Interchange Management

Kevin Kai-Chun Chang
Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan
b07901056@ntu.edu.tw

ABSTRACT

The areas near interchanges are often the most congested parts of a highway. In this work, we formulate the interchange management problem as an scheduling, decision-making, and optimization problem. We convert the problem onto the graph domain and adopt simulated annealing to optimize. Experimental result shows that our algorithm outperforms the first-come-first-serve strategy in terms of the total time required for all the vehicles passing the interchange area.

KEYWORDS

Connected and autonomous vehicle, constraint graph, simulated annealing, interchange management

1 INTRODUCTION

The areas near interchanges are often the most congested parts of a highway. If the trajectories and passing orders of vehicles are be planned appropriately, the traffic jam on the highway could be extremely improved. As an interchange is a complicated system, the interchange management problem combines multiple sub-problems such as lane changing, lane merging, and lane splitting, etc. Previous works mainly focus on one or some of the sub-problems, and several methods have been proposed to solve the sub-problems respectively, which may be adopted to solve the interchange management problem. However, the solution quality may degrade during the integration across the sub-problems and sub-solutions.

In this work, we model the interchange system and define the interchange management problem accordingly. We treat the problem as a scheduling, decision-making, and optimization problem. Given the information of vehicles and road, the trajectories of vehicles are then determined. We introduce "constraint graph" to represent the problem, and we can perform solution extraction, cost evaluation, and optimization on the graph structure. To optimize the solutions, we adopt a simulated-annealing-based approach.

The main contribution of this work is summarized as follows:

- We model the system for interchange management and formulate it as a scheduling, decision-making, and optimization problem.
- We convert the problem onto the graph domain, on which solution extraction, cost evaluation, and optimization could be performed. Moreover, we can benefit from the well-developed algorithms on graphs.
- We adopt simulated annealing to optimize the solutions which is suitable for the real situation. In the real world, the interchange manager receives the information from vehicles periodically. Through the simulated annealing, we may converge to a high-quality solution rapidly.

Experimental results shows that our algorithm outperforms the first-come-first-serve strategy in terms of the total time required for all the vehicles passing the interchange area. In addition, the runtime overhead of our algorithm is subtle.

The remaining of this paper is organized as follows. Section 2 models the system of interchange and formulate the interchange management problem. Section 3 details our proposed algorithm. Section 4 shows the experimental results, and Section 5 concludes this paper.

2 SYSTEM MODELING AND PROBLEM FORMULATION

2.1 System Modeling

To formulate and solve the problem of interchange management, we can model the system via the following definitions, which are mainly inspired by [2]:

- **Lanes and Lane Changing**: In the area near the interchange, there are two kinds of lanes: inner lanes and outer lanes (i.e., exit lanes). Vehicles that intend to stay on the highway should switch to or keep on inner lanes, while vehicles that intend to leave must change to or keep on exit lanes. As Figure 1 shows. Moreover, all the lane changing should be performed before the end of the road segment.

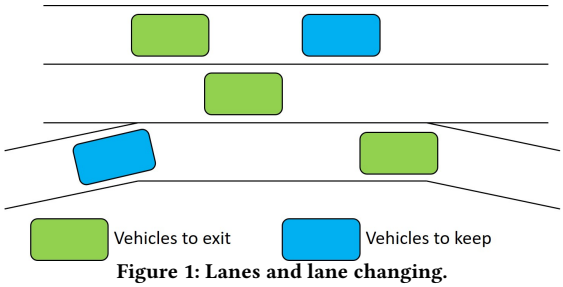

Figure 1: Lanes and lane changing.

- **Discrete Points and Trajectory**: To treat the interchange management problem as an optimization problem, some discrete points are added into each lane, as shown in Figure 2. The intervals between each pair of adjacent discrete points are the same, and the number of discrete points on each lane is adjustable. Vehicles could change lane at most once between each pair of adjacent points. That is, if vehicle Δ_i passes the k-th discrete point on the h-th lane, it may pass the $k+1$-th discrete point on the $h-1$-th lane, the h-th lane, or the $h+1$-th lane. As a result, the trajectory of a vehicle could be determined by (1) *the arrival time* and (2) *on which lane it locates* at each discrete point.

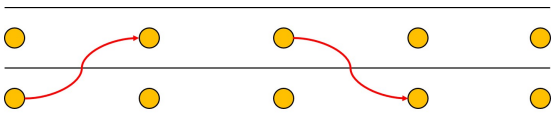

Figure 2: Discrete points. Vehicles could change lane at most once between adjacent discrete points, shown as the red arrows.

