Distributed Coordination and Centralized Scheduling for Automobiles at Intersections

Yi-Ting Lin, Chung-Wei Lin, Iris Hui-Ru Jiang, and Changliu Liu

1 Introduction

The fundamental goal of vehicles is to perform transportation tasks between sources and destinations safely and efficiently. The conflicts between vehicles occur when the corresponding vehicles intend to pass through a location at the same time, and intersections are one of the most common conflicting scenarios. Traditionally, traffic lights, stop signs, and priorities defined by traffic rules can be applied to resolve conflicts at intersections. As the technology advances, connected and autonomous vehicles (CAVs) provide a revolutionary solution at intersections, where:

- **Connectivity** provides sufficient information between vehicles and/or roadside units so that a safe and efficient passing order of vehicles can be decided.
- **Autonomy** provides precise control so that the decided passing order of vehicles can be performed.

Y.-T. Lin
Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan
e-mail: f07943102@ntu.edu.tw

C.-W. Lin (✉)
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan
e-mail: cwlin@csie.ntu.edu.tw

I. H.-R. Jiang
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
e-mail: huirujiang@ntu.edu.tw

C. Liu
Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA
e-mail: cliu6@andrew.cmu.edu

V. K. Kukkala, S. Pasricha (eds.), *Machine Learning and Optimization Techniques for Automotive Cyber-Physical Systems*, https://doi.org/10.1007/978-3-031-28016-0_3

In this chapter, we consider connected and autonomous vehicles at intersections and introduce approaches solving the problem of intersection management, also known as the problem of conflict resolution in a more general perspective. The approaches are categorized into two categories:

- **Distributed coordination**, where vehicles coordinate and then decide a passing order of vehicles separately.
- **Centralized scheduling**, where a centralized unit, called intersection manager, decides a passing order of vehicles and provides the instructions to vehicles.

No matter an approach is distributed or centralized, also no matter from the perspective of an individual vehicle or the overall transportation system, the approach should provide the following properties:

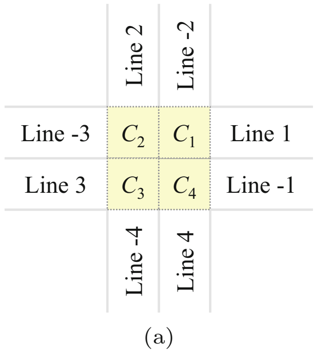
- **Feasibility**. The decided passing order of vehicles and the corresponding trajectories, including spatial and temporal constraints, much be physically achievable by the vehicles.
- **Safety** (collision-freeness). The deciding passing order must resolve the conflict for each pair of vehicles which intend to pass through a same location. Here, we define a conflict zone, and the safety requirement is that there is at most one vehicle occupying a conflict zone at the same time.
- **Liveness** (deadlock-freeness). The deciding passing order must not lead to a deadlock, i.e., an infinite waiting between multiple vehicles.
- **Stability**. The passing order must be stable along with the time line.
- **Efficiency**. The passing order should try to optimize the traffic efficiency or minimize delays of vehicles, i.e., allow vehicles to pass through intersections as soon as possible.
- **Real-Time Decision**. The passing order should be decided in real time without delaying vehicles due to waiting the decision or the corresponding instructions.

The chapter is organized as follows. Sections 2 and 3 present our distributed coordination and centralized scheduling approaches, respectively. Section 4 provides a summary.

2 Distributed Coordination

In this section, we present a distributed coordination approach for the problem of intersection management. The approach does not require a centralized intersection manager. There are three steps to implement the approach:

1. Each vehicle broadcasts its estimated time intervals to occupy the corresponding conflict zones.
2. Given the broadcast information, all vehicles reach a consensus of the passing order by solving a conflict graph locally.



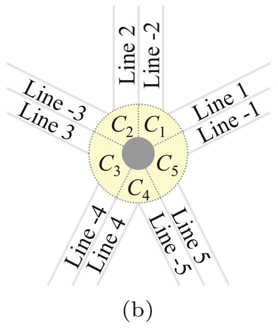


Fig. 1 Example environments: (**a**) intersection and (**b**) roundabout

3. Each vehicle adjusts its speed profile according to the passing order and updates its estimated time intervals to occupy the corresponding conflict zones.

We assume that the communication has no delay or packet loss. Figure 1 illustrates two example environments: a (real) intersection and a roundabout. A conflict zone is formulated when the extensions of two incoming lanes intersect with each other.

The rest of the discussion is organized as follows: Sect. 2.1 formulates the problem. Section 2.2 introduces the distributed approach and its theoretical guarantees. Section 2.3 provides simulation results, and Sect. 2.4 concludes the discussion.

2.1 *Problem Formulation*

A conflict zone is formulated when the extensions of two incoming lanes intersect with each other. The conflict zones are denoted by $C_1, C_2, \ldots, C_L$, where L is the total number of conflict zones. There are N vehicles, indexed from 1 to N, intend to pass through an intersection. The intention (the target lane after passing through the intersection) of vehicle i is G_i. The state of vehicle i at time t is denoted as $x_i(t)$. The system state at time step t is denoted as $x(t) := [x_1(t); x_2(t); \ldots; x_N(t)]$.

The system objective is to ensure that the intentions of the vehicles are satisfied efficiently and maintain the system safety. The safety constraint requires that the minimum distance between any two vehicles is larger than or equal to a threshold $d_{\min}$, e.g.,

$$\mathcal{X} := \{x \mid d(x_i, x_j) \geq d_{\min},\ \forall i, j, i \neq j\}, \tag{1}$$

where the function d measures the minimum distance between two vehicles i and j.

In a distributed setting, each individual vehicle only has a local view and local information, i.e., vehicle i only considers the vehicles in its neighborhood $\mathcal{N}_i$. Moreover, the other vehicles' states in the safety constraint are not directly accessible, so they need to be estimated. The navigation problem for vehicle i can be formulated as the following optimization problem:

$$\min_{x_i} \; J(x_i, G_i), \tag{2a}$$

$$\text{s.t. } \dot{x}_i(t) \in \Gamma(x_i(t)), \tag{2b}$$

$$d(x_i(t), \hat{x}^i_j(t)) \geq d_{\min}, \; \forall j \in \mathcal{N}_i, \tag{2c}$$

where J is the objective, and $\hat{x}^i_j(t)$ is the estimation of $x_j(t)$ made by vehicle i. Equation (2b) is the feasibility constraint to ensure that there is a low level controller to track the trajectory, e.g.,

$$\Gamma(x_i) := \{\dot{x}_i \mid \exists u_i, \; \dot{x}_i = f(x_i, u_i)\}, \tag{3}$$

where $\dot{x}_i = dx_i/dt$, u_i is the vehicle control input (wheel angle and throttle torque), and f describes the vehicle dynamics. It is assumed that all vehicles are equipped with perfect controllers that can execute the planned trajectories without any error if the planned trajectory is feasible.

In current design of autonomous vehicles, $\hat{x}^i_j$ is estimated based on local sensors [1, 2]. In order to account for uncertainties in the estimation, the behaviors of autonomous vehicles tend to be conservative. As a result, all vehicles may decide to slow down to yield, which is very inefficient. The behaviors of "connected" and autonomous vehicles can be less conservative due to more information and more accurate estimation. From the system level, less conservative behaviors imply smaller delay and larger throughput. Before we dive deep into the distributed coordination solution, we first introduce several assumptions and notation.

