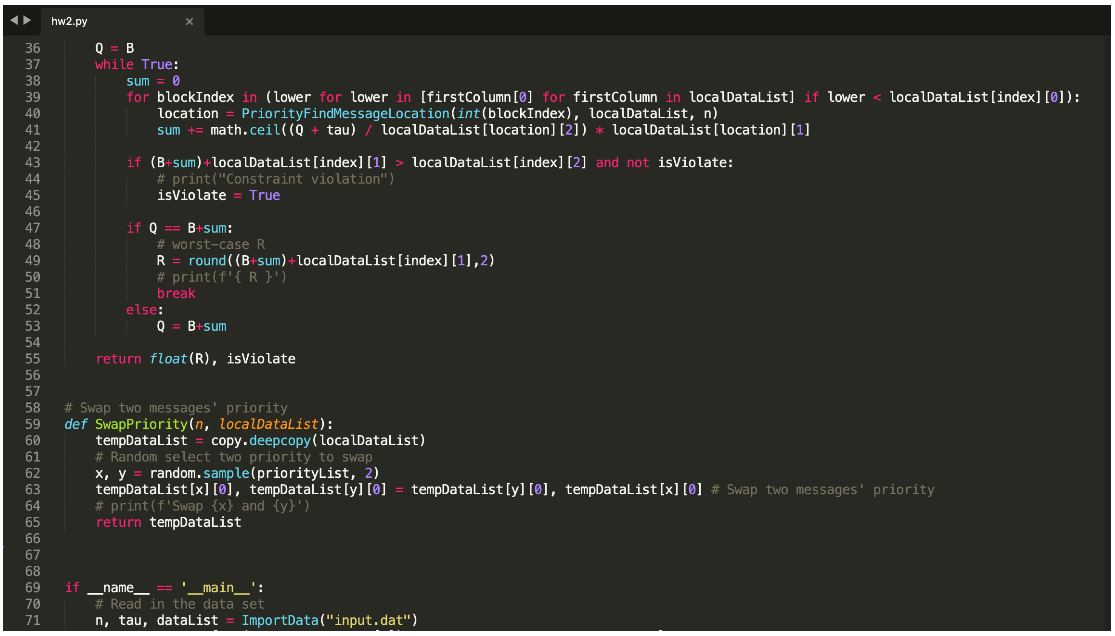


```
(base) pattyde-MacBook-Pro:Homework2 patty$ python3 hw2.py
11
5
2
4
3
6
9
1
7
8
14
15
13
0
16
10
12
204.12
```

