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)

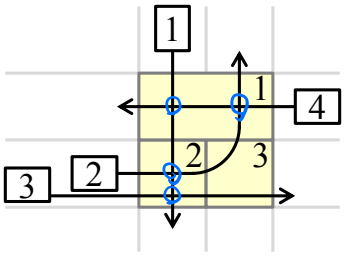
Please download the benchmark "input.dat" from NTU COOL. In the benchmark, the first number is n, the number of messages. The second number is τ. Each of the following lines contains the priority (P_i), the transmission time (C_i), and the period (T_i) of each message. Now, you are asked to use the Simulated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

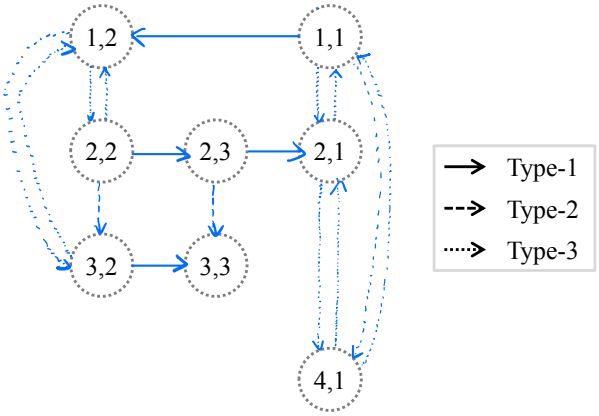
You are required to do three things in your submission:

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.

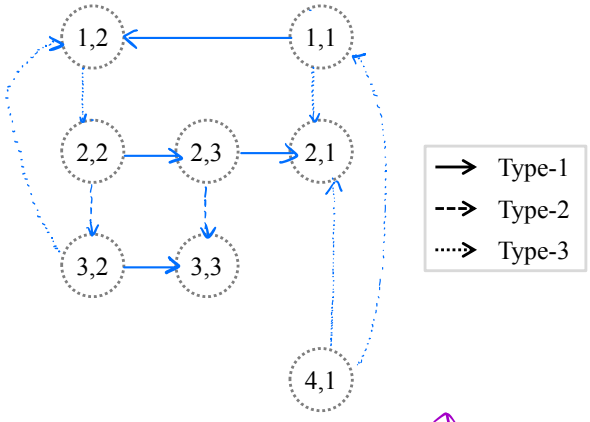
2 Intersection Management: Part I (20pts)



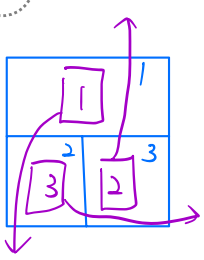
1. (8pts) Given the scenario in the figure above, follow the legend and draw the corresponding timing conflict graph.



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.



1 在 zone 1，想進 zone 2，waiting 3
2 在 zone 3，想進 zone 1，waiting 1
3 在 zone 2，想進 zone 3，waiting 2
4 已離開



⇒ 互相 waiting ⇒ DEADLOCK #

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?

此時的 G 不會有 type I 的 edge ⇒ H 沒有任何 vertex & edge ⇒ H 中 no cycle ⇒ no DEADLOCK #

4 Intersection Management: Part III (8pts)

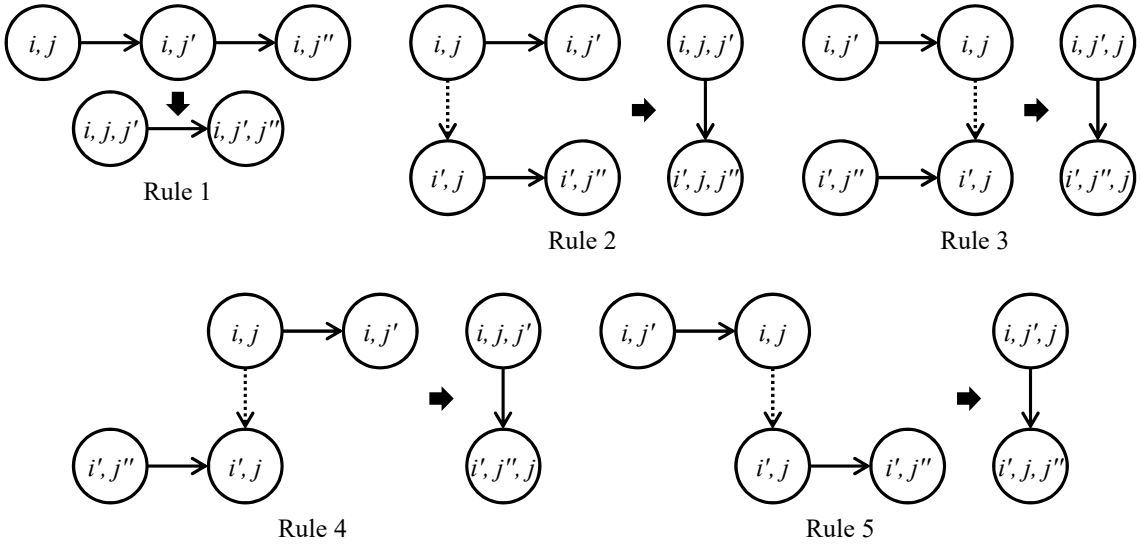


Figure 1: The construction rules.

