Consecutive Lane Merging with Dynamic Programming

b09902047 林品翰, b09902049 黃振承, b09902123 張偉彥

1 Introduction

The type of our project is a combination of "survey" and "implementation" and is mostly based on previous work [1]. Besides, we can promise that the report does not have any double assignment or any completed work before this semester.

With the development of V2V and V2X technology in recent years, vehicles can now communicate when running on the road. Moreover, this feature enables all vehicles to cooperate to increase road use efficiency and there have been several related topics about such issues. One of the popular topics is lane merging. If we can derive a scheduling strategy that optimizes the time used during lane merging, the resulting occurrence of traffic jams can definitely decrease.

To deal with the issue, a dynamic programming method comes in handy for calculating the time it costs. Based on the experiment results from [1], we can know that the efficiency can be obviously improved when applying such an approach to a two-lane merging scenario. However, the paper didn't conduct a complete experiment of consecutive lane merging. But we think it's still an important case because people may probably encounter such a situation when leaving highways which could lead to traffic jams during holidays. Therefore, we try to implement the three experiments regarding it and give an explanation to specify why this algorithm works and is optimized.

2 Formulation and Algorithm

2.1 Assumptions and Notation

The main issue of our project is to deal with the scenario of the consecutive lane merging like **figure 1**. Hence, we have to make several assumptions and notations first.

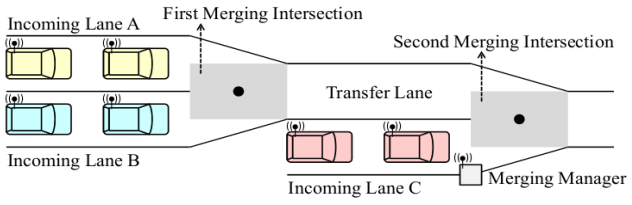

figure 1 (figure 4 from [1])

Assumptions:

- All vehicles are connected and autonomous.
- All vehicles are moving along the same direction.
- The earliest arrival time of each vehicle follows a Poisson distribution (λ : the frequency).
- Consecutive vehicles from the same lane can keep the vehicle-following gaps maintained by CACC right after merging, which is common for current vehicles.
- Generally, $W^{=}$ is greater than W^{+}.

index	$i/j/k$	the index of vehicles on Lane A/B/C
Given (input)	$\delta i/\Delta j/\zeta k$	the $i^{th}/j^{th}/k^{th}$ vehicle on Lane A/B/C
	$\alpha/\beta/\gamma$	the number of vehicles on Lane A/B/C
	$a_i/b_j/c_k$	the earliest arrival time of $\delta i/\Delta j/\zeta k$
	$W_1^{=}/W_1^{+}$	the same/different lane(s) waiting time for the first merging point
	$W_2^{=}/W_2^{+}$	the same/different lane(s) waiting time for the second merging point
	T_f	the transfer time between intersections
Output	$s_i^{'}/t_j^{'}$	the scheduled entering time of $\delta i/\Delta j$ to the first merging point
	$s_i/t_j/u_k$	the scheduled entering time of $\delta i/\Delta j/\zeta k$ to the second merging point

Table 1: Notation of the scenario (Table 2 and Table 3 from [1])

2.2 Algorithm

Now, we can start to consider the scenario. Our ultimate goal is to

$$\text{minimize} \max_{i,j,k} \{s_i, t_j, u_k\},$$

and we will show how to transform this formulation into a more detailed one. Different from a single-point lane merging, the consecutive lane merging has to know the previous passing order and the scheduled entering time of the previous vehicle at both merging points. To decompose this problem into small pieces, we can first consider the last vehicle to pass the second merging point. If the vehicle comes from *Lane A* or *Lane B*, the last one passing the first merging point must comes from the same lane, for the reason that all vehicles can't overtake each other on the transfer lane. Otherwise, we have to record the last vehicle passing the first merging point is from which lane.

With the two situations mentioned below, we can further break this problem into four subproblems:

(1) The last vehicle passing both merging points is from *Lane A*.

(2) The last vehicle passing both merging points is from *Lane B*.

(3) The last vehicle passing the second merging point is from *Lane C*, while the last vehicle passing the first merging point is from *Lane A*.

(4) The last vehicle passing the second merging point is from *Lane C*, while the last vehicle passing the first merging point is from *Lane B*.