2.1.1 Assumption on Fixed Paths

We assume that each vehicle follows a fixed path, and Eq. (2) only optimizes for the speed profile along the path. Let x_i^* be the optimal trajectory of vehicle i that does not consider the collision avoidance constraint, e.g.,

$$x_i^* = \arg \min_{\dot{x}_i(t) \in \Gamma(x_i(t))} J(x_i, G_i). \tag{4}$$

Hence, the path of vehicle i is fixed along x_i^*, and the vehicle only adjusts its speed profile to meet the collision avoidance constraint. This assumption is reasonable since vehicles are usually not allowed to change lanes at intersections. In the following discussion, let $x_i^*(s)$ be the distance s parameterized path for vehicle i. The speed profile for vehicle i is denoted as $s_i(t)$ which is a mapping from time to the distance along the path. Then, $x_i^*(s_i(t))$ is the trajectory.

We say that vehicle i passes through the conflict zone C_l if there exists $s \in \mathbb{R}^+$ such that $\mathcal{B}_i(x_i^*(s)) \cap C_l \neq \emptyset$, where $\mathcal{B}_i$ denotes the area occupied by vehicle i at state $x_i^*(s)$. Define the segment on path x_i^* that intersects with the conflict zone C_l as $\mathcal{L}_{i,l} := \{s \mid \mathcal{B}_i(x_i^*(s)) \cap C_l \neq \emptyset\}$. Hence, $\mathcal{L}_{i,l} = \emptyset$ if and only if vehicle i does not pass through the conflict zone C_l. Denote the set of indices of conflict zones that vehicle i passes through as $\mathcal{A}_i := \{l \mid \mathcal{L}_{i,l} \neq \emptyset\}$. Then, two vehicles i and j pass through a same conflict zone if and only if $\mathcal{A}_i \cap \mathcal{A}_j \neq \emptyset$.

2.1.2 Notations of Discrete States

In addition to the continuous vehicle state x_i, to better describe the vehicle behaviors at intersections, we define a discrete state $\mathcal{S}_i$ for vehicle i, where

- $\mathcal{S}_i = IL$ if vehicle i is on an incoming lane, and it is not the first vehicle on the lane.
- $\mathcal{S}_i = FIL$ if vehicle i is on an incoming lane, and it is the first vehicle on the lane.
- $\mathcal{S}_i = I$ if vehicle i is at the intersection.
- $\mathcal{S}_i = OL$ if vehicle i is on an outgoing lane.

Vehicle i may enter the control area with $\mathcal{S}_i = IL$ or FIL. $\mathcal{S}_i$ can transit from IL to FIL, from FIL to I, and from I to OL, i.e., becoming the first vehicle on an incoming lane, entering the intersection, and leaving the intersection, respectively. It can leave the control area when $\mathcal{S}_i = OL$. For any vehicle i such that $\mathcal{S}_i = IL$ or OL, its front vehicle is denoted $\mathcal{F}_i$.

2.2 Distributed Coordination Approach

The key insight here is that communication can help the ego vehicle to better determine the constraint in Eq. (2c). Indeed, instead of estimating others' trajectories $\hat{x}_j^i$, what really matters to the ego vehicle is the time that other vehicles occupy the conflict zones. We design the communication protocol to be that each vehicle should broadcast the following two types of information:

- The estimated times to occupy the conflict zones once the vehicle enters a control area of the intersection, e.g., the shaded area in Fig. 1.
- The basic information such as the vehicle ID, the current state (position, heading, speed, and $\mathcal{S}_i$), and the time stayed in the control area.

Based on the broadcast information, the vehicles will seek a consensus on the passing order and compute desired time slots to pass through the conflict zones, which are then taken as temporal constraints on the vehicles' trajectories. This naturally breaks the problem into two parts as shown in Fig. 2a:

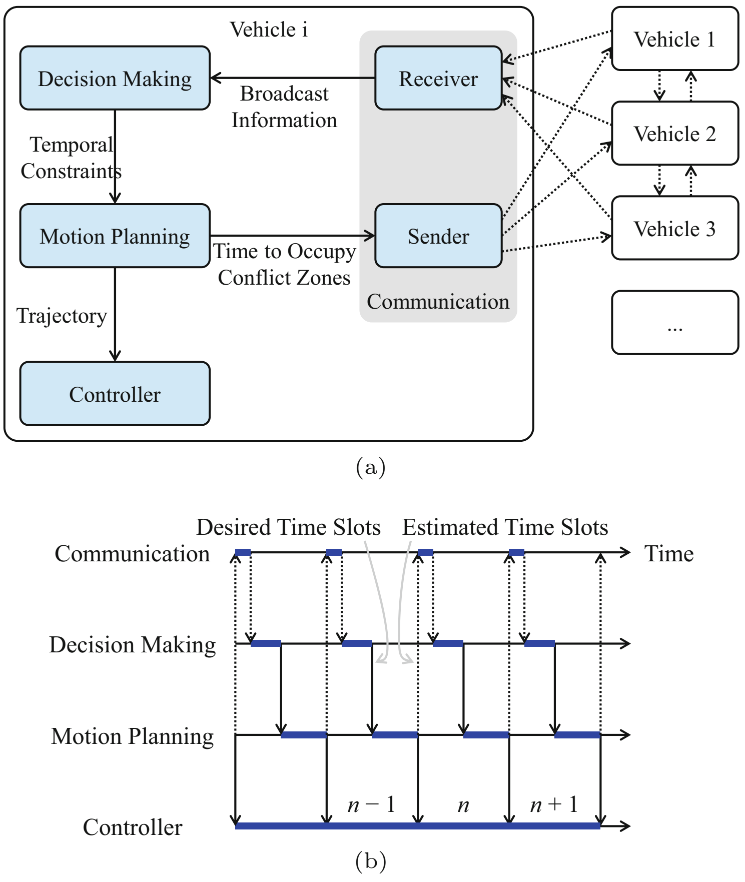


Fig. 2 Architecture of the conflict resolution mechanism. (**a**) System block diagram. (**b**) Time flow of execution

1. Decision making: determination of passing order and hence temporal constraints.
2. Motion planning: computation of trajectory.

The time flow and the coordination among different modules are shown in Fig. 2b. It is assumed that all vehicles are synchronized. At time step $n-1$, the estimated time interval $[\mathbb{T}_{j,l}^{in,n-1}, \mathbb{T}_{j,l}^{out,n-1}]$ for vehicle j to occupy C_l is broadcast for all j and l. At time step n, vehicle i evaluates all information received from other vehicles and computes the desired time slots to pass through the conflict zones in the decision maker, i.e., $[T_{i,l}^{in,n}, T_{i,l}^{out,n}]$ for all l, which are then sent to the motion planner as temporal constraints. After motion planning, the planned trajectory is sent to the controller for execution and the estimated time slots to occupy the conflict zones given the new trajectory, i.e., $[\mathbb{T}_{i,l}^{in,n}, \mathbb{T}_{i,l}^{out,n}]$ for all l, are broadcast to other vehicles.

The decomposition of decision making and motion planning can also be adopted in centralized intersection management, where the manager takes the responsibility

Algorithm 1 The decision making algorithm for vehicle i for computing the temporal constraints $[T_{i,l}^{in,n}, T_{i,l}^{out,n}], \forall l$ at time step n given information $[\mathbb{T}_{j,l}^{in,n-1}, \mathbb{T}_{j,l}^{out,n-1}], \forall j, l$

```
Initialize, $n = 0$
while $\mathcal{S}_i \in \{IL, FIL, I\}$ do
    Receive other's information $\mathbb{T}_{j,l}^{in,n-1}, \mathbb{T}_{j,l}^{out,n-1}$
    Initialize $\mathcal{Y}_i = \emptyset$, $T_{i,l}^{in,n} = -\infty$, $T_{i,l}^{out,n} = \infty$
    if $\mathcal{S}_i = IL$ then
        $i$ yields its front vehicle ($\mathcal{Y}_i = \{\mathcal{F}_i\}$)
    end if
    for $j$ that has spatial conflicts with $i$ ($j \in \mathcal{U}_i$) do
        if $j$ has a temporal advantage over $i$ ($j \in \mathcal{V}_i$) then
            if $\nexists$Tie($i, j$) or $j$ has priority over $i$ then
                $i$ yields $j$ ($\mathcal{Y}_i = \mathcal{Y}_i \cup \{j\}$)
            end if
        end if
        if $i$ has a temporal advantage over $j$ ($i \in \mathcal{V}_j$) then
            if $\exists$Tie($j, i$) and $j$ has priority over $i$ then
                $i$ yields $j$ ($\mathcal{Y}_i = \mathcal{Y}_i \cup \{j\}$)
            end if
        end if
    end for
    for $j$ that $i$ yields ($j \in \mathcal{Y}_i$) do
        for $C_l$ that both $i$ and $j$ traverse ($l \in \mathcal{A}_i \cap \mathcal{A}_j$) do
            $T_{i,l}^{in,n} = \max\{T_{i,l}^{in,n}, \mathbb{T}_{j,l}^{out,n-1} + \Delta_{\mathcal{S}_i}\}$
        end for
    end for
    $n = n + 1$
end while
```

of decision making, and the vehicles takes the responsibility of motion planning [3]. We discuss the decision making in Sect. 2.2.1 and the motion planning in Sect. 2.2.2.

