Term Project Report

R10944015 陳法熏
B07902062 張彧瑋

January 4, 2022

1 Introduction

In the class, we are told that automation and connectivity of vehicles could improve safety and increase traffic flow. However, we still wonder how those algorithms works in real life. Curiosity urges us to have a look at the details of implementations and do some simulations to see concrete results. We chose the specific scenario, lane merging, to be our target to focus on, with the hope to get better understanding on possibility of intelligent vehicles. In the following section, we assume all vehicles are connective and autonomous and equipped with CACC (Cooperative Adaptive Cruise Control).

We first survey some related papers and focus on two of them. The first one [2] implementing a FCFS-like method (first-come-first-serve) with detail settings, provide us the rough concept about simulation. The second one [1] proposed problem formulation and a dynamic programming algorithm (DP) about lane merging, which is our main purpose to reproduce. We'll show the results and comparisons of these two approaches in section 2. In section 3, we propose a different approach to achieve lane merging in an easier way. Our contributions and discussions are shown in section 4. You can check section 5 for more information about our works.

2 Implementation

In this section, we tried to implement and reproduce the results of [1]. Since we use their formulation directly, we will briefly describe the formulation and method here.

Our works are implemented with `Python 3.7`, and we visualize our result with the help of the `vpython` package. All works are done under the environment of `jupyter notebook`, including our extended approach in section 3.

2.1 Formulation

Index	i/j	the index of vehicles on Lane A/B
Given (Input)	Δ_i/δ_j	the i-th/j-th vehicle on Lane A/B
	α/β	the number of vehicles on Lane A/B
	a_i/b_j	the earliest arrival time of Δ_i/δ_j
	$W^=/W^+$	the waiting time if two consecutive vehicles are from the same/ different incoming lane(s)
	V_{max}	the speed limit of the road
	A_{max}/A_{min}	the max acceleration/ deceleration speed of vehicles
Output	s_i/t_j	the scheduled entering time of Δ_i/δ_j

Table 1: Notations of DP algorithm for two-lane merging

The algorithm assumed that the vehicle dynamics is capable of following the scheduled entering time at the start of the intersection region (IR), which is the merging point. The notations used in the paper are denoted in Table 1. The constraints of the problem are defined as follows:

$$
\begin{aligned}
\textbf{minimize} \quad & \max_{i,j}\{s_i, t_j\} \\
\textbf{subject to} \quad & s_i \geq a_i, \\
& t_j \geq b_j, \\
& s_{i+1} - s_i \geq W^{=}, \\
& t_{j+1} - t_j \geq W^{=}, \\
& |s_i - t_j| \geq W^{+}, \\
& i = 1, 2, ..., \alpha,\ j = 1, 2, ..., \beta.
\end{aligned}
$$

There are few definitions and assumptions worth noticing in the original paper.

- **Earliest Arrival Time**: For calculation, the original paper supposed the earliest arrival time of a vehicle is the earliest possible time the vehicle can arrive at the merging point if there are no vehicles ahead. In our case, we denote the distance for vehicle Δ_i to reach V_{max} as $S(\Delta_i)$, the initial speed of Δ_i as V_{Δ_i}, and the distance between Δ_i and merging point as Dis_{Δ_i}. The calculation of the earliest arrival time is as follows:

$$
S(\Delta_i) = \frac{(V_{max} - V_{\Delta_i})}{A_{max}} \times \frac{(V_{max} + V_{\Delta_i})}{2} \tag{1}
$$

$$
a_i = \begin{cases} \dfrac{(V_{max} - V_{\Delta_i})}{A_{max}} + \dfrac{(Dis_{\Delta_i} - S(\Delta_i))}{V_{max}}, & \text{if } S(\Delta_i) \leq Dis_{\Delta_i}. \\ \dfrac{\sqrt{V_{\Delta_i}^2 + 2 \times A_{max} \times Dis_{\Delta_i}} - V_{\Delta_i}}{A_{max}}, & \text{otherwise.} \end{cases} \tag{2}
$$

  The same calculation stands for δ_j.

