ItIV HW2

Problem 1

1.

α	β	γ	**LHS**	$\alpha+\beta-\gamma\leq 1$	$\alpha-\beta+\gamma\leq 1$	$-\alpha+\beta+\gamma\leq 1$	**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**	=
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.
When $\beta = 0$, we have $x - M \leq y = 0 \leq x$; therefore, $x \leq M$
When $\beta = 1$, we have $0 \leq y = x \leq M$; therefore, $x \leq M$
Since, $x \leq 2021$, M can be set as any number that is larger than or equal to 2021.

Problem 2

1.
Transmitting μ_0 requires 44+8+3=55 bits, and
transmitting μ_1 requires 44+16+3=63 bits. This totally requires 118 bits for transmitting per 50 ms.
While transmitting μ_0' only requires 44+16+3=63 bits per 50 ms.

2.
No, the sender are different for message μ_0' and μ_2.

3.
Yes, we can further improve the number of bits to be transmitted.
We design to sent μ_3 per 50ms so that it can be packed with μ_0'. Suppose the new packed message is μ_0'', the new design messages is as follows:

Message	Sender	Receiver	Number of Bits	Period
μ_0''	ε_0	$\varepsilon_1, \varepsilon_3$	32	50
μ_2	ε_1	$\varepsilon_2, \varepsilon_3$	10	50

In this new design messages, it requires $32 + 44 + 3 = 79$ bits for message μ_0''. Totally trasmitting 158 bits per 100 ms.
While the previous design messages requires $(16 + 44 + 3) \times 2 = 189$ bits per 100 ms for transmitting μ_0' and μ_3.

Problem 3

1.
There are many prioirty assigments that come up with the same objective value. One of the possible priority is as follows :

```
14
4
0
6
1
5
9
2
7
8
11
15
3
10
16
13
12
```

2.

The objective value of the previous solution is 204.12.

3.

Our source code is writen in MATLAB and is shown as follows:

```
tic
filename = "input.dat";

fildID = fopen(filename, 'r');

n_task = str2num(fgetl(fildID));
tau = str2num(fgetl(fildID));
data = fscanf(fildID, "%f %f %f\n", [3 Inf]);

temp = 100;
r = 0.5;
global n_task tau data response_time;
CAN_timing_analysis(); best_data = data; best_response_time = sum(response_time);
prev_response_time = inf;
for n_iter = 1 : 1000
    SA_iter(0);
    valid = CAN_timing_analysis();
    if(valid && (sum(response_time) < sum(prev_response_time)))
        best_data = data; best_response_time = sum(response_time);
        prev_response_time = response_time;
    else
        if(~valid) SA_iter(1);
        elseif(sum(response_time) > sum(prev_response_time))
            p = exp((sum(prev_response_time)-sum(response_time))/temp);
            if(p > 1) p = 1; end
            if(rand(1)<p) prev_response_time = response_time;
            else SA_iter(1);
            end
        end
    end
    temp = r * temp;
end
fprintf("%d\n",best_data(1,:))
% fprintf("%d %f %f\n",best_data)
CAN_timing_analysis(); sum(response_time)
toc

function SA_iter(reverse)
    global n_task data;
    persistent idx_1 idx_2;
    if(reverse)
        data(1,[idx_1 idx_2]) = data(1,[idx_2 idx_1]);
    else
        idx_1 = randi(n_task);
        idx_2 = randi(n_task);
        if(idx_2 ~= idx_1) data(1,[idx_1 idx_2]) = data(1,[idx_2 idx_1]); end
    end
end


function valid = CAN_timing_analysis()
    global n_task tau data response_time;
```