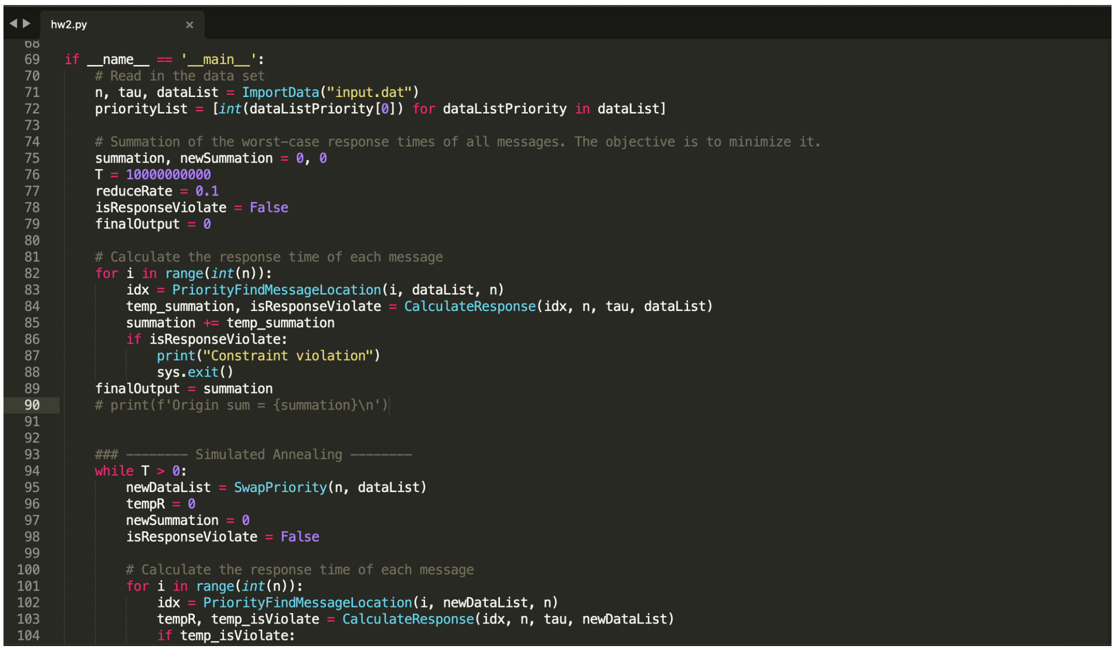
Code



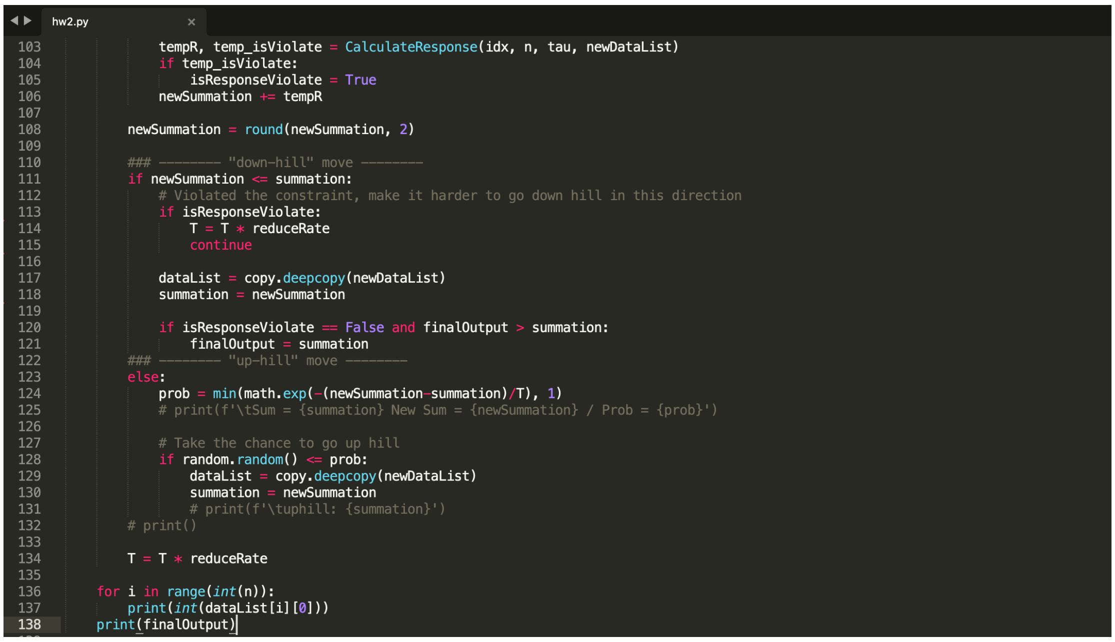
```python
import math
import random
import sys
import copy

# Read the input.dat file. Get the total number of the messages, tau, and lines contains the priority (Pi), the transmission time (Ci),
def ImportData(filename):
    n = open(filename).readlines()[:1]
    tau = open(filename).readlines()[1:2]
    n = int(''.join(n).strip())
    tau = float(''.join(tau).strip())
    dataList = [i.strip().split() for i in open(filename).readlines()[2:]] # read data to a 2D list
    dataList = [list(map(float, data)) for data in dataList] # turn type string to float

    return n, tau, dataList

# Find the message's index based on the priority
def PriorityFindMessageLocation(prior, localDataList, n):
    idx = 0
    for x in range(n):
        if localDataList[x][0] == prior:
            idx = x
    return idx


def CalculateResponse(index, n, tau, localDataList):
    Q, B, R = 0, 0, 0
    isViolate = False

    # Find the B value (blocking time of the longest lower or same priority message).
    for blockIndex in (lower for lower in [firstColumn[0] for firstColumn in localDataList] if lower >= localDataList[index][0]):
        location = PriorityFindMessageLocation(int(blockIndex), localDataList, n)
        if B < localDataList[location][1]:
            B = localDataList[location][1]

```



```python
    Q = B
    while True:
        sum = 0
        for blockIndex in (lower for lower in [firstColumn[0] for firstColumn in localDataList] if lower < localDataList[index][0]):
            location = PriorityFindMessageLocation(int(blockIndex), localDataList, n)
            sum += math.ceil((Q + tau) / localDataList[location][2]) * localDataList[location][1]

        if (B+sum)+localDataList[index][1] > localDataList[index][2] and not isViolate:
            # print("Constraint violation")
            isViolate = True

        if Q == B+sum:
            # worst-case R
            R = round((B+sum)+localDataList[index][1],2)
            # print(f'{ R }')
            break
        else:
            Q = B+sum

    return float(R), isViolate


# Swap two messages' priority
def SwapPriority(n, localDataList):
    tempDataList = copy.deepcopy(localDataList)
    # Random select two priority to swap
    x, y = random.sample(priorityList, 2)
    tempDataList[x][0], tempDataList[y][0] = tempDataList[y][0], tempDataList[x][0] # Swap two messages' priority
    # print(f'Swap {x} and {y}')
    return tempDataList



if __name__ == '__main__':
    # Read in the data set
    n, tau, dataList = ImportData("input.dat")
```