- **Waiting Time**: Waiting time denotes the minimum time needed between two consecutive vehicles to arrive at the merging point. The assumed waiting time is W^+ if two consecutive vehicles comes from different lanes; $W^=$ if they're from the same lane. Considering CACC, we supposed that $W^=$ is way smaller than W^+. In our case, W^+ is double the time of $W^=$, i.e. $W^+ = 2 \cdot W^=$

- **Scheduled Entering Time**: The algorithm assumed that the scheduled entering time s_i is calculated by the roadside merging manager with the velocity and position information provided by the vehicles, then be broadcast back to Δ_i. The algorithm also assumed that the vehicle dynamics is capable of following the scheduled entering time, which unlike other paper [2], did not consider the safety constraint before the merging point. This is likely, since with CACC, the vehicles can handle the safety constraint on their own.

- **Predefined Parameters of Our Work**: In our attempt, we assumed the following parameters:
  - $V_{max} = 27(m/s)$
  - $A_{max} = 4(m/s^2)/\ A_{min} = -4(m/s^2)$
  - $W^= = \frac{5}{V_{max}}(s)/\ W^+ = \frac{10}{V_{max}}(s)$

The algorithm is designed to minimize the latest time for all vehicles to pass the merging point. The paper defined *L(i, j, A)* to represent the minimum last scheduled entering time for the first i vehicles of *Lane A* and the first j vehicles of *Lane B* to past the merging point while the last vehicle is from *Lane A*. And define *L(i, j, B)* to represent the minimum last scheduled entering time when the last vehicle comes from *Lane B*.

Therefore, we can derive the DP algorithm as follows:

$$
L(i, j, A) = min\{max\{a_i, L(i-1, j, A) + W^=\}, max\{a_i, L(i-1, j, B) + W^+\}\} \tag{3}
$$

$$
L(i, j, B) = min\{max\{b_j, L(i, j-1, A) + W^+\}, max\{b_j, L(i, j-1, B) + W^=\}\} \tag{4}
$$

, where *L(0, 0, A)* $= 0$, *L(0, 0, B)* $= 0$, $L(0, j, A) = \infty\ \forall\ j \in [1, \beta]$, and $L(i, 0, B) = \infty\ \forall\ i \in [1, \alpha]$. Thus, we can get the minimum last scheduled entering time by comparing $L(\alpha, \beta, A)$ and $L(\alpha, \beta, B)$, then obtain the optimal sequence by backtracking. The algorithm is shown in algorithm 2.1.

Algorithm 1 Lane Merging Management with Dynamic programming

Input: $\{a_i\}, \{b_j\}, W^+, W^=$
Output: Optimal merging sequence

$L(0,0,A) = L(0,0,B) = 0$
$L(1,0,A) = a_1$
$L(0,1,B) = b_1$
for $i \leftarrow 2$ **to** α **do**
    $L(i,0,A) = max\{a_i, L(i-1,0,A) + W^=\}$;
end for
for $j \leftarrow 2$ **to** β **do**
    $L(0,j,B) = max\{b_j, L(0,j-1,B) + W^=\}$;
end for
for $i \leftarrow 1$ **to** α **do**
    $L(i,0,B) = \infty$;
end for
for $j \leftarrow 1$ **to** β **do**
    $L(0,j,A) = \infty$;
end for
for $i \leftarrow 1$ **to** α **do**
    for $j \leftarrow 1$ **to** β **do**
        $L(i,j,A) = min\{max\{a_i, L(i-1,j,A) + W^=\}, max\{a_i, L(i-1,j,B) + W^+\}\}$;
        $L(i,j,B) = min\{max\{b_j, L(i,j-1,A) + W^+\}, max\{b_j, L(i,j-1,B) + W^=\}\}$;
        Record the order for both cases;
    end for
end for
Backtrack the merging sequence;
Output the results;

Road 1			Road 2		
i	$V_i(m/s)$	$Dis_i(m)$	j	$V_j(m/s)$	$Dis_j(m)$
1	20	55	1	22	40
2	22	62	2	20	60
3	21	69	3	25	73
4	25	84	4	22	80
5	23	91	5	21	87

Table 2: Initial vehicle profile

Dynamic programming				First-come-first-serve		
Road Id	Veh Id	Time(s)		Road Id	Veh Id	Time(s)
2	1	1.60		2	1	1.60
1	1	2.27		1	1	2.27
1	2	2.46		1	2	2.46
2	2	2.81		2	2	2.81
2	3	3.00		2	3	3.00
2	4	3.20		1	3	3.37
2	5	3.46		2	4	3.74
1	3	3.76		1	4	4.11
1	4	3,94		2	5	4.48
1	5	4.12		1	5	4.86
	4.12		T_{last}		4.86	
	0.33		T_{delay}		0.55	