For each situations, we can define $L_1(i, j, k, A/B/C_A/C_B)$ and $L_2(i, j, k, A/B/C_A/C_B)$ respectively, where L_h represents the last scheduled entering time at the h^{th} merging point for $h \in \{1, 2\}$. Furthermore, with the definitions above, we can simply conclude that

$$\text{minimize} \max_{i,j,k} \{s_i, t_j, u_k\},$$

is equal to

$$\begin{aligned} \text{minimize} \quad & \min \{L_2(\alpha, \beta, \gamma, A), L_2(\alpha, \beta, \gamma, B), \\ & L_2(\alpha, \beta, \gamma, C_A), L_2(\alpha, \beta, \gamma, C_B)\}. \end{aligned}$$

Therefore, the algorithm below is generated and we will implement it to do several experiments. (The mathematics proof is omitted.)

Algorithm Dynamic programming for consecutive lane merging (algorithm 2 from [1])

Input: $\{a_i\}, \{b_j\}, \{c_k\}, W_1^=, W_1^+, W_2^=, W_2^+, T_f$
Output: $\min\{L_2(\alpha, \beta, \gamma, A), L_2(\alpha, \beta, \gamma, B), L_2(\alpha, \beta, \gamma, C_A), L_2(\alpha, \beta, \gamma, C_B)\}$
Initialization;
Compute boundary conditions for $i = 0, j = 0$, or $k = 0$;
for $i \leftarrow 1$ **to** α **do**
  for $j \leftarrow 1$ **to** β **do**
    for $k \leftarrow 1$ **to** γ **do**
      Derive $L_2(i, j, k, A)$;
      Derive $L_2(i, j, k, B)$;
      Derive $L_2(i, j, k, C_A)$;
      Derive $L_2(i, j, k, C_B)$;
      Record the passing order and arrival times for these four situations;
    end for
  end for
end for
Output the results;

3 Experiment and Result

3.1 Method

We implemented the algorithm in Python programming language. In a real-world event, the earliest arrival time of each vehicle should be periodically calculated based on its current position, kinematics, and dynamics. However, since our project only focuses on solving the optimal passing order of vehicles, we assume that the earliest arrival time of each vehicle is fixed and follows a Poisson distribution. Moreover, we use two metrics that help us evaluate our scheduling results.

(1) The scheduled entering time of the last vehicle T_{last}.

T_{last} stands for the total time cost during the merging; for a consecutive lane merging scenario, it's namely the scheduling entering time at the second merging point. Besides, it also works as the objective function in the dynamic programming based algorithm.

(2) The average delay time of all vehicles T_{delay}.

T_{delay} implies the average difference between each vehicle's scheduled entering time and its possible earliest entering time. When calculating the possible earliest entering time of a vehicle, we only take the ones in front of it on the same incoming lane into account. Moreover, T_{delay} also represents the gap between a solution and a lower bound to this problem. Thus, it can reflect the delay time for all vehicles, not only for the last one.

3.2 Results

In our experiments, we compare the results of the dynamic programming based algorithm and *First-Arrive-First-Go greedy method* which simply chooses the vehicle with earliest arrival time at both merging points. There are totally three situations we have simulated: (1) different λ, (2) different numbers of vehicles / N, (3) different vehicle-following gaps / $W^=$.

3.2.1 Difference λ

This situation enables us to evaluate the influence of traffic intensity. We set the number of vehicles from both incoming lanes to 30, $W^= = 1s$, and $W^+ = 3s$. The results are shown in **Table 2**.

With lower λ, both approaches can reach the minimum T_{last}. However, the dynamic programming based algorithm got a higher T_{delay} when $\lambda = 0.1$. The probable reason is that under such light traffic intensity, there would be several optimal passing orders to choose from, since T_{delay} isn't our objective function, our algorithm won't consider its value. Hence, T_{delay} may be larger than the greedy method.

With the rising of λ, it is obvious that *First-Arrive-First-Go greedy method* can't guarantee the minimum T_{last} and the difference between T_{last} and T_{delay} of it and those of our approach would become more obvious.