- **Interchange Manager**: We assume there is a centralized interchange manager at the roadside. It periodically receives information (e.g., speed and position) from each vehicle, and determines the vehicle trajectories. To simplify the problem, we assume the interchange manager could compute the *earliest arrival time* of each vehicle to the first discrete point before any vehicle enters the road segment, and the earliest arrival time of each vehicle remains fixed afterwards.
- **Same-Lane Traveling Time (T_{ST})**: To model the speed and the movement of vehicles, we define the same-lane traveling time, denoted as T_{ST}, to represent the minimum time required for a vehicle traveling from the k-th discrete point

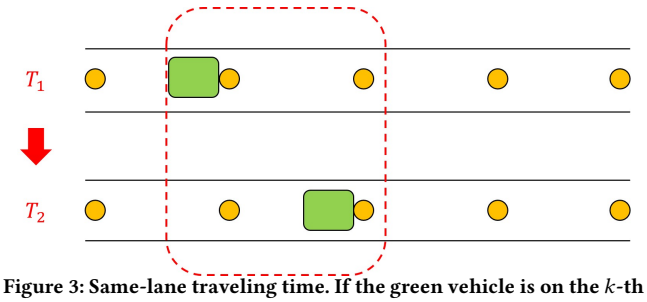

Figure 3: Same-lane traveling time. If the green vehicle is on the k-th discrete point at time T_1, and it arrives at the $k+1$-th discrete point on the same lane at time T_2, the time interval $T_2 - T_1$ must be larger than or equal to T_{ST}.

to the $k+1$-th discrete point on the same lane, as Figure 3 shows.

- **Cross-Lane Traveling Time (T_{CT})**: Similarly, we could define the cross-lane traveling time, denoted as T_{CT}, to represent the minimum time required for a vehicle traveling from the k-th discrete point on the h-th lane to the $k+1$-th discrete point on the $h-1$-th or the $h+1$-th lanes, as Figure 4 shows.

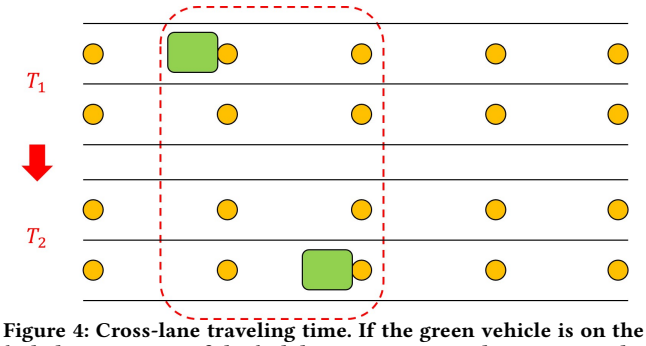

Figure 4: Cross-lane traveling time. If the green vehicle is on the k-th discrete point of the h-th lane at time T_1, and it arrives at the $k+1$-th discrete point of the $h+1$-th (or $h-1$-th) lane at time T_2, the time interval $T_2 - T_1$ must be larger than or equal to T_{CT}.

- **Same-Lane Separating Time (T_{SS})**: To model the safe following distance of two vehicles on the same lane, we define the same-lane separating time, denoted as T_{SS}, to represent the minimum time gap between the arrivals of two vehicles to the same discrete point, as shown in Figure 5.

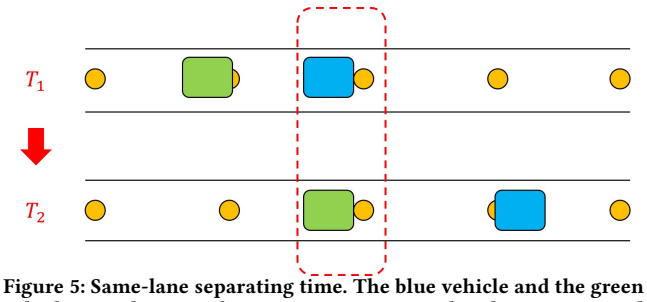

Figure 5: Same-lane separating time. The blue vehicle and the green vehicle pass the same discrete point successively. The time interval $T_2 - T_1$ must be larger than or equal to T_{SS}.

- **Cross-Lane Separating Time (T_{CS})**: We define the cross-lane separating time, denoted as T_{CS}, to model the situation that two vehicles on adjacent lanes perform lane-changing at the same segment. Figure 6 illustrates the definition.

