```
[[      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

```python
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))
```

CSIE 5452 HW2 P10942A05 鄭閔軒

1. MILP Linearization

1. Given α β γ which are binary variables

prove that $\alpha+\beta+\gamma \neq 2 \iff \alpha+\beta-\gamma \leq 1 \land \alpha-\beta+\gamma \leq 1 \land -\alpha+\beta+\gamma \leq 1$ (and, and)

α β γ	LHS	$\alpha+\beta-\gamma\leq1$	$\alpha-\beta+\gamma\leq1$	$-\alpha+\beta+\gamma\leq1$	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. prove $\alpha\beta = \gamma \iff \alpha+\beta-1 \leq \gamma \land \gamma \leq \alpha \land \gamma \leq \beta$

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. select M to guarantee

$\beta x = y \iff 0 \leq y \leq x \land x - M(1-\beta) \leq y \land y \leq M\beta$. $\quad x \leq 2021$

β	LHS	$0\leq y\leq x$	$x-M(1-\beta)\leq y$	$y\leq M\beta$	
0	$0=y$	$0\leq y\leq x$	$x-M\leq y$	$y\leq 0$	$\Rightarrow x-M\leq y=0\leq x$
1	$x=y$	$0\leq y\leq x$	$x\leq y$	$y\leq M$	$\Rightarrow 0\leq x=y\leq M$

In case $\beta=1$ $\overset{LHS}{}$ $x=y$ $\quad x-M\leq0$ and $x\leq2021$ $\quad M\geq x$

$\underline{M\geq2021}$ #

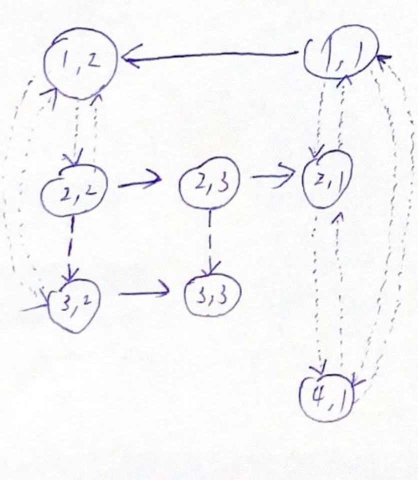
2. Signal Packing

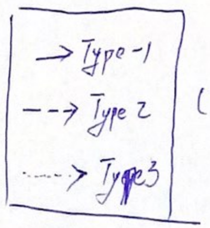
1. New design is better due to enhance the data efficiency, packing signal can reduce header and other bits message, so packing same period signal is better

2. we can know M_0' sender is E_0 and receiver is E_1 but M_2 sender is E_1 and Reciever is E_2, E_3, above two message have different sender and receiver so can't merge them.

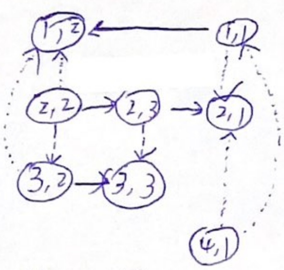
3. If we can seperate M_3 messages' sender and receiver to two part ($E_0 \to E_3$ to $E_0 \to E_1$ and $E_1 \to E_3$), then we can ~~merge~~ packing M_3 signal into M_0' and M_2 messages signal. Above action can enhance data efficiency and have more frequent messages to replace period time 100 msec M_3 message.

4. (1)





(2)



4 enter conflict 1

When vehicle 1 in conflict zone 1
vehicle 2 in conflict zone 3
vehicle 3 in conflict zone 2
will become dead lock