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.

condition : 只要用 rule 1 & 4 ∵ 1 代表 1 台車的軌跡 ; 4 代表 zone j 要先被使用完畢才可被下一個使用 #

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.

1.

可在人駕駛與自駕之間切換，人需在需要時接手，目前已實現了，只是有「切換」要多早之前提醒 driver、如何判斷 driver ready for 接手…等等問題。

在 human comfort 與 safety 的議題未完全解決，且法律層面也未有關於自駕責任歸屬的適當規範。

2.

在某些地方可完全自駕，人不用準備接手，只是在哪些地方並未明確定義。

我認為在科技上已實現了，只是礙於責任歸屬問題，沒有車廠敢讓駕駛完全不用準備接手。

3.

在任何地方皆可完全自駕，科技應還不能完全在保障 safety 的情況下達成，因為太多特殊情況和難以預料的情形了。

這對 human comfort 來說是最好的，完全不用消耗心力在載具上。

法律層面仍是最大的問題，責任歸屬的規範必須完善。

社會也不會太輕易接納 Level 5，會有很多不信任感。

1-

```python
import math
import random
import copy

class Task:
    def __init__(self, task_id, priority, transmission_time, period):
        self.task_id = task_id
        self.priority = priority
        self.transmission_time = transmission_time
        self.period = period
        self.worst_response_time = 0

def CAN_schedule(n, tau, tasks):
    sorted_tasks = sorted(tasks, key=lambda x: x.priority)

    for i in range(n):
        task = sorted_tasks[i]
        B = max((t.transmission_time for t in sorted_tasks[i: ]), default=0)
        Q = B
        RHS = 0
        while True:
            Q_plus_tau = Q + tau
            RHS = B
            for j in range(i):
                RHS += math.ceil(Q_plus_tau / sorted_tasks[j].period) * sorted_tasks[j].transmission_time

            if Q == RHS:
                break

            Q = RHS

        task.worst_response_time = round(task.transmission_time + Q, 5)

        if task.worst_response_time > task.period:
            return False

    tasks = sorted(sorted_tasks, key=lambda x: x.task_id)
    return True

def cost(n, tau, S): # sum Task.worst_response_time, will set worst_response_time
    if not CAN_schedule(n, tau, S):
        return 1e10
    else:
        return sum(task.worst_response_time for task in S)

def neighbor(S): # Randomly choose two tasks, and swap their priorities
    i, j = random.sample(range(n), 2)
    S[i].priority, S[j].priority = S[j].priority, S[i].priority

def Simulated_Annealing(n, tau, tasks):

    T = 10000
    T0 = 0.001
    r = 0.995

    S = copy.deepcopy(tasks)

    best_S = copy.deepcopy(S)
    best_total_worst_response_time = cost(n, tau, best_S)

    while T > T0:
        cost_S = cost(n, tau, S)
        S_prime = copy.deepcopy(S)
        neighbor(S_prime)
        cost_S_prime = cost(n, tau, S_prime)
        delta_C = cost_S_prime - cost_S
        P = min(1, math.exp(-1 * delta_C / T))
        S = S_prime if random.random() < P else S

        cost_S = cost(n, tau, S)
        if cost_S < best_total_worst_response_time:
            best_S, best_total_worst_response_time = copy.deepcopy(S), cost_S

        T *= r

    return best_S, best_total_worst_response_time

if __name__ == '__main__':
    with open('./input.dat', 'r') as file:
        lines = file.readlines()
        n = int(lines[0])
        tau = float(lines[1])
        tasks = [Task(i, int(P), float(C), int(T)) for i, (P, C, T) in enumerate(line.split() for line in lines[2:])]

    tasks, best_total_worst_response_time = Simulated_Annealing(n, tau, tasks)
    for task in tasks:
        print(task.priority)
    print(f"{best_total_worst_response_time:.5f}")
```

ans:

```
14
6
0
4
3
5
9
1
7
8
13
16
11
2
15
10
12
204.12000
```