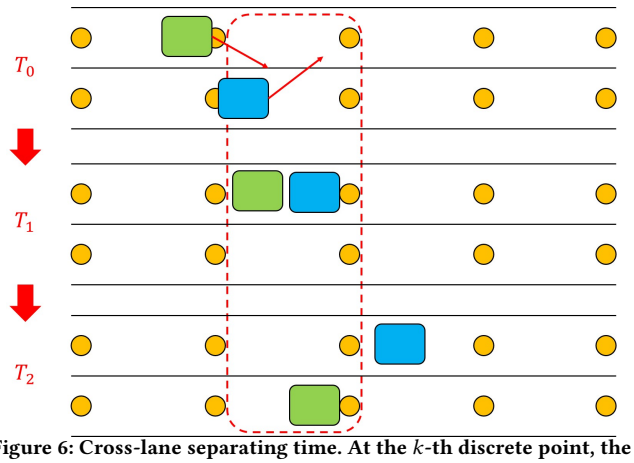

Figure 6: Cross-lane separating time. At the k-th discrete point, the blue vehicle is on the h-th lane and the green vehicle is on the $h+1$-th (or $h-1$-th) lane. Then, the blue vehicle moves to the $h+1$-th (or $h-1$-th) lane at the k-th discrete point, while the green vehicle moves to the h-th lane at the same time. The time interval $T_2 - T_1$ must be larger than or equal to T_{CS}.

2.2 Problem Definition

Based on the above system modeling, the interchange management problem could be defined as follows.

- **Input:**
  1. The number of vehicles N, the number of discrete points on each lane M, the number of inner lanes A, and the number of exit lanes B.
  2. The earliest arrival time, incoming lane, and whether to exit of each vehicle Δ_i.
  3. The traveling and separating timing constraints including T_{ST}, T_{CT}, T_{SS}, and T_{CS}.
- **Output:** The trajectory of each vehicle.
- **Objective:** Minimize the scheduled arrival time of the last car at the last discrete point, denoted as the *longest arrival time*.

3 PROPOSED ALGORITHM

Figure 7 show the algorithm flow of our work. Since the algorithms and manipulations on graphs are well-developed, and graph-based methods have been applied on several problems of connected vehicles such as the intersection management problem [3], we convert the input information into a graph, called as the *constraint graph*. Then, the initial solution could be extracted from the constraint graph and compute the corresponding cost (i.e., the longest arrival time). Eventually, simulated annealing is performed to find an optimal solution.

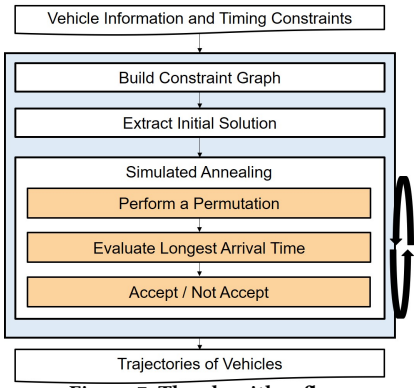

Figure 7: The algorithm flow.

3.1 Constraint Graph Construction

- **Nodes**: For each vehicle Δ_i, we enumerate all the discrete points that Δ_i might pass. That is, if Δ_i passes the k-th discrete point on the h-th lane within some of its trajectories, a node (i, h, k) is added to the constraint graph. Figure 8 shows an example. The first and the second lanes are inner lanes (i.e., keeping lanes), while the third and the fourth lanes are outer lanes (i.e., exit lanes). The vehicle Δ_i which intends to leave the highway comes from the first lane. Then, it could change at most one lane between each pair of discrete points, and leave the road segment via the third or the fourth lane.

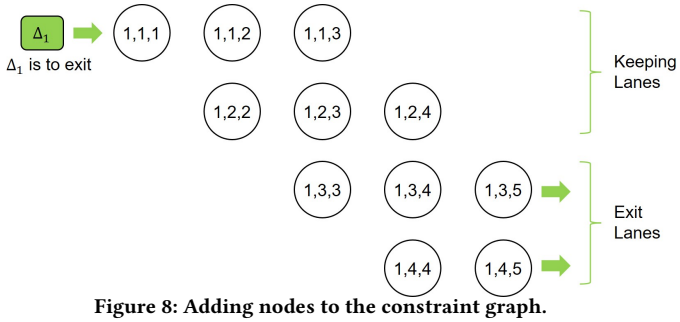

Figure 8: Adding nodes to the constraint graph.