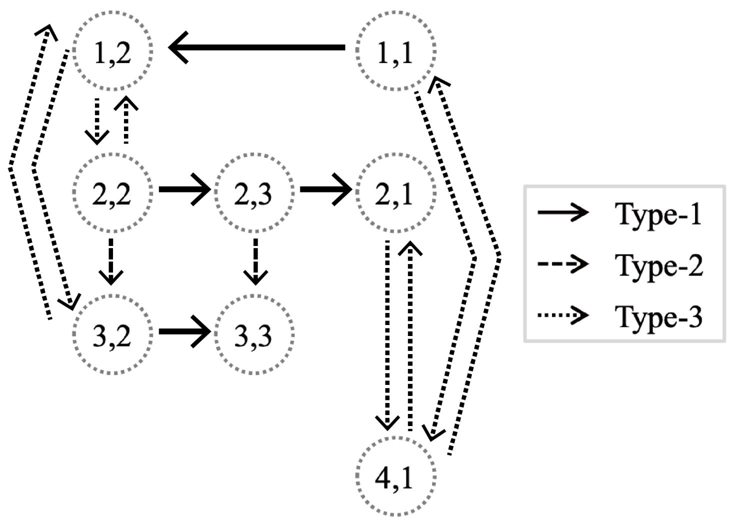
```python
if __name__ == '__main__':
    # Read in the data set
    n, tau, dataList = ImportData("input.dat")
    priorityList = [int(dataListPriority[0]) for dataListPriority in dataList]

    # Summation of the worst-case response times of all messages. The objective is to minimize it.
    summation, newSummation = 0, 0
    T = 10000000000
    reduceRate = 0.1
    isResponseViolate = False
    finalOutput = 0

    # Calculate the response time of each message
    for i in range(int(n)):
        idx = PriorityFindMessageLocation(i, dataList, n)
        temp_summation, isResponseViolate = CalculateResponse(idx, n, tau, dataList)
        summation += temp_summation
        if isResponseViolate:
            print("Constraint violation")
            sys.exit()
    finalOutput = summation
    # print(f'Origin sum = {summation}\n')


    ### -------- Simulated Annealing --------
    while T > 0:
        newDataList = SwapPriority(n, dataList)
        tempR = 0
        newSummation = 0
        isResponseViolate = False

        # Calculate the response time of each message
        for i in range(int(n)):
            idx = PriorityFindMessageLocation(i, newDataList, n)
            tempR, temp_isViolate = CalculateResponse(idx, n, tau, newDataList)
            if temp_isViolate:
```



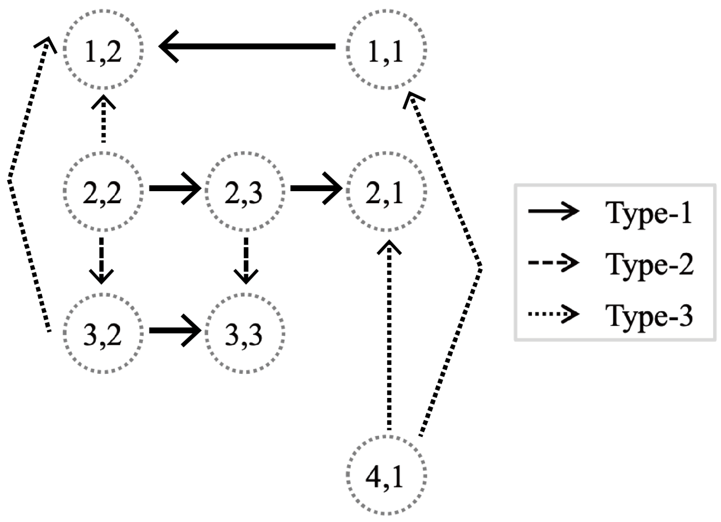
```python
            tempR, temp_isViolate = CalculateResponse(idx, n, tau, newDataList)
            if temp_isViolate:
                isResponseViolate = True
            newSummation += tempR

        newSummation = round(newSummation, 2)

        ### -------- "down-hill" move --------
        if newSummation <= summation:
            # Violated the constraint, make it harder to go down hill in this direction
            if isResponseViolate:
                T = T * reduceRate
                continue

            dataList = copy.deepcopy(newDataList)
            summation = newSummation

            if isResponseViolate == False and finalOutput > summation:
                finalOutput = summation
        ### -------- "up-hill" move --------
        else:
            prob = min(math.exp(-(newSummation-summation)/T), 1)
            # print(f'\tSum = {summation} New Sum = {newSummation} / Prob = {prob}')

            # Take the chance to go up hill
            if random.random() <= prob:
                dataList = copy.deepcopy(newDataList)
                summation = newSummation
                # print(f'\tuphill: {summation}')
        # print()

        T = T * reduceRate

    for i in range(int(n)):
        print(int(dataList[i][0]))
    print(finalOutput)
```

4. Intersection Management

Q1.



Q2.


1,2
1,1
2,2
2,3
2,1
3,2
3,3
4,1
Type-1
Type-2
Type-3