2.2.1 Decision Making

At time step n, vehicle i needs to compute the desired time interval $[T_{i,l}^{in,n}, T_{i,l}^{out,n}]$ to pass through the conflict zones given the broadcast information $[\mathbb{T}_{j,l}^{in,n-1}, \mathbb{T}_{j,l}^{out,n-1}]$ for all j and l. The basic strategy is that whoever arrives first in a conflict zone goes first.[1] However, this strategy may create deadlocks when one vehicle arrives earlier in one conflict zone, while the other vehicle arrives earlier in another conflict zone. As a result, a tie breaking mechanism is needed. Here, we first discuss a

[1] Note that this is different from the strategies discussed in [4], which only considers the arrival time at the intersection.

general methodology to deal with distributed coordination with multiple conflict zones, which is summarized in Algorithm 1.

If $\mathcal{S}_i = IL$, it is physically "constrained" by its front vehicle and should yield all vehicles that its front vehicle yields. The decisions when $\mathcal{S}_i = FIL$ or I are the most important as conflicts usually come among vehicles in these two states. When $\mathcal{S}_i = OL$, the vehicle no longer needs to compute the desired time interval. However, its information should be broadcast in order for the proceeding vehicles to follow the lane safely. In the following discussion, we focus on vehicle i with $\mathcal{S}_i = FIL$ or I.

2.2.1.1 Spatial Conflict

We say that there is a spacial conflict between vehicles i and j if and only if their paths pass through a same conflict zone. Consider the scenario shown in Fig. 3a, where nine vehicles locate in a six-way intersection. The shaded area denotes the six conflict zones. By adding edges between any pair of vehicles that have spatial conflicts, we formulate an undirected graph as shown in Fig. 3b, where every vertex represents one vehicle. Whenever there is an edge between two vehicles, we need to decide which vehicle goes first. In other words, the undirected graph needs to be transformed into a directed graph as shown in Fig. 3d such that the passing order is decided by the topological order. Denote the set of vehicles that have spacial conflicts with vehicle i as

$$\mathcal{U}_i := \{j \mid \mathcal{S}_j = FIL \text{ or } I, \mathcal{A}_i \cap \mathcal{A}_j \neq \emptyset\}. \tag{5}$$

Recall that $\mathcal{A}_i$ denotes the set of indices of conflict zones that vehicle i passes through. Hence, $\mathcal{A}_i \cap \mathcal{A}_j \neq \emptyset$ means that vehicle j passes through one or more conflict zones that vehicle i also needs to pass through. The graph in Fig. 3b is denoted as $\mathcal{U} := \cup_i \cup_{j \in \mathcal{U}_i} (i, j)$, where (i, j) represents an edge between i and j. There is an undirected edge between any i and j such that $j \in \mathcal{U}_i$. In literature, this graph is identified as a conflict graph [5]. Finding the optimal passing order regarding the conflict graph is NP-hard. The approach presented here is a heuristic approach which finds one feasible passing order in linear time.

2.2.1.2 Temporal Advantage

At time step n, we say that vehicle $i \in \mathcal{U}_j$ has a temporal advantage over vehicle j, if one of the following conditions holds:

- $\mathcal{S}_i = I, \mathcal{S}_j = FIL$ and vehicle j leaves some conflict zones later than vehicle i enters, i.e.,

$$\exists l \in \mathcal{A}_i \cap \mathcal{A}_j, \mathbb{T}_{j,l}^{out,n-1} > \mathbb{T}_{i,l}^{in,n-1}. \tag{6}$$

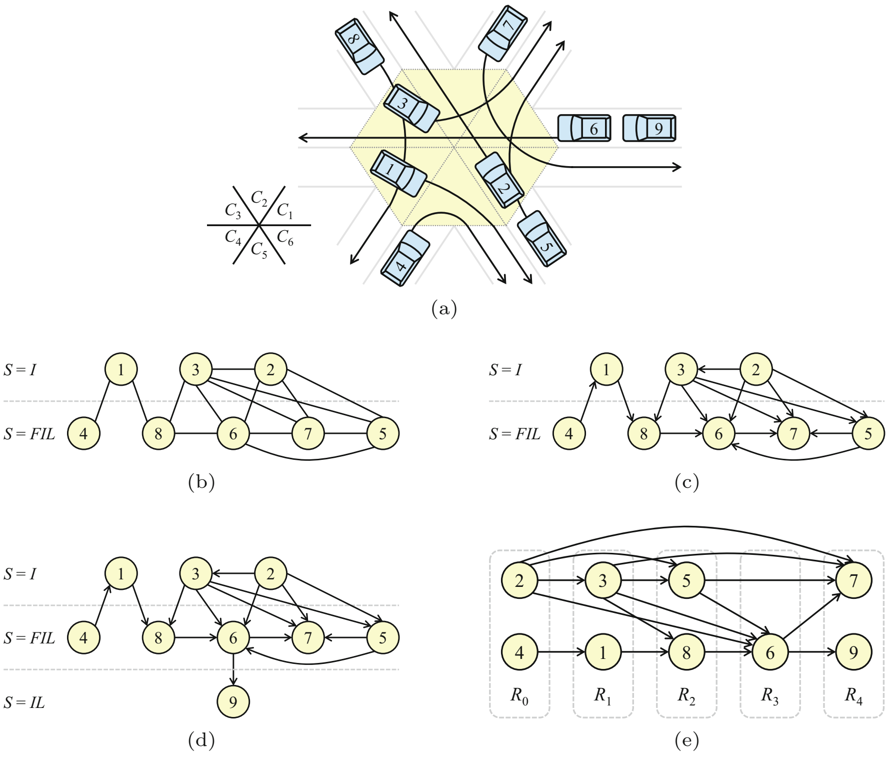


Fig. 3 The conflict graphs. (**a**) The scenario: nine vehicles in a six-way intersection. (**b**) Graph of spacial conflicts $\mathcal{U}$ at one time step. Edge $(i, j) \in \mathcal{U}$ implies that vehicle j has spacial conflicts with vehicle i at some conflict zone. (**c**) Graph of temporal advantages $\mathcal{V}$ at one time step. Edge $(i, j) \in \mathcal{V}$ implies that vehicle i has temporal advantages over vehicle j. (**d**) Graph of passing order $\mathcal{Y}$ at one time step. Edge $(i, j) \in \mathcal{Y}$ implies that vehicle j yields vehicle i. (**e**) Convergence of the graph $\mathcal{Y}$ through time. The passing order converges at time 0 for leaf vertices in $\mathcal{R}_0$, then for vertices with depth 1 in $\mathcal{R}_1$ at time 1, and so on