λ	First-Arrive-First-Go		Dynamic Programming	
	T_{last}	T_{delay}	T_{last}	T_{delay}
0.1	379.078	1.059	397.078	18.989
0.2	189.125	12.058	189.125	3.351
0.3	156.116	25.938	124.692	5.07
0.4	163.683	37.147	105.573	8.44
0.5	153.488	54.614	101.439	19.5

Table 2: Result for different λ.

3.2.2 Difference numbers of vehicles

We use different numbers of vehicles to run the simulations with $\lambda = 0.5$, $W^{=} = 1s$, and $W^{+} = 3s$. The results are shown in **Table 3** which proves that our algorithm can provide a much more efficient passing order than the greedy method when N rises.

N	First-Arrive-First-Go		Dynamic Programming	
	T_{last}	T_{delay}	T_{last}	T_{delay}
20	107.056	35.148	67.716	11.432
40	208.293	77.951	129.199	25.99
60	310.241	115.491	189.563	36.881
80	438.582	174.399	250.545	51.029
100	564.143	211.793	310.755	51.379

Table 3: Result for different N.

3.2.3 Difference vehicle-following gaps

The simulation with different $W^{=}$ helps us analyze the advantages of our algorithm under the different vehicle-following gaps. In this instance, we set the number of vehicles from both incoming lanes to 30, $\lambda = 0.5$, and $W^{+} = 3s$. The results are shown in **Table 4**.

We can observe that if $W^{=} = W^{+}$, i.e., $W^{=} = 3.0$, both approaches can find the optimal solution for this problem. Nevertheless, as $W^{=}$ decreases, the performance of our algorithm is much greater than that of the greedy method. This means that our algorithm is more effective if the vehicle-following gaps maintained by CACC get improved.

$W^{=}$	First-Arrive-First-Go		Dynamic Programming	
	T_{last}	T_{delay}	T_{last}	T_{delay}
1	153.488	54.614	101.439	19.5
1.2	172.088	63.414	116.658	26.243
1.4	182.688	67.538	132.639	33.971
1.6	196.088	72.667	149.239	41.931
1.8	208.688	79.254	165.839	49.776
2.0	212.488	79.058	182.439	56.961
2.2	223.488	82.241	199.039	65.037
2.4	232.088	85.16	215.888	73.267
2.6	245.488	90.534	233.088	81.716
2.8	257.462	95.386	250.428	90.376
3.0	267.862	98.984	267.862	98.984

Table 4: Result for different $W^{=}$.

With the three experiments we have done, we can conclude that the dynamic programming based algorithm can truly improve efficiency compared to the greedy method. After we have done that, there could be a question coming to your mind, why not just apply dynamic programming two lanes merging twice? In fact, this question is simple. Since our ultimate goal is to minimize T_{last}, some vehicles may choose to sacrifice their earliest time to pass the merging point to reach the optimal solution. As a result, the outcome of the first merging point based on the algorithm 1 from [1] can't guarantee that it will be the optimal solution for the whole process. Instead, if we consider both two merging points and record all possible combinations, we can definitely achieve the optimal solution, and that is exactly what our algorithm does.

4 Conclusion

Although there are more complicated situations in the world, the 2-lane merging algorithm provides fundamental conflict resolution. Moreover, we experimented on a more perplexing situation, consecutive 2-lane merging, and show that the corresponding algorithm is significantly more powerful than the traditional greedy algorithm. Intuitively, the more complicated problems we solve, the more benefits we get.

Of course, there is still a long way to go in realizing that all vehicles are connected and autonomous. Furthermore, even if we can solve this problem, there could be other obstacles, such as security, safety, or cost, coming after another. However, with the rapid development of science and technology, we believe that the vision will come true, and there will be more innovative ideas to deal with lane merging issues. Therefore, people can save more time and energy, which will enhance social welfare, too.

5 Work Distribution

a. 林品翰：oral presentation and slide

b. 黃振承：pseudo code implementation and experiment

c. 張偉彥：experiments design and final report

The above only lists the main tasks of each member. Most of the work is done jointly by all members. Besides, this link contains the code we implement to do the experiment.

References

[1] Shang-Chien Lin et al. "A Dynamic Programming Approach to Optimal Lane Merging of Connected and Autonomous Vehicles". In: *2020 IEEE Intelligent Vehicles Symposium (IV)* (2020), pp. 349–356. DOI: https://ieeexplore.ieee.org/document/9304813.