Table 3: Simulation results of DP and FCFS

2.2 Experimental Results

We compared the result of the DP algorithm with a FCFS algorithm similar to [2], which only considers the earliest arrival time of a vehicle when scheduling. We ran a simple case of $\alpha = 5$ and $\beta = 5$. The initial velocity and distance to IR are shown in Table 2, and the result of both algorithms are shown in table 3. You can also check the video comparison between DP and FCFS in section 5 for the animation.

2.3 Observations

From the result in Table 3, we can clearly see that the DP algorithm has a way **better performance** than the FCFS one, either in T_{last} (last scheduled entering time) or T_{delay} (the average delayed time of all vehicles). T_{delay} is calculated by s_i - a_i for all vehicles. We also observed that the optimal solution **tends to allow vehicles on the same lane to pass through at once**, which is reasonable, since it creates better utilization of the road.

However, the algorithm has a few potential concerns while applying it to real-life application.

- **Huge Overhead for Continuous Computing**: Due to the nature of DP, applying it on a continuous vehicle array may produce unneglectable overhead on calculation. Considering the computing power of roadside unit and the tolerance range of delay time for traffic application, even though the number of vehicles for computation might not be large, time complexity $O(k \cdot n^k)$ for every n vehicles on k lanes would be an issue to pay attantion to.
- **Possibly Frequent Change in Order**: The algorithm may cause frequent changes in the sequence of vehicles, which lead to inefficiency in the movement of the vehicles. Take the scenario in Table 4 as an example. By adding a single car into the queue, the scheduled order and entering time changed almost completely, result in continual changes of velocity of vehicles. This may cause discomfort to passengers, as well as do harm to the engine and the utilization of the road.
- **Starvation**: Consider the following scenario. There is always a new vehicle on one lane that arrives just before the scheduled time of the leader vehicles on the other lane. It is possible that vehicles in the opponent lane would never be able to enter the intersection, which causes unfairness in the management.

	Earliest Arrival Time	Scheduled Entering Time	Order
Δ_1	1	6	3
δ_1	2	2	1
δ_2	3	3	2
Δ_2	4	7	4
Δ_3	5	8	5

	Earliest Arrival Time	Scheduled Entering Time	Order
Δ_1	1	1	1
δ_1	2	4	2
δ_2	3	5	3
Δ_2	4	9	5
Δ_3	5	10	6
δ_3	6	6	4

	Earliest Arrival Time	Scheduled Entering Time	Order
Δ_1	1	1	1
δ_1	2	8	4
δ_2	3	9	5
Δ_2	4	4	2
Δ_3	5	5	3
δ_3	6	10	6
δ_4	8	11	7

Table 4: Scenario of frequently changing scheduled order($W^{=} = 1, W^{+} = 3$)

3 Extended Work

After getting the results from section 2, we start to think about how to apply the algorithm on continuous vehicle array while keep the simulation reasonable in real life. However, the approach did

not goes well since dynamic programming does not suit the environment of continuous vehicles due to the reasons we've been discussed in section 2.3. Thus, we try to develop another method to achieve lane merging for vehicles with CACC. In this section, we discard the assumption that vehicles from the same lane can enter the IR with half of the time that vehicles from different lanes take, i.e. $W^+ = W^=$

3.1 Proposed Approach

With the assumption that each vehicle is equipped with CACC, we decided to make good use of it. We assume that every vehicle can follow another vehicle (even in different lane) by CACC, which is reasonable since vehicles are connective under our assumption. And here is the algorithm we construct to achieve lane merging of continuous vehicles. The basic idea of the algorithm is to maintain a merge sequence beyond two vehicle arrays, and let vehicles follow their successor in the sequence, i.e. another vehicle with minimum higher priority.

