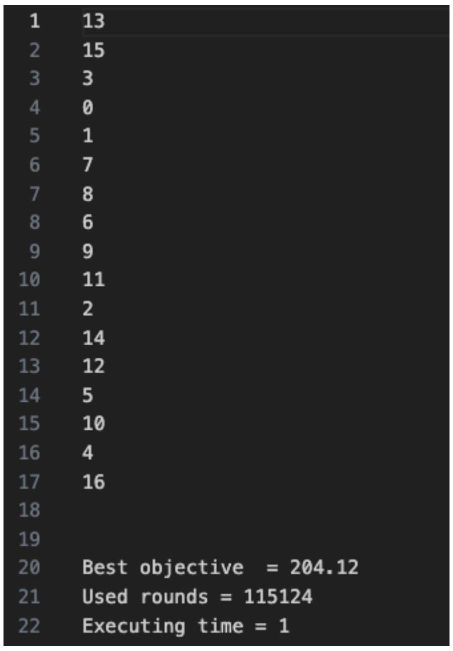
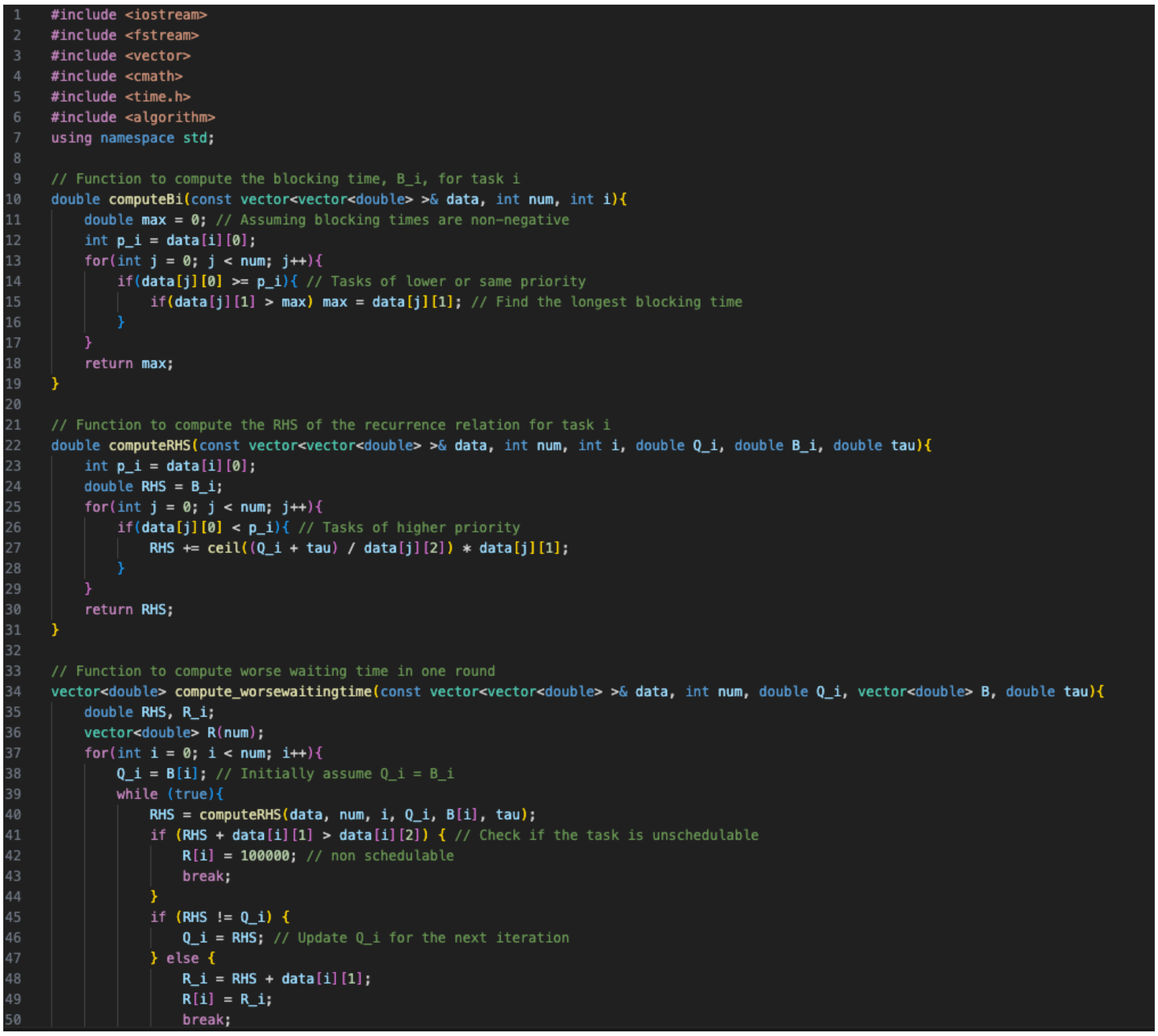
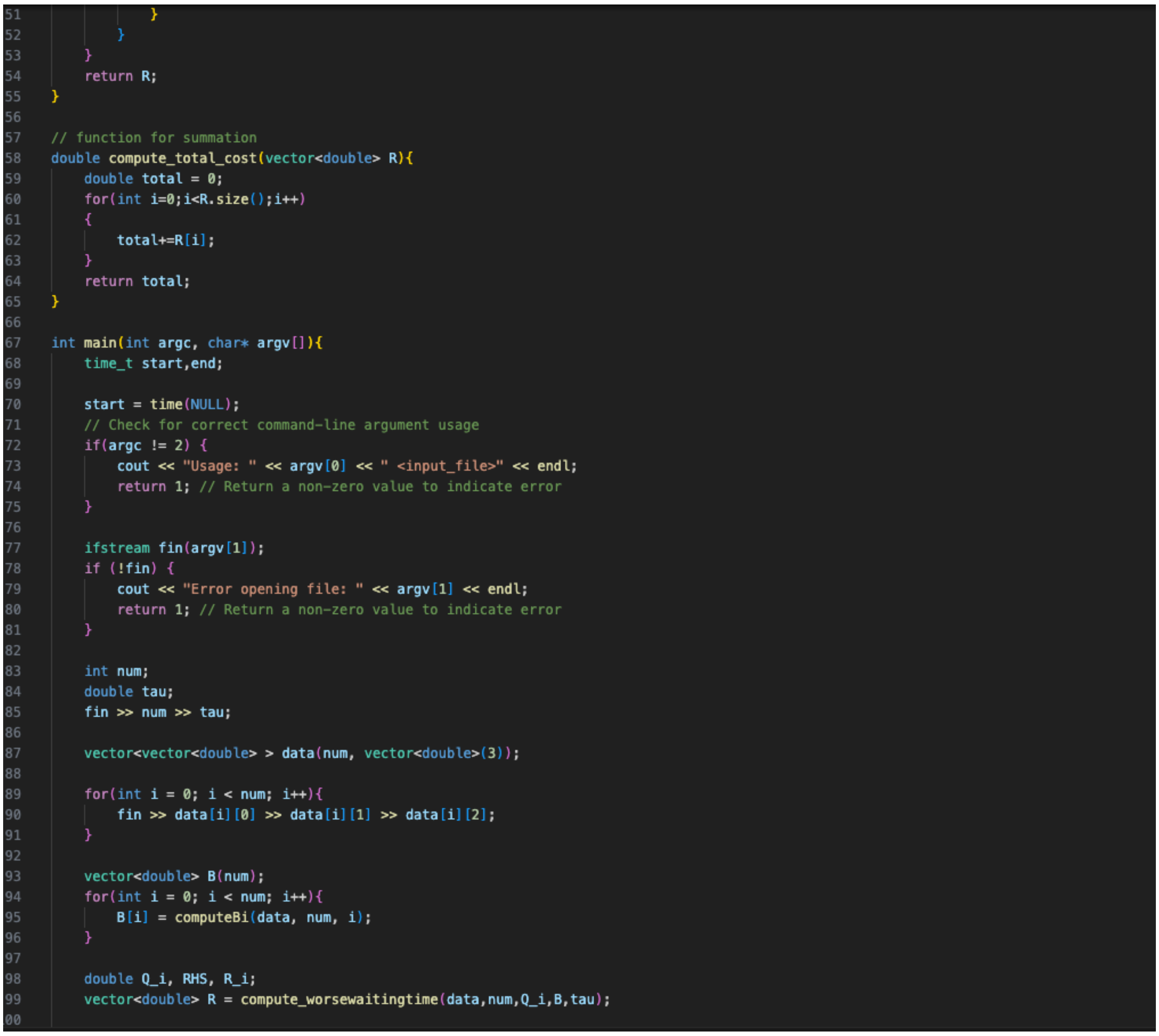