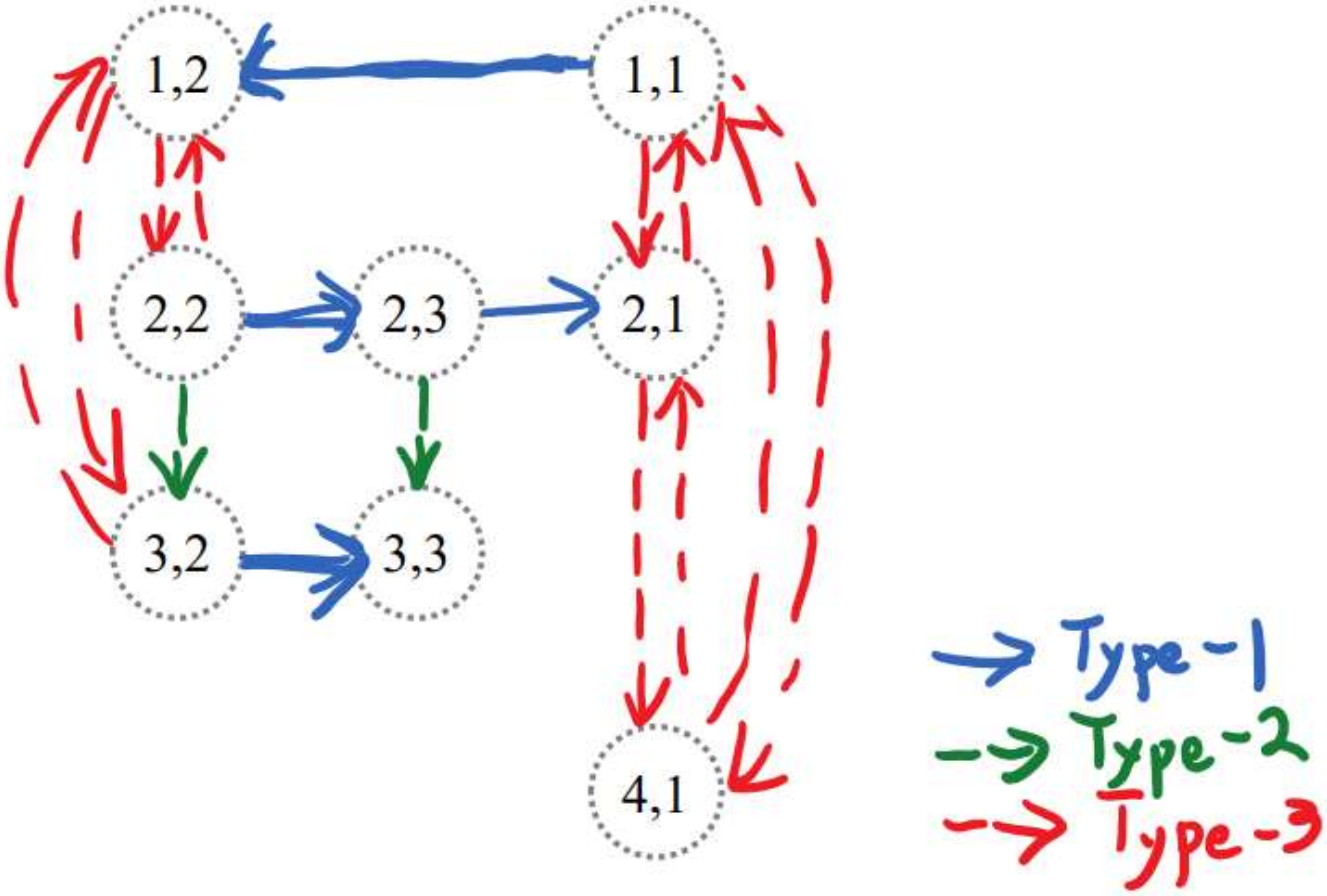
```
    priority = data(1,:);
    [~, permut] = sort(priority);
    time = data(2,permut);
    period = data(3,permut);
    response_time = [];
    for i_task = 1 : n_task
        block_time = max(time(i_task:end));
        Q = block_time;
        while true
            if(i_task == 1) RHS = block_time;
            else RHS = block_time + sum(ceil((Q+tau)./period(1:i_task-1)).*time(1:i_task-1)
            end
            if(Q == RHS) response_time(i_task) = Q + time(i_task); break; end
            Q = RHS;
        end
    end
    valid = (prod(period > response_time) == 1);
end
```

We've run the above source code for 100 times.
The mean of the program runtime is 0.1177 second.
Also, the objective value of all program are all 204.12.