- **Path Edges**: After adding all the nodes for vehicle Δ_i, we connect the nodes by path edges, which represent the possible paths for Δ_i. The edges are directed and weighted, with the direction begin the same as the possible direction of the vehicle, and the weight being the corresponding minimum traveling time across the path (i.e., T_{ST} or T_{CT}). Figure 9 shows the path edges and their weights.
- **Constraint Edges**: For the separating timing constraints, two kinds of constraint edges are added to the graph. For each pair of nodes (i, h, k) and (j, h, k), add an edge for the same-lane separating time constraint. The weight of the edge is the same-lane separating time T_{SS}, as shown in Figure 10. Similarly, for the cross-lane separating time constraint, if edge $(i, h, k) \rightarrow (i, h{+}1, k{+}1)$ and edge $(j, h{+}1, k) \rightarrow (j, h, k{+}1)$ both exist, add an edge connecting $(i, h+1, k+1)$ and $(j, h, k+1)$ with the weight being the cross-lane separating time T_{CS}, as illustrated in Figure 11. Note that in the most cases, the directions of the constraint edges is reversible. Thus, the directions can be viewed as bi-directed here and will be determined during each solution extraction.

3.2 Solution Extraction

As we enumerated the possible nodes and paths for each vehicle, the constraint graph collect all the solutions for the problem. To extract a specific solution from the constraint graph, there are two thing to determine. First, we need to determine the path of each vehicle. That is, select the nodes the vehicle is going to pass in the current

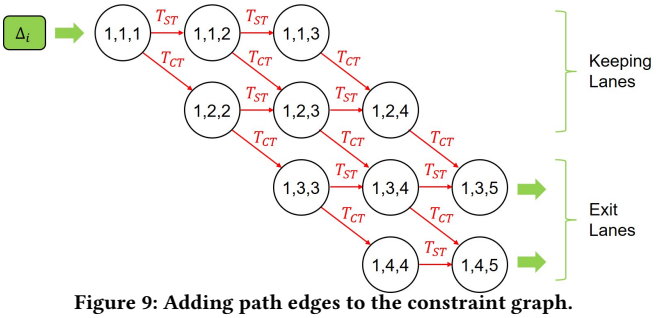

Figure 9: Adding path edges to the constraint graph.

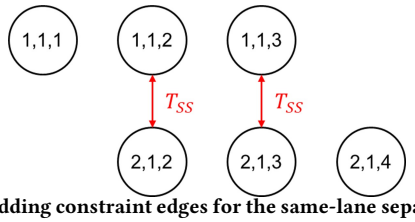

Figure 10: Adding constraint edges for the same-lane separating time to the constraint graph.

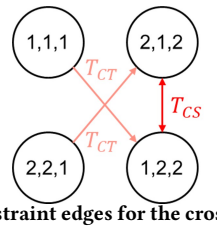

Figure 11: Adding constraint edges for the cross-lane separating time to the constraint graph.

solution. On the other words, it can viewed as determining the lane-changing points for the vehicles. Figure 12 shows an example. Second, we have to determine the direction for each constraint edge. As cycles in the constraint graph could incur deadlocks, we maintain a priority sequence for each lane. After we determine the priorities of each vehicle on each lane, the directions of constraint edges could be decided accordingly, and no cycle will appear. Figure 13 shows an example.

The solution extraction could be viewed as extracting a sub-graph from the original constraint graph. As shown in Figure 12, the green nodes and red edges are collected into the sub-graph, while the remaining nodes and edges could be neglected.

For the initial solution, the lane-changing points for each vehicle are determined randomly. The priority sequences of lanes are all initialized in the increasing order of the given earliest arrival time of the vehicles. As a consequence, there will be no cycles or deadlocks at the beginning.

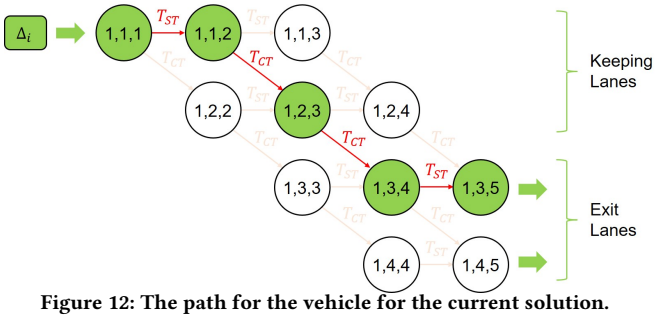

Figure 12: The path for the vehicle for the current solution.

3.3 Cost Evaluation

After extracting a solution (i.e. a sub-graph) from the constraint graph, we can compute its cost (i.e. the longest arrival time). First, we add a "psuedo-source" node s. For each vehicle Δ_i, we add an edge from s to the first node of Δ_i with weight being the earliest arrival time of Δ_i. As the traveling time, separating time, and the earliest arrival time of vehicles are all included in the extracted graph via directed edges, the longest path from s to each node is the minimum time required for the corresponding vehicle traveling to that node. Thus, among all the longest paths from s to the last nodes of vehicles, the length of the longest one is the longest arrival time.