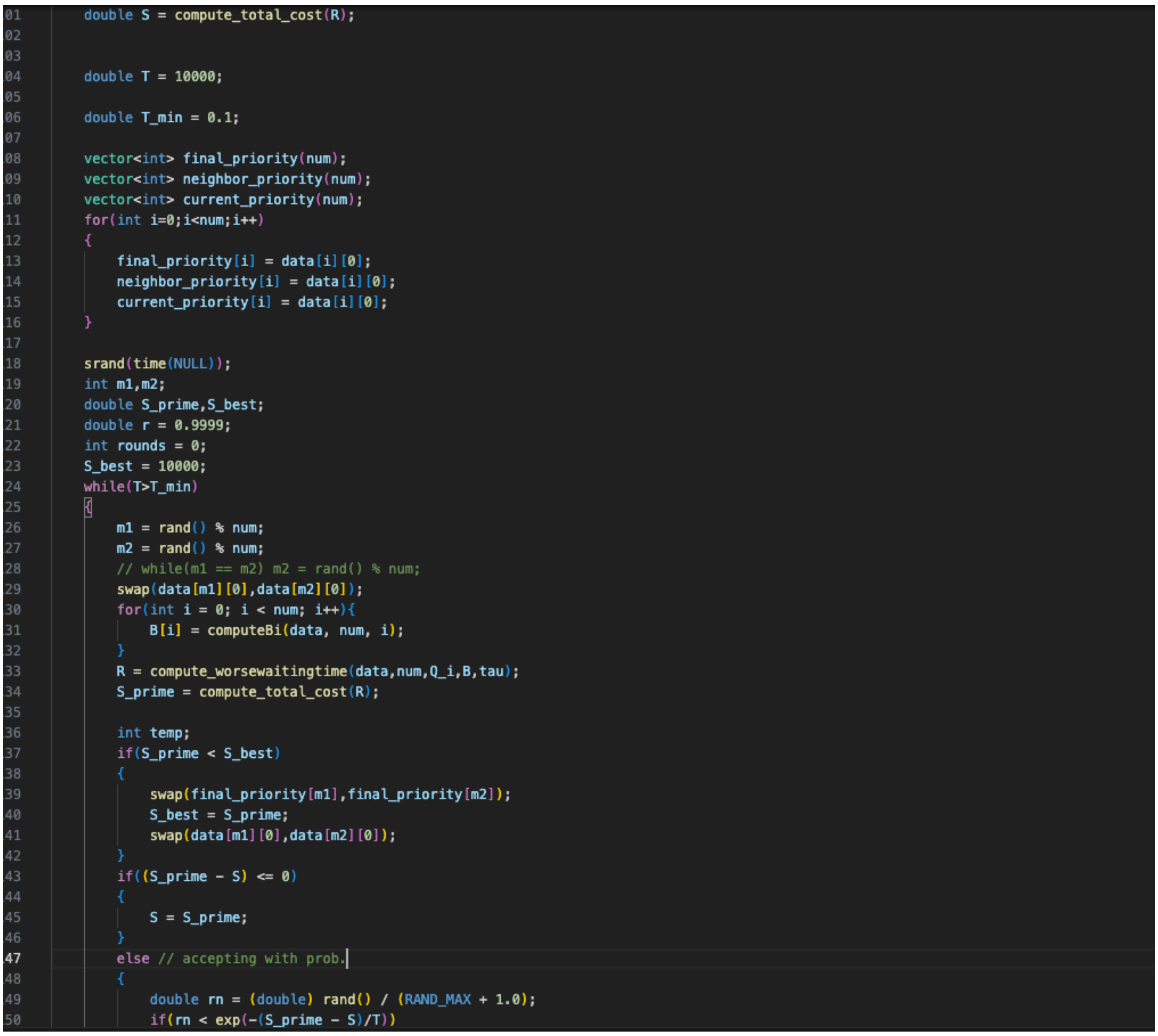
```
13
15
3
0
1
7
8
6
9
11
2
14
12
5
10
4
16


Best objective  = 204.12
Used rounds = 115124
Executing time = 1
```