- $\mathcal{S}_i = FIL, \mathcal{S}_j = I$ and vehicle i leaves all conflict zones earlier than vehicle j enters, i.e.,

$$\forall l \in \mathcal{A}_i \cap \mathcal{A}_j, \mathbb{T}_{i,l}^{out,n-1} \leq \mathbb{T}_{j,l}^{in,n-1}. \tag{7}$$

- $\mathcal{S}_i = \mathcal{S}_j = FIL$ or I and vehicle i enters some conflict zones earlier than vehicle j, i.e.,

$$\exists l \in \mathcal{A}_i \cap \mathcal{A}_j, \mathbb{T}_{i,l}^{in,n-1} \leq \mathbb{T}_{j,l}^{in,n-1}. \tag{8}$$

According to the above definitions, for vehicles i and j with different discrete states, either i or j should have a temporal advantage over the other. If vehicles

i and j have the same discrete state, it is possible that both i and j have temporal advantages over the other. Denote the set of vehicles that have temporal advantages over vehicle j at time step n as $\mathcal{V}_j^n$. The superscript n in the following discussion is ignored for simplicity. It is obvious that $\mathcal{V}_j \subset \mathcal{U}_j$; and $\mathcal{V} := \cup_j \cup_{i \in \mathcal{V}_j} (i, j)$ is a directed graph as shown in Fig. 3c, where there is a directed edge from any $i \in \mathcal{V}_j$ to any j. However, there are cycles among vertices with the same discrete state, e.g., between vertices 6 and 7, as well as among vertices with different discrete states, e.g., among vertices 6, 7, and 3. If vehicles yield each other according to the graph, there are deadlocks. We will introduce a tie breaking mechanism to avoid these deadlocks.

2.2.1.3 Tie Breaking

For any vehicle i and vehicle $j \in \mathcal{V}_i$, it is called a *tie* if:

- $\mathcal{S}_i = \mathcal{S}_j$ and there exists a sequence of vehicles $\{q_m\}_1^M$ with $q_1 = i$, $q_M = j$, $M \geq 2$ and $\mathcal{S}_{q_m} = \mathcal{S}_i$ for all m such that $q_m \in \mathcal{V}_{q_{m+1}}$ for $m = 1, 2, \ldots, M - 1$.
- $\mathcal{S}_i = I$, $\mathcal{S}_j = FIL$ and there exists a sequence of vehicles $\{q_m\}_1^M$ with $q_1 = i$, $q_M = j$ and $M \geq 2$ such that $q_m \in \mathcal{V}_{q_{m+1}}$ for $m = 1, 2, \ldots, M - 1$.

Let $\text{Tie}(i, j)$ denote all these sequences. The relationship in a tie is neither symmetric nor exclusive, i.e., $\exists\text{Tie}(i, j)$ neither implies $\exists\text{Tie}(j, i)$ nor $\nexists\text{Tie}(j, i)$. For example, in Fig. 3c, there is a tie from vertex 5 to vertex 6 via the sequence $\{5, 7, 6\}$, but there is not a tie from vertex 6 to vertex 5 since $5 \notin \mathcal{V}_6$. There is a tie from vertex 2 to vertex 3 via the sequence $\{2, 3\}$ and a tie from vertex 3 to vertex 2 via the sequence $\{3, 2\}$.

We assume that each vehicle has a unique priority score P. For example, the priority score of a fire truck is higher than that of a passenger vehicle. We say that vehicle i has *priority* over vehicle j if there exists a sequence in $\text{Tie}(i, j)$ such that $P(i) > P(k)$ for all $k \neq i$ in the sequence. The basic principles are: (1) vehicles already in the intersection should always have priority over vehicles on the incoming lanes; (2) for vehicles in the same discrete state, the order implied by the priority score should not change over time. If vehicle i has priority over vehicle $j \in \mathcal{V}_i$, instead of i yielding j, vehicle j should yield vehicle i, although vehicle j has a temporal advantage. For example, in Fig. 3, we identify the score P with the vehicle index. Since vertex 5 has priority in the sequence $\{5, 7, 6\}$, the edge from 5 to 6 is reversed in Fig. 3d. Since there is a tie between vertex 2 and vertex 6, the edge from 2 to 6 is also reversed in Fig. 3d.

2.2.1.4 Passing Sequence

After tie breaking, all those remaining edges for vehicle j represent the set of vehicles that vehicle j decides to yield at time step n, which is denoted by $\mathcal{Y}_j^n$. The superscript n in the following discussion is ignored for simplicity. Indeed,

$\mathcal{Y} := \cup_j \cup_{i \in \mathcal{Y}_j} (i, j)$ is a directed graph as shown in Fig. 3d, which encodes the order for the vehicles to pass through the intersection. Note that it is not necessary for vehicle i to construct the whole graphs $\mathcal{U}$ and $\mathcal{V}$ to determine $\mathcal{Y}_i$. For example, vehicle 4 in Fig. 3 only needs to compute $\mathcal{U}_4$ and $\mathcal{V}_4$ locally to determine that $\mathcal{Y}_4 = \emptyset$. Those local decisions form the passing sequence globally. In the extreme case, the passing order follows the order specified by the priority scores. If all vehicles agree on the above tie breaking mechanism, they can solve the conflicts even if the vehicles plan and control their motions differently.

According to Algorithm 1, if $\mathcal{S}_j = IL$, the vehicle j yields its front vehicle, i.e., $\mathcal{Y}_j = \{\mathcal{F}_j\}$, as shown by vehicle 9 in Fig. 3d. If vehicle j decides to yield vehicle i, then for all $l \in \mathcal{A}_i \cap \mathcal{A}_j$, we set

$$T_{j,l}^{in,n} \geq \mathbb{T}_{i,l}^{out,n-1} + \Delta_{\mathcal{S}_j}, \tag{9}$$

where $\Delta_{\mathcal{S}_j}$ is a margin to increase the robustness of the algorithm, which is chosen such that $\Delta_{IL} > \Delta_{FIL} > \Delta_I$. Δ_{IL} is chosen to be larger than Δ_{FIL} to ensure the leading vehicles have temporal advantages over vehicles on the middle of other lanes. For example, vehicle 7 has a temporal advantage over vehicle 9 in Fig. 3d. Similarly, Δ_{FIL} is chosen to be larger than Δ_I.

2.2.2 Motion Planning under Temporal Constraints

At time step n, given the temporal constraint $[T_{i,l}^{in,n}, T_{i,l}^{out,n}]$ specified by the decision maker, the problem in Eq. (2) for vehicle i can be rewritten as:

$$\min_{s_i} \quad J(x_i^*(s_i), G_i) \tag{10a}$$

$$\text{s.t.} \quad \frac{\partial x_i^*(s_i)}{\partial s_i}\dot{s}_i \in \Gamma(x_i^*(s_i)), \tag{10b}$$

$$s_i(t) \notin \mathcal{L}_{i,l}, \ \forall t \notin [T_{i,l}^{in,n}, T_{i,l}^{out,n}], \ \forall l, \tag{10c}$$

where $s_i(t)$ is the speed profile that needs to be optimized. Equation (10c) specifies that the vehicle should only enter the conflict zone C_l in the time interval $[T_{i,l}^{in,n}, T_{i,l}^{out,n}]$. For simplicity, the constraint for vehicle following is omitted in presentation (but included in problem solution).

A method to efficiently solve the problem in Eq. (10) via temporal optimization is discussed in [6]. Here, we assume that vehicles can take unbounded deceleration, which is reasonable when vehicle speeds are low. Considering $T_{i,l}^{out,n} = \infty$ by Algorithm 1, there is always a solution of problem in Eq. (10). In the worst case, vehicle i just stops immediately. In practice, the vehicles do not necessarily need to take unbounded deceleration as this will be demonstrated in Sect. 2.3, since the conflicts are resolved before they enter the intersection. The feasibility of the problem in Eq. (10) under bounded deceleration is left as future work.