Algorithm 2 Lane Merging Management with CACC

Require: list of vehicles: $lane_1$, $lane_2$; region: preparation_area

Ensure: Rearrange the CACC target of each vehicle such that all vehicles can enter IR by CACC without collision.

```
1: t(car) := estimated time to enter IR
2: car.prev := CACC target of car
3: for car ∈ lane_2 do
4:     if car ∈preparation_area and car has not been merged then
5:         for all (car'.prev, car')  do
6:             if t(car'.prev) ≤ t(car) ≤ t(car') then
7:                 car.prev ← car'.prev
8:                 car ← has been merged
9:                 car'.prev ← car
10:                break
11:            end if
12:        end for
13:    end if
14: end for
```

While implementing, we also add some heuristic methods to improve the decision of priority. With this algorithm and CACC on every vehicle, we can easily achieve smoothly lane merging with randomly generated vehicle sequence. Noted that CACC is needed here rather than ACC because connectivity is needed to follow the vehicle in another lane. The management can be distributed if the computation of function $t(\cdot)$ are fixed on every vehicle.

3.2 Experimental Results

The simulation is not easy to show in the paper. It is suggested to see the video lane merging simulation in section 5 for better experience. We screenshot three picture continuously in that video, and do negative film effect on them since the origin background is black. The result shows in figure 1. It's not hard to see that CACC makes two red vehicles slower for waiting two blue vehicles that algorithm assign higher priority. Noted that the merging process only required CACC, the algorithm is more like an agreement between vehicles.

3.3 Observations

There are two main issues found during the experiments. First, the accuracy of function $t(\cdot)$ play an important role for the merging results and it is actually depends on the range of preparation_area. We found out that when the length of preparation_area is too large, the merging process become worse. Which is unexpected to us, since we originally thought that larger preparation_area gives more time for CACC to adjust the position of each vehicle. We conjecture that excessive preparation_area cause lower accuracy of $t(\cdot)$.

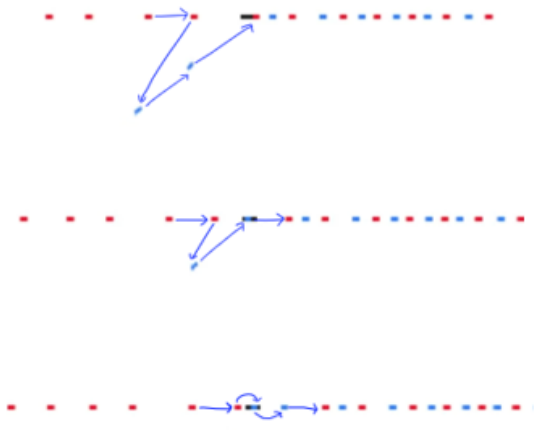

Figure 1: continuous screenshot in one simulation

Second, the ability of CACC matters a lot. We tried a few settings on how to implement CACC and most of them are not competent for smoothly lane merging. However, we believe this technique has been received high attention in real life, it should not be the problem for us to worry about.

4 Conclusions

We reproduced the algorithm proposed in [1], and showed that the dynamic programming approach is useful for finding the optimal strategy for a two-lane merging scenario. However, we also identified some hidden worries and difficulties implementing the algorithm in real-time application. To solve the problem, we thought that adding some human heuristic into the assumption is needed, rather than only considering to minimize the last scheduled entering time.

For applying to realistic scenario, we designed a new algorithm to insert vehicles in neighboring lanes with CACC only. Which can be easily implemented in real life, while heuristics can be added or modified in a short amount of time. We ran a few simulations to test the practicability of this algorithm and it seems promising most of the time. The functionality of this approach is actually similar to human behavior, i.e., human drivers seek for and adjust their speed to fit in an appropriate position. With the complete information provided by CACC, the measure can be done more efficiently and accurately.

We put the three main programs of the experiments on `github`. Check the github repo in section 5 and feel free to try it out.

5 Appendix

1. video: comparison between DP and FCFS
2. video: lane merging simulation
3. github repo

References

[1] Shang-Chien Lin, Hsiang Hsu, Yi-Ting Lin, Chung-Wei Lin, Iris Hui-Ru Jiang, and Changliu Liu. A dynamic programming approach to optimal lane merging of connected and autonomous vehicles. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 349–356. IEEE, 2020.

[2] Gurulingesh Raravi, Vipul Shingde, Krithi Ramamritham, and Jatin Bharadia. Merge algorithms for intelligent vehicles. 2007.