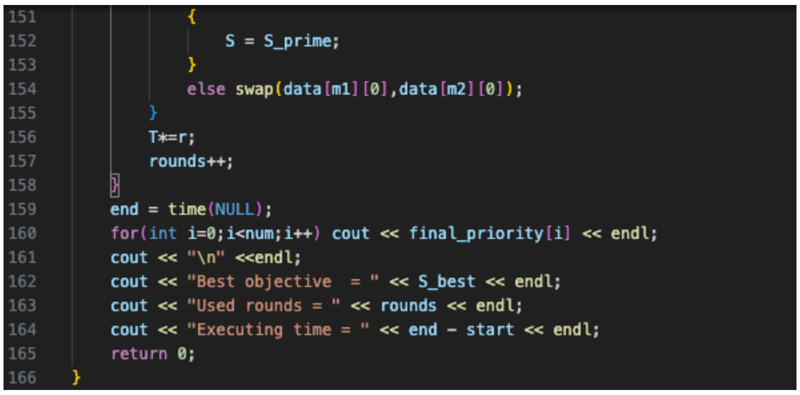
```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include <time.h>
#include <algorithm>
using namespace std;

// Function to compute the blocking time, B_i, for task i
double computeBi(const vector<vector<double> >& data, int num, int i){
    double max = 0; // Assuming blocking times are non-negative
    int p_i = data[i][0];
    for(int j = 0; j < num; j++){
        if(data[j][0] >= p_i){ // Tasks of lower or same priority
            if(data[j][1] > max) max = data[j][1]; // Find the longest blocking time
        }
    }
    return max;
}

// Function to compute the RHS of the recurrence relation for task i
double computeRHS(const vector<vector<double> >& data, int num, int i, double Q_i, double B_i, double tau){
    int p_i = data[i][0];
    double RHS = B_i;
    for(int j = 0; j < num; j++){
        if(data[j][0] < p_i){ // Tasks of higher priority
            RHS += ceil((Q_i + tau) / data[j][2]) * data[j][1];
        }
    }
    return RHS;
}

// Function to compute worse waiting time in one round
vector<double> compute_worsewaitingtime(const vector<vector<double> >& data, int num, double Q_i, vector<double> B, double tau){
    double RHS, R_i;
    vector<double> R(num);
    for(int i = 0; i < num; i++){
        Q_i = B[i]; // Initially assume Q_i = B_i
        while (true){
            RHS = computeRHS(data, num, i, Q_i, B[i], tau);
            if (RHS + data[i][1] > data[i][2]) { // Check if the task is unschedulable
                R[i] = 100000; // non schedulable
                break;
            }
            if (RHS != Q_i) {
                Q_i = RHS; // Update Q_i for the next iteration
            } else {
                R_i = RHS + data[i][1];
                R[i] = R_i;
                break;
```



```cpp
            }
        }
    }
    return R;
}

// function for summation
double compute_total_cost(vector<double> R){
    double total = 0;
    for(int i=0;i<R.size();i++)
    {
        total+=R[i];
    }
    return total;
}

int main(int argc, char* argv[]){
    time_t start,end;

    start = time(NULL);
    // Check for correct command-line argument usage
    if(argc != 2) {
        cout << "Usage: " << argv[0] << " <input_file>" << endl;
        return 1; // Return a non-zero value to indicate error
    }

    ifstream fin(argv[1]);
    if (!fin) {
        cout << "Error opening file: " << argv[1] << endl;
        return 1; // Return a non-zero value to indicate error
    }

    int num;
    double tau;
    fin >> num >> tau;

    vector<vector<double> > data(num, vector<double>(3));

    for(int i = 0; i < num; i++){
        fin >> data[i][0] >> data[i][1] >> data[i][2];
    }

    vector<double> B(num);
    for(int i = 0; i < num; i++){
        B[i] = computeBi(data, num, i);
    }

    double Q_i, RHS, R_i;
    vector<double> R = compute_worsewaitingtime(data,num,Q_i,B,tau);