Given the optimal solution s_i^* of the problem in Eq. (10), the expected time slot $[\mathbb{T}_{i,l}^{in,n}, \mathbb{T}_{i,l}^{out,n}]$ for vehicle i to occupy the conflict zone C_l is computed as:

$$\mathbb{T}_{i,l}^{in,n} := \min_{s_i^*(t)\in\mathcal{L}_{i,l}} t \geq T_{i,l}^{in,n}, \quad \mathbb{T}_{i,l}^{out,n} := \max_{s_i^*(t)\in\mathcal{L}_{i,l}} t \leq T_{i,l}^{out,n} \tag{11}$$

If $\mathcal{L}_{i,l} = \emptyset$, then $\mathbb{T}_{i,l}^{in,n} := \infty$ and $\mathbb{T}_{i,l}^{out,n} := -\infty$. If vehicle i has entered or left C_l, then $\mathbb{T}_{i,l}^{in,n}$ and $\mathbb{T}_{i,l}^{out,n}$ are chosen as the time that it entered or left C_l, respectively.

2.2.3 Theoretical Guarantees

Here, we introduce the theoretical results to show that the proposed strategy solves the conflicts safely and efficiently in real time. The physical feasibility of the trajectories is verified in the motion planning part. Proposition 1 ensures that the passing order is completely determined. Proposition 2 states that there is no deadlock for any pair of vehicles that pass through a same conflict zone at every time step. Proposition 3 shows that a stable consensus on conflict-resolution can be reached in finite time steps. The proofs can be found in [7].

Proposition 1 (Completeness) *For any j that has spacial conflicts with i, at least one statement is true: "i yields j" or "j yields i". In other words, $j \in \mathcal{U}_i$ implies $j \in \mathcal{Y}_i$ or $i \in \mathcal{Y}_j$.*

Proposition 2 (Deadlock-Freeness) *There is no cycle in the directed graph $\mathcal{Y}$ of passing order.*

Proposition 3 (Finite Time Convergence) *If S_i and $\mathcal{U}_i$ remain the same for all i for more than N time steps, then $\mathcal{Y}_i^n$ and $[\mathbb{T}_{i,l}^{in,n}, \mathbb{T}_{i,l}^{out,n}]$ converge in at most N steps to $\mathcal{Y}_i^*$ and $[\mathbb{T}_{i,l}^{in*}, \mathbb{T}_{i,l}^{out*}]$ such that*

$$\mathbb{T}_{i,l}^{in*} \geq \mathbb{T}_{j,l}^{out*} + \Delta_{\mathcal{S}_i}, \ \forall l, \ \forall j \in \mathcal{Y}_i^* \tag{12}$$

Proposition 3 implies that if the sampling time is short enough compared with the time needed between two transitions of S_i's, the system can still reach consensus when S_i's are changing. Nonetheless, after a transition of some S_i, the system needs several steps to settle down. The consistency of the passing orders $\mathcal{Y}^n$ considering those transitions is more intricate to prove, which is left as future work. Indeed, the consistency is demonstrated in simulation.

2.3 Simulation Results

In this section, we illustrate the performance of the proposed distributed conflict resolution mechanism through extensive traffic simulations. The sampling time in the system is chosen to be $dt = 0.1$ s. The robustness margins are chosen as $\Delta_{IL} = 0.5$ s, $\Delta_{FIL} = 0.3$ s and $\Delta_I = 0.1$ s. The priority score P for a vehicle is chosen to be the time that the vehicle stays in the control area. If there is a tie, then the vehicle with smaller ID has the priority. The cost function of the vehicle penalizes (1) the deviation from a target speed, (2) the magnitude of acceleration or deceleration, (3) the magnitude of jerk, and 4) the time spent in every conflict zone. The target speed varies for different vehicles.

The simulation environment is a narrow four-way intersection as shown in Fig. 1a. There is only one incoming lane and one outgoing lane in every direction. Four conflict zones are identified. The control area is the whole graph. For any $i \neq j$, there is a path from lane i to lane $-j$, so there are 12 different paths. Right turn paths only go through one conflict zone. Straight paths go through two conflict zones. Left turn paths go through all four conflict zones (a vehicle is treated as a 2D object instead of a point). In the following discussion, a microscopic case study is presented first followed by the result of macroscopic traffic simulation.

2.3.1 Microscopic Case Study

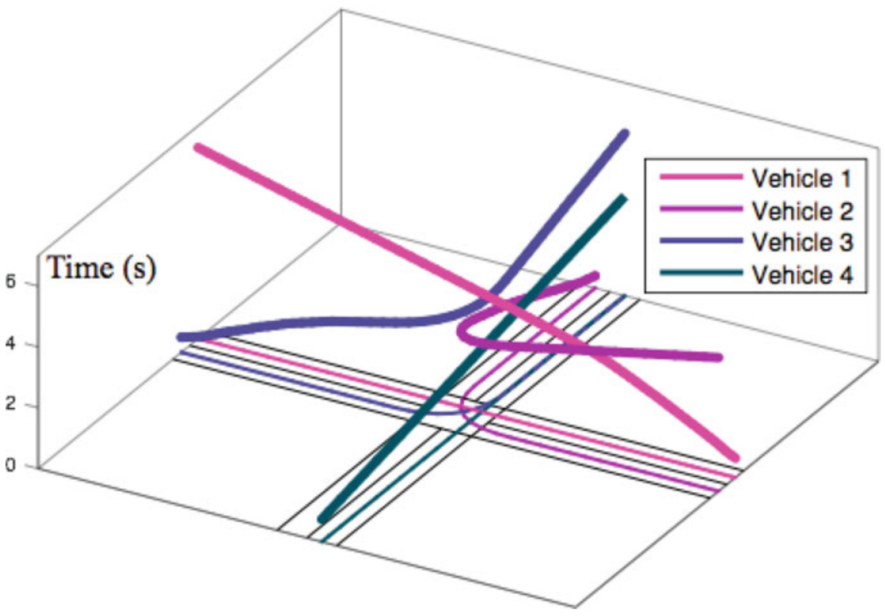
In the case study, there are four vehicles. The conditions of the vehicles (target speed, current lane, target lane, and time to enter the control area) are shown in Table 1. The paths and the executed trajectories are shown in the time-augmented space in Fig. 4a. The planned speed profiles in different time steps are shown in Fig. 4b. The left most speed profile in every subplot is the traffic-free speed profile and the others are the replanned speed profiles given the temporal constraints. Figure 5 shows the expected time intervals (the colored thick bars) for the vehicles to occupy the conflict zones. The thin vertical line indicates the current time.

In this case, vehicles 1, 2 and 3 enter the control area at the same time. According to the traffic-free speed profiles, there are temporal conflicts between vehicle 1 and vehicle 3 in conflict zones 1 and 2, and between vehicle 2 and vehicle 3 in all conflict zones. Since vehicle 2 has a temporal advantage over vehicle 3, vehicle 3 yields vehicle 2. Similarly, vehicle 1 yields vehicle 3. It takes two time steps to resolve the conflicts.

