1.

α	β	γ	LHS	$\alpha+\beta-\gamma\leq 1$	$\alpha-\beta+\gamma\leq 1$	$-\alpha+\beta+\gamma\leq 1$	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

2.

α	β	γ	LHS	$\alpha+\beta-1\leq\gamma$	$\gamma\leq\alpha$	$\gamma\leq\beta$	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

3.

β ==0:

if LHS==RHS==True:

Y==0 -> $0\leqq X\leqq 2019$

X-M $\leqq 0$ -> $X\leqq M$

if LHS==RHS==False:

Y>0 -> $Y\leqq 0$ always False

β ==1:

if LHS==RHS==True:

X==Y -> $X\leqq M$

if LHS==RHS==False:

X!=Y -> $Y\leqq X$ and $X\leqq Y$ always have 1 False

Because $X\leqq 2019$ and $X\leqq M$

M -> 2019

1.

Originally, μ0 needs 8+44+3=55 and μ1 needs 16+44+3=63 bits. Therefore, total 55+63=128 bits for 50 msec.

After redesigned, μ0' needs 16+44+3=63 bits for 50 msec. Because of 63 bits < 128 bits and with the same period, the new design is better.

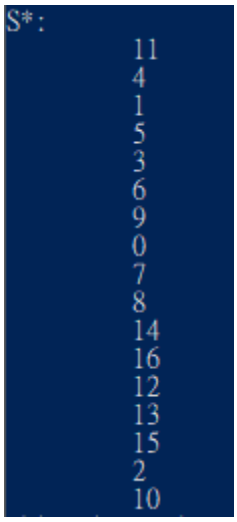
2.

The senders are different. They can't be merged.

3.

μ0' and μ2 have different sender, so they can't merge. However, μ0' and μ3 have the same sender, so they can be merged. Before merged, μ0' and μ3 take total 63*2+63=189 bits per 100 period. After merged, it takes total 79 bits per 50 msec. In this case, the period of original μ3 should be altered to 50 msec. In 100 msec, new design has 79*2=158 bits to be transmitted, and original design needs 189 bits. The performance is improved.

1.



2.

objective value: 204.11999999999995

3.

```
import numpy as np
import math
import time
import random

'''
worst case response time
'''
def worstCaseCal(n, C, tow, T, P):
    record = []
    for i in range(n):
        B = 0
        for j in range(n):
            if P[i] <= P[j] and B < C[j]:
                B = C[j]

        Q = B

        while True:
            counting = 0.0
            for j in range(n):
```

```
                    if P[j] < P[i]:
                        counting += math.ceil((Q+tow)/T[j])*C[j]
                if B+counting+C[i] > T[i]:
                    #print("not schedulable")
                    return []
                elif B+counting == Q:
                    #print(str(i)+":", Q+C[i])
                    record.append(Q+C[i])
                    break
                else:
                    Q = B+counting
        return record

'''
cal cost
'''
def cost(n, C, tow, T, P):
        return sum(worstCaseCal(n, C, tow, T, P))

'''
Read in data and pruning
'''
with open("input.dat") as f:
        data = f.read().split('\n')[:-1]
        n = int(data[0])
        tow = float(data[1])
        data = data[2:]
        P = []
        C = []
        T = []
        for i in data:
                tmp = i.split(' ')
                tmp2 = []
                for j in tmp:
                        if len(j) != 0:
                                tmp2.append(float(j))
                P.append(tmp2[0])
                C.append(tmp2[1])
```

```
            T.append(tmp2[2])
        P = np.array(P)
        C = np.array(C)
        T = np.array(T)
'''
config
'''
temp = 1000.0
Sstar = P.copy()
tempMin = 1.0
startTime = time.time()
iterCount = 100
k = 0.95

'''
Main code frame
'''
while (temp > tempMin) and (time.time() - startTime < 15):
    for ite in range(iterCount):
        currCost = cost(n, C, tow, T, P)
        Pnew = P.copy()
        index1 = random.randint(0, n-1)
        index2 = random.randint(0, n-1)
        tmp = Pnew[index1]
        Pnew[index1] = Pnew[index2]
        Pnew[index2] = tmp
        newCost = cost(n, C, tow, T, Pnew)
        if int(newCost) != 0:
            deltaCost = newCost - currCost
            if newCost < cost(n, C, tow, T, Sstar):
                Sstar = Pnew.copy()
            if deltaCost <= 0:
                P = Pnew.copy()
            else:
                prob = math.exp(-(deltaCost/temp))
                if np.random.choice([0,1],1,p=[prob,1-prob])[0] == 1:
                    P = Pnew.copy()
        else:
```

```
                #print('cons break')
                continue

        temp *= k
        print('next!', "time:", time.time() - startTime)

print('S*:')
for i in Sstar:
        print('\t', int(i))
print('objective value:', cost(n, C, tow, T, Sstar))
```