```



```cpp
    double S = compute_total_cost(R);


    double T = 10000;

    double T_min = 0.1;

    vector<int> final_priority(num);
    vector<int> neighbor_priority(num);
    vector<int> current_priority(num);
    for(int i=0;i<num;i++)
    {
        final_priority[i] = data[i][0];
        neighbor_priority[i] = data[i][0];
        current_priority[i] = data[i][0];
    }

    srand(time(NULL));
    int m1,m2;
    double S_prime,S_best;
    double r = 0.9999;
    int rounds = 0;
    S_best = 10000;
    while(T>T_min)
    {
        m1 = rand() % num;
        m2 = rand() % num;
        // while(m1 == m2) m2 = rand() % num;
        swap(data[m1][0],data[m2][0]);
        for(int i = 0; i < num; i++){
            B[i] = computeBi(data, num, i);
        }
        R = compute_worsewaitingtime(data,num,Q_i,B,tau);
        S_prime = compute_total_cost(R);

        int temp;
        if(S_prime < S_best)
        {
            swap(final_priority[m1],final_priority[m2]);
            S_best = S_prime;
            swap(data[m1][0],data[m2][0]);
        }
        if((S_prime - S) <= 0)
        {
            S = S_prime;
        }
        else // accepting with prob.
        {
            double rn = (double) rand() / (RAND_MAX + 1.0);
            if(rn < exp(-(S_prime - S)/T))
```



```cpp
            {
                S = S_prime;
            }
            else swap(data[m1][0],data[m2][0]);
        }
        T*=r;
        rounds++;
    }
    end = time(NULL);
    for(int i=0;i<num;i++) cout << final_priority[i] << endl;
    cout << "\n" <<endl;
    cout << "Best objective  = " << S_best << endl;
    cout << "Used rounds = " << rounds << endl;
    cout << "Executing time = " << end - start << endl;
    return 0;
}
```