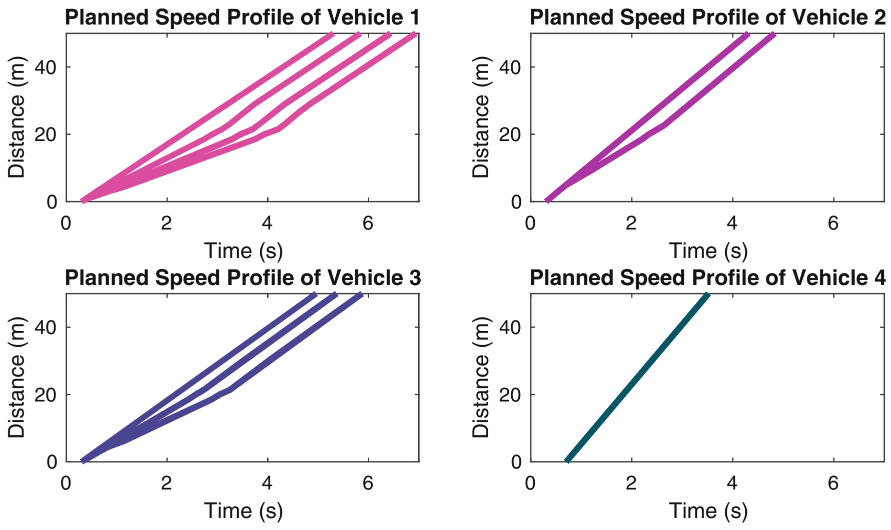
At 0.6 s, vehicle 4 enters, which creates new conflicts. The system settles down after 3 time steps as shown in Fig. 5, which verifies Proposition 3. The planned

Table 1 Conditions in the case study

Vehicle ID	Target speed (m/s)	From	To	Enter time (s)
1	10	Lane 1	Lane -3	0.2
2	12.5	Lane 2	Lane -1	0.2
3	10.75	Lane 3	Lane -2	0.2
4	17.75	Lane 4	Lane -2	0.6



(a)



(b)

Fig. 4 Speed profiles and trajectories in the case study. (**a**) Executed trajectories in the time-augmented space. (**b**) Planned speed profiles in different time steps

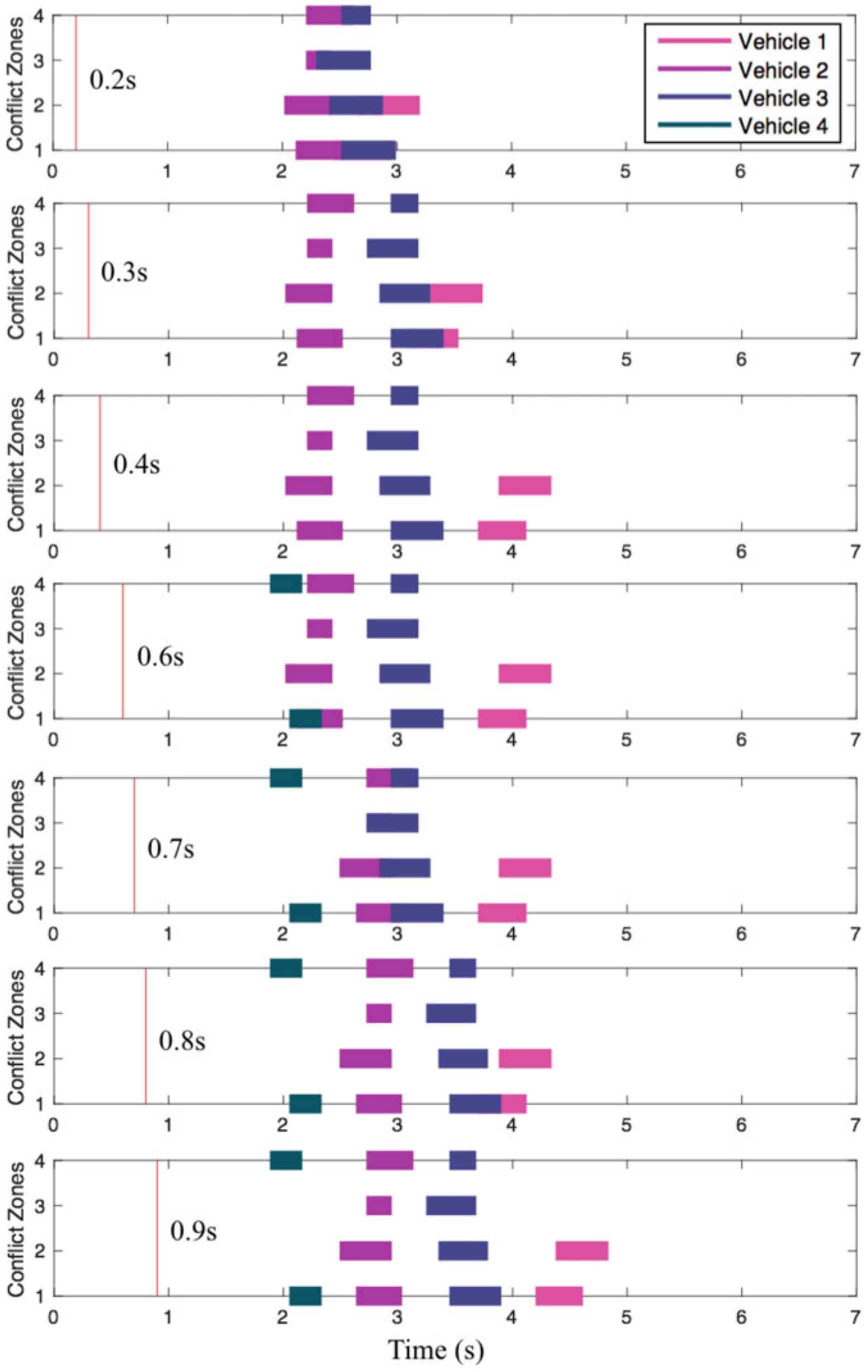


Fig. 5 Conflict resolution in the case study. The scenario in $0.5s$ is omitted since it is the same as the scenario in $0.4s$

speed profiles change accordingly as shown in Fig. 4b. The right most speed profile in each subplot is the executed speed profile.

2.3.2 Macroscopic Traffic Simulation

2.3.2.1 Traffic

In the macroscopic traffic simulation, the traffic is generated at every incoming lane by a Poisson distribution where the density λ is chosen to be 0.5, 0.25 and 0.1, which implies that on average, vehicles arrive every 2, 4, and 10 s. Two groups of traffic are generated:

- Group 1 (G1): 50% of vehicles go straight, 25% turn right and 25% turn left.
- Group 2 (G2): all vehicles go straight.

The second group is introduced to create a relatively fair comparison among performances under distributed strategies and performances under traffic lights. Since we don't have left turn lane or left turn light, when a vehicle wants to turn left, it will block all the vehicles behind, thus significantly increase the delay time. The desired longitudinal speed v_i^r of the vehicle i follows from a uniform distribution from 7.5 to 15 m/s.

2.3.2.2 Comparison

The proposed mechanism is compared against other mechanisms as listed below.

- Case 1 (3D): 3D intersection such as overpass without connectivity. In this case, there is no conflict among vehicles at the intersection. Since the delay is only caused by car following, the simulation result provides a lower bound for the delay time and an upper bound for the throughput.
- Case 2 (NC): unmanaged 2D intersection without connectivity. Vehicles are able to see vehicles from other directions when approaching the intersection. Then vehicles' strategy is: if there is no other vehicles from other directions or other vehicles are too far from the intersection (i.e., there is no temporal conflict even if the other vehicle accelerates with maximum acceleration), cross the intersection without stop; if there are other vehicles from other directions that are close to the intersection, stop and "first stop first go". The delay time in this case is upper bounded by the delay time in the case of a four-way-stop intersection.
- Case 3 (TL-5): 2D intersection with traffic light that changes every 5 s without connectivity. For example, the traffic light for the horizontal direction (lane 1 and lane 3) is green from 0 to 5 s and red from 5 to 10 s while the traffic light for the vertical direction (lane 2 and lane 4) is red from 0 to 5 s and green from 5 to 10 s.
- Case 4 (TL-10): 2D intersection with traffic light that changes every 10 s without connectivity.

- Case 5 (MP-IP): 2D intersection with the maximum progression intersection protocol (MP-IP) [4]. Vehicles broadcast their intentions and estimated time slots to occupy the conflict zones. Conflicting vehicles can make concurrent progress inside the intersection, though low priority vehicles need to yield high priority vehicles, i.e., entering the conflict zones after the high priority vehicles leave. In the simulation, the priority is determined by the priority score P.
- Case 6 (AMP-IP): 2D intersection with the advanced maximum progression intersection protocol (AMP-IP) [4]. In addition to MP-IP, the lower priority vehicles are allowed to cross and clear the conflict zone before the earliest possible arrival of the higher-priority vehicle to that conflict zone.

