2024 Introduction to Intelligent Vehicles HW2

StudentID : R12922054

[1] Simulated Annealing for Priority Assignment

1

Ans :

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

2

Ans :

```
Best objective value : 204.12
```

3

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <cmath>
#include <stdlib.h>
#include <algorithm>
using namespace std;

#define TEMPERATURE         1000.0
#define COOL_RATE           0.9
#define ANNEALING_INTERVAL  20

struct Msg{
    int     id;
    int     priority;
    double  transmissionTime;
    double  period;
};
```

```
double costFun_ResponseTime(int numOfMsg, vector<Msg>& vecMsg, double tau);
bool cmpPriorty(Msg A, Msg B);

int main(int argc, char *argv[])
{
    srand(time(NULL));
    string fileName = argv[1];
    ifstream file(fileName);

    vector<Msg> vecMsg;
    int     numOfMsg;
    double  tau;
    string  line;
    file >> numOfMsg >> tau;

    int id_Msg = 0;
    while(id_Msg < numOfMsg){
        Msg msg;
        msg.id = id_Msg;
        file >> msg.priority >> msg.transmissionTime >> msg.period;
        vecMsg.push_back(msg);
        id_Msg++;
    }
    file.close();

    int trial_count = 10;
    while(trial_count--){
        double temperature  = TEMPERATURE;
        double coolRate     = COOL_RATE;
        int annealingInterv = ANNEALING_INTERVAL;
        int round           = 0;
        vector<Msg> S;
        vector<Msg> S_star;
        S      = vecMsg;
        S_star = S;
        while( temperature >  1e-8 ){
            vector<Msg> S_prime = S;
            int random_idx1 = rand() % numOfMsg;
            int random_idx2;
            do{
                random_idx2 = rand() % numOfMsg;
            }while(random_idx1 == random_idx2);

            // swap priority
            int tmpPriority;
            tmpPriority = S[random_idx1].priority;
            S_prime[random_idx1].priority = S_prime[random_idx2].priority;
            S_prime[random_idx2].priority = tmpPriority;
            // cost
            double costS       = costFun_ResponseTime(numOfMsg, S,       tau);
            double costS_prime = costFun_ResponseTime(numOfMsg, S_prime, tau);
            double costS_star  = costFun_ResponseTime(numOfMsg, S_star,  tau);
            // update new best solution
            if(costS_prime < costS_star){
                S_star = S_prime;
            }

            double delta_cost = costS_prime - costS;
            if(delta_cost <= 0){
                S = S_prime;
            }
            else{
                double random_number = rand() % 100 / 100;
                double prob          = exp(-delta_cost /temperature);
                if( random_number < prob){
                    S = S_prime;
                }
            }
            // update info
            round += 1;
            if( round % annealingInterv == 0){
                temperature *= coolRate;
            }
        }
        cout << "Msg order in best objective value" << endl;
        sort(S_star.begin(), S_star.end(), cmpPriorty);
        for(auto& msg : S_star){
            cout << msg.id << endl;
        }
        cout << "Best objective value : "<< costFun_ResponseTime(numOfMsg, S_star, tau) << endl;
    }
}
double costFun_ResponseTime(int numOfMsg, vector<Msg>& vecMsg, double tau)
{
    double responseTime[numOfMsg] = {0};
    double Q[numOfMsg]            = {0};
    double B[numOfMsg]            = {0};
    double notScheduable_pay      = 1000;
    double cost                   = 0;

    // Compute block time of each Msg
    for(int id = 0; id < numOfMsg; id++){
        for(auto& msg : vecMsg){
            if(msg.priority >= vecMsg[id].priority && msg.transmissionTime > B[id]){
                B[id] = msg.transmissionTime;
            }
        }
        Q[id]= B[id];
    }

    for(int id = 0; id < numOfMsg; id++){
        while(1){
            double rhs = 0.0;
            rhs += B[id];
            // Compute RHS
            for(auto& msg : vecMsg){
                if(msg.priority < vecMsg[id].priority){
                    rhs += ceil( (Q[id]+tau)/msg.period ) * msg.transmissionTime;
                }
            }
            // Check RHS
            if(rhs + vecMsg[id].transmissionTime > vecMsg[id].period){
                responseTime[id] = notScheduable_pay;
                break;
            }
            else if(Q[id] == rhs){
                responseTime[id] = Q[id] + vecMsg[id].transmissionTime;
                break;
            }
            else{
                Q[id] = rhs;
            }
        }
    cost += responseTime[id];
    }