# CSIE 5452, Spring 2024: Homework 2

Due April 2 (Tuesday) at Noon

When you submit your homework, select the corresponding page(s) of each question. Points will be deducted if no appropriate intermediate step is provided.

## 1 Simulated Annealing for Priority Assignment (40pts)

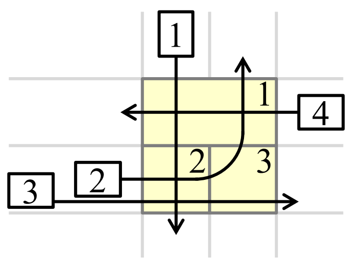
Please download the benchmark "input.dat" from NTU COOL. In the benchmark, the first number is $n$, the number of messages. The second number is $\tau$. Each of the following lines contains the priority ($P_i$), the transmission time ($C_i$), and the period ($T_i$) of each message. Now, you are asked to use the Simulated Annealing to decide the priority of each message. The requirements are:

- The objective is to minimize the summation of the worst-case response times of all messages.
- The priority of each message must be an integer in the range $[0, n-1]$.
- The priority of each message must be unique.
- The worst-case response time of each message must be smaller than or equal to the period of each message.
- The given priorities are the initial solution in the Simulated Annealing.
- We expect the total runtime less than 15 seconds.

You are required to do three things in your submission:

1. You should print out $n$ numbers (one number per line) representing the priorities of those messages. Note that you need to follow the message ordering in the benchmark, *e.g.*, the first number in the list is the priority of the first message in the benchmark.
2. You should print out 1 number representing your objective value (best one during your run).
3. You should also print out your source codes. We may ask you to provide your source codes which must be the same as those on your printout. If the worst-case response times above are correct but the source codes are clearly wrong implementation, it is regarded as academic dishonesty.