In Cases 1 to 4, there is no communication among vehicles and the vehicles are equipped with adaptive cruise control for car following. In Cases 5 and 6, vehicles communicate with one another. The two protocols only determine the passing order, not the vehicle trajectories. In the simulation, the vehicles under the two cases adopt the motion planning algorithm discussed in the previous section. The temporal constraints are determined by Eq. (9) according to the passing order. To create a fair comparison, the adaptive cruise control algorithm is integrated into the motion planning algorithm. At each time step, the output of the adaptive cruise control module will be treated as an upper bound on vehicle's acceleration, which is added to the optimization Eq. (10). In the following discussion, we analyze: (1) the average delay time and (2) the throughput in certain time horizon.

2.3.2.3 Average Delay

The delay time of a vehicle is computed as the difference between the actual time and the traffic-free time for the vehicle to travel cross the control area as shown in Fig. 4b. The average delay (mean ± standard deviation) of all vehicles traveled in the control area in 10min under different mechanisms are shown in Table 2. The proposed strategy always outperforms other mechanisms except for the case with 3D intersection which provides a theatrical lower bound of this problem. When the traffic density is low, the performances of Case 2 (without communication) and Cases 5 and 6 (with communication) are similar to the performance of the proposed method, which outperforms the cases with traffic lights. When the traffic density goes up, the performance of Case 2 gets worse dramatically as it almost functions as a stop sign mechanism. The proposed method still outperforms the cases with traffic lights (Cases 3 and 4) since it is more flexible. For example, in the proposed mechanism, four simultaneous right turns are allowed, while in the traffic light case, at most two simultaneous right turns can be tolerated.

The proposed method always outperforms Cases 5 and 6. Though more parallelism inside the intersection area (i.e., allowing more vehicles to cross the intersection at the same time) has been introduced in these two cased compared to Case 2, the rigidity of the priority queue (which does not adjust in real time) limits their performances. For example, consider the case study in Sect. 2.3.1. Since

Table 2 The delay time for traffic in 10 min

	λ	Case 1: 3D	Case 2: NC	Case 3: TL-5	Case 4: TL-10
G1	0.5	0.5 ± 0.8 s	135.6 ± 78.6 s	53.2 ± 30.6 s	57.8 ± 34.4 s
	0.25	0.2 ± 0.4 s	2.9 ± 2.8 s	3.2 ± 2.6 s	5.2 ± 3.8 s
	0.1	0.1 ± 0.2 s	0.4 ± 0.7 s	2.1 ± 2.0 s	3.8 ± 3.9 s
G2	0.5	0.5 ± 0.8 s	134.8 ± 81.0 s	22.0 ± 14.1 s	29.3 ± 17.6 s
	0.25	0.2 ± 0.6 s	9.2 ± 9.3 s	2.8 ± 2.3 s	4.4 ± 3.9 s
	0.1	0.1 ± 0.4 s	0.5 ± 0.7 s	1.9 ± 1.8 s	3.9 ± 3.9 s

	λ	Case 5: MP-IP	Case 6: AMP-IP	Proposed	
G1	0.5	31.2 ± 19.7 s	20.5 ± 13.2 s	11.4 ± 7.0 s	
	0.25	1.9 ± 1.7 s	1.2 ± 1.2 s	0.5 ± 0.7 s	
	0.1	0.4 ± 0.6 s	0.3 ± 0.6 s	0.2 ± 0.3 s	
G2	0.5	8.8 ± 6.4 s	6.3 ± 4.7 s	4.3 ± 3.3 s	
	0.25	2.5 ± 2.9 s	2.2 ± 2.9 s	2.1 ± 2.7 s	
	0.1	0.3 ± 0.5 s	0.3 ± 0.5 s	0.3 ± 0.5 s	

vehicle 4 arrives later than others, it has to wait for others according to MP-IP in Case 5. Even with AMP-IP in Case 6, vehicle 4 wouldn't be able to cut in front of vehicle 2, since it does not leave conflict zone 1 before vehicle 2 enters. Hence high-speed vehicles in Cases 5 and 6 experience larger delay compared to those in the proposed method, where they can cut into the queue only causing other vehicles to slow down slightly. Moreover, the average delay goes up from 8.8 to 52.4 s in Case 5 with "straight only" traffic $\lambda = 0.5$ if the motion planning algorithm is replaced with only adaptive cruise control (ACC). Since the travel time in the intersection is not penalized in ACC, vehicles tend to stop right before the intersection and consequently take longer time to traverse the intersection (as their acceleration is bounded) than they do when they optimize their speed profiles to slow down before approaching the intersection and then speed up to pass the intersection at full speed. Hence the efficiency of the proposed algorithm benefits from both the decision making module (determination of efficient passing order) and the motion planning module (temporal optimization) as well as their integration.

2.3.2.4 Throughput

The throughput is computed as the number of vehicles that cross the control area in a given time slot. The throughput in 10min in all scenarios are shown in Table 3. When the traffic density is high, Case 2 reduces to the case with stop signs. Hence the throughput is roughly upper bounded by $10 \cdot 60/\delta$ where δ is the average time in seconds that is required for a single vehicle to cross the intersection. In the

Table 3 The traffic throughput (# of vehicles) in 10 min

	λ	Case 1: 3D	Case 2: NC	Case 3: TL-5	Case 4: TL-10
G1	0.5	1170	660	984	965
	0.25	590	589	589	587
	0.1	230	230	230	228
G2	0.5	1206	641	1121	1091
	0.25	599	597	595	589
	0.1	245	245	245	245

	λ	Case 5: MP-IP	Case 6: AMP-IP	Proposed	
G1	0.5	1023	1099	1139	
	0.25	590	590	590	
	0.1	230	230	230	
G2	0.5	1166	1186	1199	
	0.25	599	599	599	
	0.1	245	245	245	

simulation, $\delta \approx 1$. Hence the throughput in Case 2 is around 600 when $\lambda = 0.5$, which is much smaller than that in other cases. However, in the proposed method, the throughput almost doubles, which is higher than those in Cases 3 to 6 with traffic light or existing V2V intersection protocols, and is very close to that in Case 1 where the intersection is 3D, thus verifies the effectiveness of the proposed method.

2.4 Conclusion

This section discuss a communication-enabled distributed coordination strategy for connected and autonomous vehicles to navigate at intersections. Based on the received information, a vehicle computes a set of vehicles that it needs to yield and the desired time slots to pass the conflict zones in a decision maker. Then, it computes a desired speed profile according to the desired time slots in a motion planner and broadcasts the estimated times to occupy the conflict zones. The aggregation of these local decisions forms a global solution to a multi-vehicle navigation problem. In the simulation, it is shown that the proposed mechanism has smaller average delay and larger throughput than the comparative cases.

Although the fixed-path assumption and the discrete partitioning of the conflict zone simplifies our problem, they may potentially exclude some feasible conflict resolution strategy that can be achieved by adjusting the vehicle paths. These non-fixed-path strategies are studied in [8, 9]. A thorough analysis and comparison among all these strategies will be left for future work.

3 Centralized Scheduling

In this section, we present a centralized scheduling approach for the problem of intersection management. As shown in Fig. 6, a centralized unit installed in the roadside unit, called intersection manager, decides the passing order of the vehicles periodically. For each period, the intersection manager receives the information from vehicles within its communication range. Based on the received information, the intersection manager computes a time window to each vehicle at each conflict zone on the trajectory of the vehicle. After that, the intersection manager broadcasts these results and prepares for the next period.

