[[ 0.52 100. ]

[ 0.52 5. ]

[ 0.68 1000. ]

[ 0.52 1000. ]

[ 0.68 10. ]

[ 0.52 5. ]

[ 0.52 50. ]

[ 0.92 10. ]

[ 0.76 100. ]

[ 0.52 50. ]

[ 0.52 100. ]

[ 0.6 10. ]

[ 0.6 5. ]

[ 0.52 1000. ]

[ 0.52 10. ]

[ 0.6 5. ]

[ 0.6 5. ]]

optimum response time: 20.16

Source code

import numpy as np

import math

np.set\_printoptions(suppress=True)

signal\_num = 0

one\_bit\_trans = 0

total\_qi = 0

def calculate\_response(signal\_num, one\_bit\_trans, trans\_time, period\_time):

    """[summary]

    Args:

        signal\_num (int): signal numbers

        one\_bit\_trans (float): one bit trans time

        trans\_time (array): trans time array size depends on signal

        period\_time (array): message period time size => signal num

    Returns:

        worst response (float)

    """

    for i in range(signal\_num):

        block\_time = np.max(trans\_time[i:])

        high\_priority\_signal = trans\_time[:i]

        LHS = block\_time

        while 1:

            RHS = block\_time

            for j in range(len(high\_priority\_signal)):

                RHS += math.ceil((one\_bit\_trans + LHS)/period\_time[j])\*high\_priority\_signal[j]

            if (RHS == LHS) & (i != (signal\_num-1)) :

                # print("signal: %s response time: %s"%(i, (RHS + trans\_time[i])))

                break

            elif (RHS == LHS) & (i == (signal\_num-1)):

                # print("signal: %s response time: %s"%(i, (RHS + trans\_time[i])))

                return RHS + trans\_time[i]

            elif RHS >= LHS:

                LHS = RHS

            else:

                print("error in message %s"% (i))

                break

def get\_neighbor(message\_property):

    message\_property = message\_property[np.random.choice(range(signal\_num), signal\_num, replace=False)]

    return message\_property

def accept\_prob(delta\_cost, temperature):

    if delta\_cost < 0:

        return 1

    else:

        accept\_rate = np.exp(-(delta\_cost) / temperature)

        return accept\_rate

for idx, line in enumerate(open("input.dat", 'r')):

    item = line.rstrip()

    split\_item = item.split()

    if idx == 0:

        signal\_num = int(split\_item[0])

        trans\_time = np.zeros(signal\_num)

        period\_time = np.zeros(signal\_num)

    elif idx == 1:

        one\_bit\_trans = float(split\_item[0])

    else:

        trans\_time[int(split\_item[0])] = float(split\_item[1])

        period\_time[int(split\_item[0])] = float(split\_item[2])

message\_property = np.c\_[np.arange(signal\_num), trans\_time, period\_time]

temperature = 1

max\_step = 200

optimum\_state = message\_property

optimum\_cost = calculate\_response(len(message\_property), one\_bit\_trans, message\_property[:, 1], message\_property[:, 2])

print(temperature, optimum\_cost)

for step in range(max\_step):

    frac = step/max\_step

    T = temperature \* (1 - frac)

    new\_state = get\_neighbor(optimum\_state)

    new\_cost = calculate\_response(len(new\_state), one\_bit\_trans, new\_state[:, 1], new\_state[:, 2])

    if accept\_prob(new\_cost-optimum\_cost, T) > np.random.rand():

        optimum\_state, optimum\_cost = new\_state, new\_cost

        print(T,optimum\_cost)

print(optimum\_state[:, 1:])

print("optimum response time: %s"%(optimum\_cost))