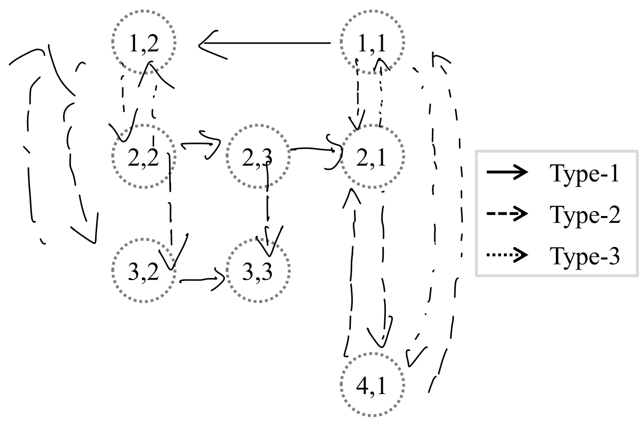
## 2 Intersection Management: Part I (20pts)



1. (8pts) Given the scenario in the figure above, follow the legend and draw the corresponding timing conflict graph.

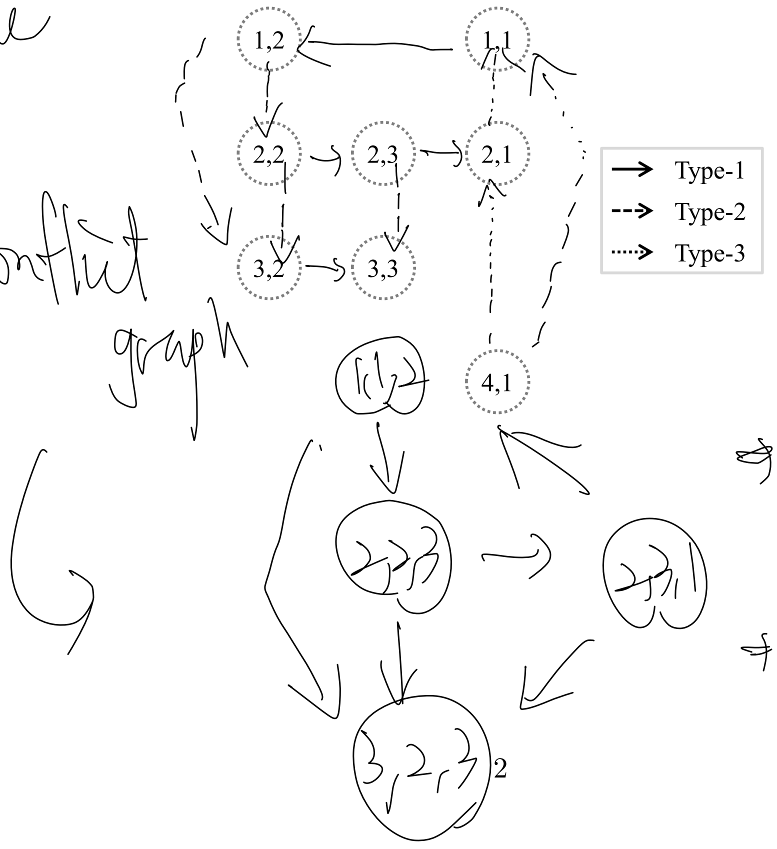
(1,2) → (2,2)

(2,1) → (1,1)



2. (12pts) Following 1., given that Vehicle 4 enters Conflict Zone 1 before Vehicles 1 and 2, find a DEADLOCK solution which has no cycle in the corresponding timing conflict graph. Follow the legend and draw the corresponding timing conflict graph. Explain why there is a deadlock.

Observe the resulting resource conflict graph



⇒ there is cycle in resource conflict graph

⇒ deadlock occurs.

## 3 Intersection Management: Part II (8pts)

In the lecture, we introduced the *timing conflict graph* $G$ to model the intersection management problem. We can remove some edges in $G$ to get an acyclic graph $G'$ as the solution of the problem. However, we need to build and verify the corresponding *resource conflict graph* $H'$ of $G'$: if $H'$ is acyclic, then there is no deadlock in the solution $G'$. Please explain what will happen in the special case that the whole intersection is modeled as one single conflict zone. How will you solve the special case?