Although finding the longest path on a general graph could be NP-complete, the cost evaluation procedure could be finished in

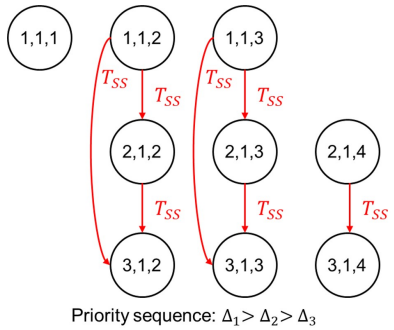

Figure 13: The priority order of the first lane and the direction of the corresponding same-lane separating edges.

Table 1: Experimental results.

N	Our Algorithm		First-Come-First-Serve	
	Cost	Runtime (s)	Cost	Runtime (s)
8	13.535	0.02	14.317	0.01
12	18.238	0.03	23.105	0.01
16	23.180	0.03	23.226	0.01
20	27.572	0.04	31.529	0.01

polynomial time as the extracted graph is a directed-acyclic-graph (DAG). In a DAG, we first perform topological sort for the nodes, and the longest path from a source node to each node could be computed one by one following the topological order. The complexity is of $O(MN^2)$, where M is the number of discrete points on each lane, and N is the number of vehicles.

3.4 Simulated Annealing

We apply simulated annealing to optimize the solution. For each permutation, we could perform one of the two following operations: (1) modifying lane-changing points for a vehicle and (2) swapping the priority of two vehicles in the priority sequence of a lane. For the first operation, we randomly select a vehicle and determine its lane-changing points. For the second operation, we randomly select a lane and an index i, then swap the priorities of the i-th and the $i+1$-th vehicles in the lane's priority sequence. Then, the directions of corresponding constraint edges may be reversed.

Note that the second operation may cause deadlock. Hence, we must assure that at least one of the two vehicles comes from a different lane. Otherwise, it's impossible for the rear vehicle to pass the front one. Besides, if we are going to change the priority for vehicle Δ_i and vehicle Δ_j, for all the node pairs (i, h, k) and (j, h, k) $(k = 1, 2, ..., M)$ on lane h, we need to check if there exists some other node (i', h', k) on other lanes connecting to both nodes through cross-lane separating constraint edges, as cycles may occurs in such situation.

4 EXPERIMENTAL RESULTS

The algorithm is implemented in the C++ programming language, and evaluated on a Linux environment. Following the settings in some previous works [1–3], we set $T_{ST} = 10/(M-1)$, $T_{CT} = 15/(M-1)$, $T_{SS} = 1$, and $T_{CS} = 2$. The intervals between earliest arrival time of vehicles follow an exponential distribution. The incoming lanes and whether to exit of vehicles are generated randomly and uniformly.

Table 1 shows our experimental result. We set $M = 6$, 3 inner lanes and 2 exit lanes in the experiment. Our algorithm outperforms the first-come-first-serve strategy in all the cases with different number of vehicles. The runtime overhead of our algorithm is subtle.

5 CONCLUSION

In this work, we formulate the interchange management problem and convert it onto the graph domain. Then, we propose a simulated-annealing-based approach to solve the problem. Experimental result shows that our method outperforms the first-come-first-serve strategy.

Future work includes more thorough experiments and analyses on our algorithm, the explorations and comparisons with previous works, and the integration of different optimization methods into our approach.

REFERENCES

[1] Shang-Chien Lin, Hsiang Hsu, Yi-Ting Lin, Chung-Wei Lin, Iris Hui-Ru Jiang, and Changliu Liu. 2020. A Dynamic Programming Approach to Optimal Lane Merging of Connected and Autonomous Vehicles. In *2020 IEEE Intelligent Vehicles Symposium (IV)*. 349–356.

[2] Shang-Chien Lin, Chia-Chu Kung, Lee Lin, Chung-Wei Lin, and Iris Hui-Ru Jiang. 2021. Efficient Mandatory Lane Changing of Connected and Autonomous Vehicles. In *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*. 1–7.

[3] Yi-Ting Lin, Hsiang Hsu, Shang-Chien Lin, Chung-Wei Lin, Iris Hui-Ru Jiang, and Changliu Liu. 2019. Graph-Based Modeling, Scheduling, and Verification for Intersection Management of Intelligent Vehicles. 18, 5s (2019).