Problem 4

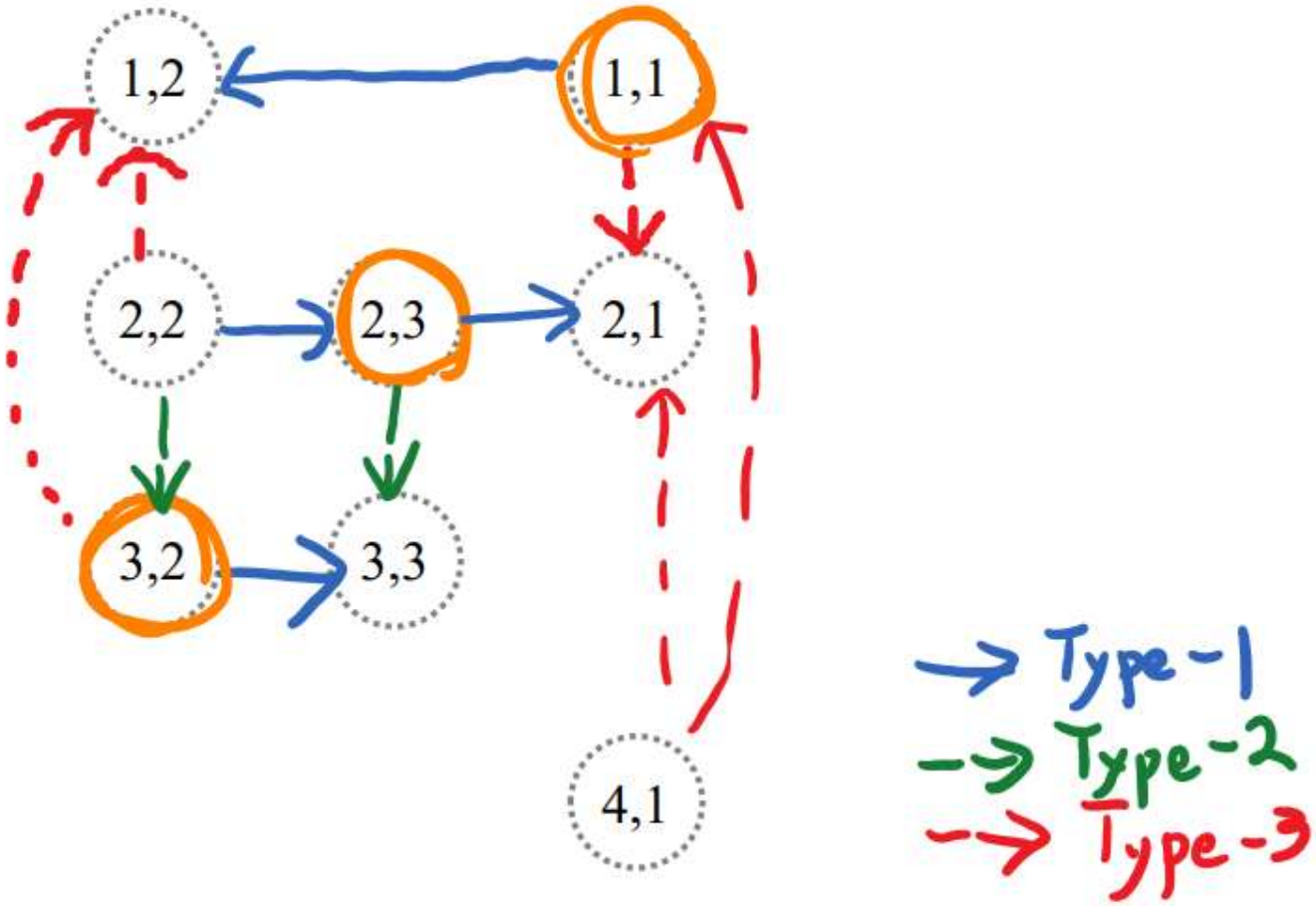
1.

The corresponding timing conflict graph



2.

The following timing conflict graph has no loops, but there is a DEADLOCK.



When Car 1, 2, 3 are at conflict zone 1, 3, 2 (the nodes are marked as orange).
For car 1, it requires waiting car 3 to pass conflict zone 2.
For car 2, it requires waiting car 1 to pass conflict zone 1.
For car 3, it requires waiting car 2 to pass conflict zone 3.
Therefore, three cars are all stuck and form a DEADLOCK!