## 4 Intersection Management: Part III (8pts)

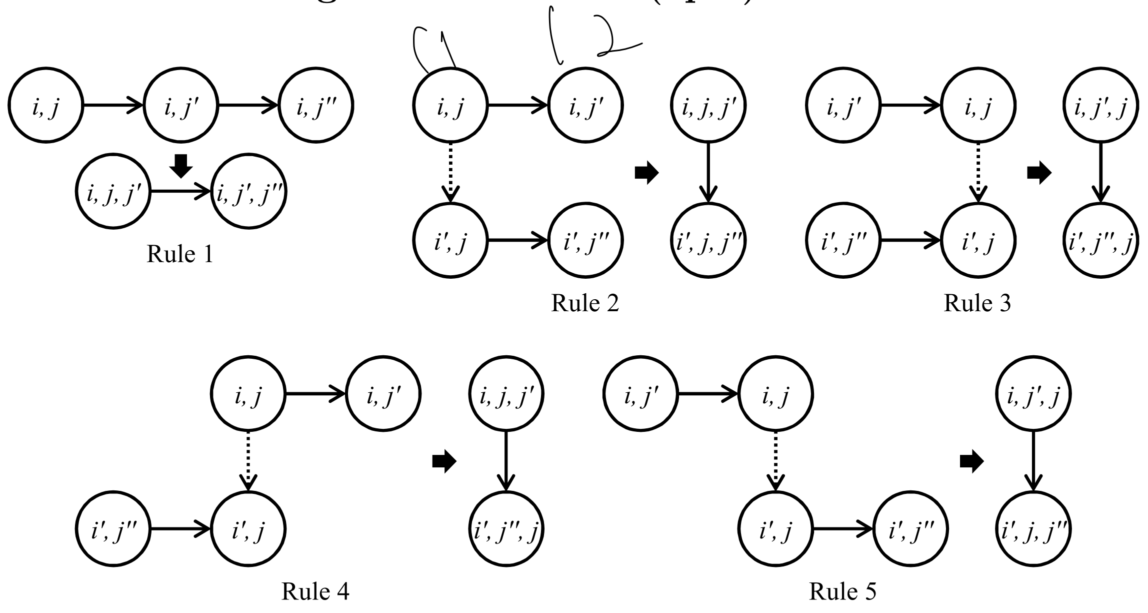


Figure 1: The construction rules.

In the lecture, we introduced the construction rules from the *timing conflict graph* to the *resource conflict graph*, as shown in Figure 1. With some conditions, we can use only two rules and ignore the other three rules for deadlock-freeness verification. Explain the conditions, identify the two rules, and discuss the benefits.

## 5 Realization of Level-X Autonomy (24pts)

In your opinion, when will be level-3/level-4/level-5 autonomy become realized? There will be no correct answers to these questions, and you can also answer them from many different perspectives including technology, cost, regulation, law, and human comfort. However, you should justify your answers with some explanation (*e.g.*, few sentences for each level). (This question will be graded by a letter grade with default grade A.)

1. (8pts) Level 3.
2. (8pts) Level 4.
3. (8pts) Level 5.

$Q_3$

if the whole intersection is modeled as a single zone
⇒ only 1 vehicle can pass in 1 time slot

using conflict graph to explain
⇒ there will be only type 2, type 3 edges.
by removing edges, because type 3 edges always have
choices ⇒ resulting timing conflict cycle G' won't
have cycles ⇒ won't be deadlocks

$Q_4$

Conditions: If the requests from vehicles follow
the order that they enter the intersection

Rules: Then we only need to use Rule 1 & 2

Benefits: Faster computation and Verification

**Q5**

**1. Level 3 Autonomy**

- I think the realization of Level 3 autonomy could take 5-10 years. Observing the ongoing investments in AI and sensor technology, gradual regulatory clarifications, I believe a broader realization of Level 3 would be implemented. However, the full utilization of Level 3 autonomy depends on regulatory approval across different regions, which has been a slow process.

**2. Level 4 Autonomy**

- I think achieving the realization of Level 4 autonomy could take 10-20 years. The major technological obstacle is huge. Also, more comprehensive regulatory frameworks need to be standardized internationally. For example, the infrastructure to support Level 4 autonomy, such as V2X communication, needs strict and serious development.

**3. Level 5 Autonomy**

- I guess the realization of Level 5 autonomy could be several decades away. This level of autonomy faces profound challenges, not only in technology and infrastructure but also in ethical, legal, and societal domains. The main and most challenging part is to connect the ADAS required. Also, how to realize such powerful autonomy while ensuring a reasonable cost that customers can accept is also a huge challenge.