    return cost;
}

bool cmpPriorty(Msg A, Msg B)
{
    return A.priority < B.priority;
}
```

[2] Intersection Management: Part I

1

Ans :

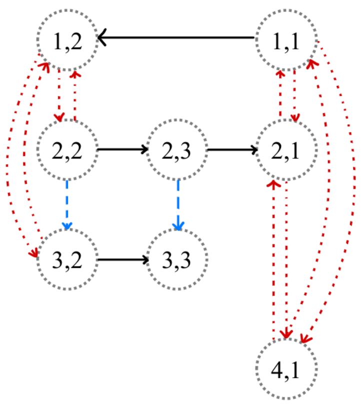

Figure 1: answer2-1.

2

Ans :

I construct resource conflict graph, Figure 3, from timing conflict graph, Figure 2. We can prove that there is a deadlock due to a cycle from resource conflict graph.

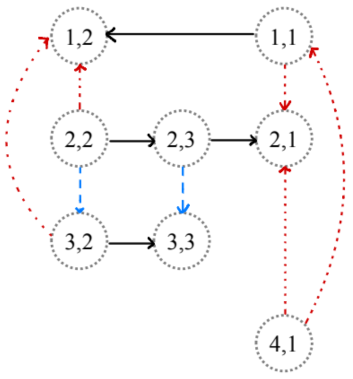

Figure 2: answer2-2_timing conflict graph.

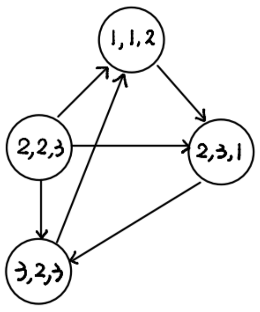

Figure 3: answer2-2_resource conflict graph.

[3] Intersection Management: Part II

Ans :

If the whole intersection is modeled as one single conflict zone, it will not generate deadlock but it might cause enormous waiting queue.

And, the way to solve this special case is that giving every vehicle the priority order. If the vehicle possesses the highest priority, it can enter the conflict zone than any other vehicle. However, if there is the condition that some of the vehicle possess the same priority, intersection manager need to randomly pick one of them to enter conflict zone.

[4] Intersection Management: Part III

Ans :

for the vertices(i ,j) and (i', j), if j is not in first conflict zone and last conflict zone, we can just use Rule 1 and Rule 4 for deadlock-freeness verification.(we can discover that Rule 2 can be used for verifying first conflict zone and Rule 3 can be used for verifying last conflict zone.)
In other words, if the conflict zone is divided into the following diagram, Figure 4, it can be implemented through Rule 1 and Rule 4.

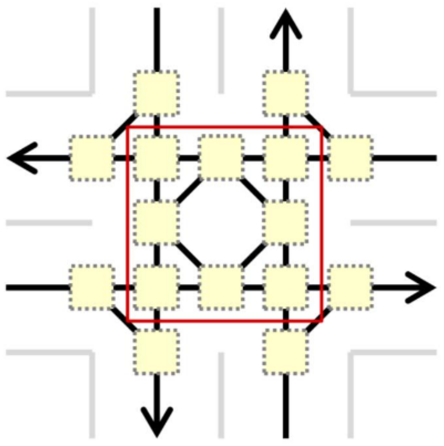

Figure 4: example.

]

The benefits are having fewer edges in the resource conflict zone and improving computational efficiency.

[5] Realization of Level-X Autonomy

1. Level 3

Ans :

Regarding L3 level autonomous driving, I think it is already underway, but this is only limited to special conditions in specific areas. If it is to be popularized, I think it will be around 2030. (My definition of popular here is a usage rate of more than 50% in advanced cities.) Currently, countries such as Germany and Japan have legislation allowing the deployment of Level 3 capabilities. The United Nations Economic Commission for Europe stipulates that the introduction of Level 3 autonomous driving in private vehicles must not exceed 60 kilometers per hour, with plans to expand this to higher speeds, potentially reaching 130 kilometers per hour from 2023[1] (2023, McKinsey & Company).

2. Level 4.

Ans :

Regarding L4 level autonomous driving, I think it is in the development stage, and indeed some autonomous driving development companies have already begun actual testing in specific areas. However, I speculate that it will take around 2028 before a small number of consumers will start trying it. After all, L4 level autonomous driving must not only obtain legal consent, but also must be 100% safe under specific circumstances. As for

[1]https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected

the time point of popularization, I think it will be between 2035-2038.
These speculations are personal opinions based on the following information:[2] (2023, McKinsey & Company) and[3] (2024, PrecisionPulse)

3. Level 5.

Ans :

Regarding L5 autonomous driving, I think even though some development companies are currently conducting relevant research or testing, I think it will take a long, long time before it can be applied on actual roads. To be pessimistic, I think L5 autonomous driving may never become popular because it is impossible to achieve 100% safety. In addition to the different road conditions in each region, we must also deal with the corresponding weather changes in each region, for example, snow, heavy rain, etc. There are even earthquakes. Although it is expected that you can choose to park if you encounter a situation that cannot be handled, but I don't think this can fundamentally solve the problem. Therefore, based on the above, I think the level that can actually be achieved is 80% complete autonomous driving, or 20% requires human control.
These speculations are personal opinions based on the following information:[4] (2022, Alex Roy)

[2]https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected

[3]https://www.linkedin.com/pulse/l4-autonomous-driving-market-report-2031-key-insights-0r61f/

[4]https://www.linkedin.com/pulse/why-l5-autonomous-vehicles-make-sense-alex-roy