The rest of the discussion is organized as follows: Sect. 3.1 presents our timing conflict graph model and problem formulation. Section 3.2 demonstrates our resource conflict model and verification approach. Section 3.3 describes our scheduling algorithm based on cycle removal. Section 3.4 discusses lane merging, a special case of intersection management. Section 3.5 provides experimental results, and Sect. 3.6 concludes the discussion.

3.1 Problem Formulation

In this section, we introduce our graph-based model and formulate the centralized intersection management problem. The notation is summarized in Table 4.

Conflict Zone Same as the definition in Sect. 2, a conflict zone is the crossing location of two trajectories, and two vehicles cannot be at (occupy) the same conflict zone at the same time. There are n conflict zones, $\Xi_1, \Xi_2, \ldots, \Xi_n$, in the intersection. This model allows us to consider different granularities of an intersection, as shown in Fig. 7.

Vehicle Each vehicle has a fixed route—it fixes its source lane, destination lane, and trajectory, and it does not change lanes before and after the intersection. Two

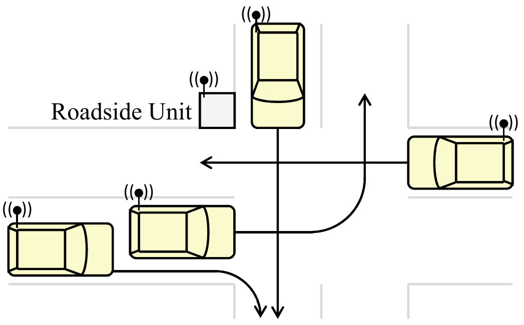


Fig. 6 A centralized unit installed in the roadside unit, called intersection manager, decides the passing order of the vehicles

Table 4 The notation

Index	i, i'	The index of a vehicle
	j, j', j''	The index of a conflict zone
	(i, j)	The index of a vertex
	k	The index of an edge
Given (Input)	G	A timing conflict graph
	m	The number of vehicles
	n	The number of conflict zones
	Δ_i	The i-th vehicle
	Ξ_j	The j-th conflict zone
	$v_{i,j}$	The (i, j)-th vertex
	e_k	The k-th edge
	a_i	The earliest arrival time of Δ_i
	$p_{i,j}$	The vertex passing time of $v_{i,j}$
	w_k	The edge waiting time of e_k
Output	G'	An acyclic timing conflict graph
	$s_{i,j}$	The vertex entering time of $v_{i,j}$

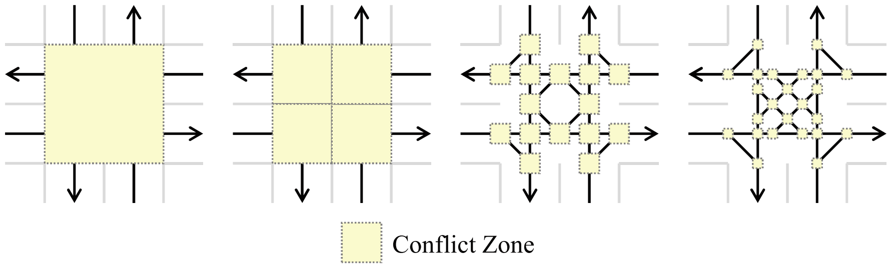


Fig. 7 The model allows us to consider different granularities of an intersection. The intersection can be modeled by 1, 4, 16, and 24 conflict zone(s), and much more alternatives are possible

vehicles have a potential *conflict* at zone Ξ_j if and only if Ξ_j is on the both trajectories.

Timing Conflict Graph A directed timing conflict graph $G = (V, E)$ is constructed by the following rules:

- There is a *vertex* $v_{i,j}$ if and only if Ξ_j is on the trajectory of Δ_i.
- There is a *Type-1 edge* $(v_{i,j}, v_{i,j'})$ if and only if the next conflict zone of Ξ_j on the trajectory of Δ_i is $\Xi_{j'}$.
- There is a *Type-2 edge* $(v_{i,j}, v_{i',j})$ if and only if Δ_i and $\Delta_{i'}$, on the same source lane and with the order where Δ_i is in front of $\Delta_{i'}$, have a conflict at Ξ_j.
- There are two *Type-3 edges* $(v_{i,j}, v_{i',j})$ and $(v_{i',j}, v_{i,j})$ if and only if Δ_i and $\Delta_{i'}$, on different source lanes, have a conflict at Ξ_j.

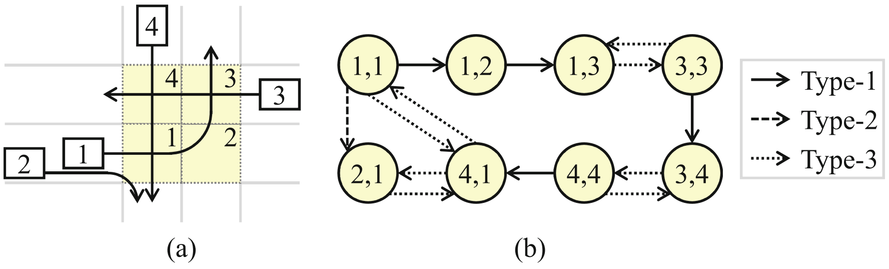


Fig. 8 (**a**) An example and (**b**) its timing conflict graph

Note that the vertex set is a subset of the Cartesian product of the sets of vehicles and conflict zones. An example and its timing conflict graph are shown in Fig. 8a, b, respectively.

Earliest Arrival Time Each vehicle Δ_i is associated with a_i, the earliest arrival time for Δ_i to arrive at the first conflict zone on its trajectory, without being delayed by any other vehicle (i.e., no vehicle is in front of Δ_i before the intersection). It can be either computed or provided by Δ_i or computed by the intersection manager.

Edge Waiting Time Each edge $e_k = (v_{i,j}, v_{i',j'})$ is associated with w_k, the waiting time "length" from Δ_i leaving Ξ_j to $\Delta_{i'}$ entering $\Xi_{j'}$, without being delayed by any other vehicle. For a Type-1 edge e_k (where $i = i'$), w_k is the time from Δ_i leaving Ξ_j to Δ_i entering $\Xi_{j'}$; for a Type-2 or Type-3 edge e_k (where $j = j'$), w_k is the time from Δ_i leaving Ξ_j to $\Delta_{i'}$ entering Ξ_j. In practice, the waiting time of a Type-2 edge e_k is smaller than that of a Type-3 edge $e_{k'}$ as vehicles from the same source lane can perform better in vehicle-following.

Vertex Passing Time Edge vertex $v_{i,j}$ is associated with $p_{i,j}$, the time "length" for Δ_i from entering Ξ_j to leaving Ξ_j.

Vertex Entering Time Each vertex $v_{i,j}$ is associated with $s_{i,j}$, the time for Δ_i to enter Ξ_j, which implies that the earliest time for Δ_i to leave Ξ_j is $s_{i,j} + p_{i,j}$. If a timing conflict graph G' is *acyclic*, the vertex entering time of each vertex is assigned as follows:[2]

- As the graph is acyclic, the assignment can follow a topological order. If there are multiple options, a Type-1 edge has a higher priority than a Type-2 or Type-3 edge.
- If $v_{i,j}$ is the first conflict zone on the trajectory of Δ_i,

[2] As there is dependency between vehicles, the vertex entering time of each vertex cannot be given as an input.

$$s_{i,j} = \max \left\{ a_i, \max_{k|e_k=(v_{i',j'},v_{i,j})\in G'} \left\{ s_{i',j'} + p_{i',j'} + w_k \right\}, \right.$$
$$\left. \max_{k'|e_k=(v_{i',j'},v_{i,j})\in G', e_{k'}=(v_{i',j'},v_{i',j''})\in G', e_k\neq e_{k'}} \left\{ s_{i',j''} - w_{k'} + w_k \right\} \right\} \tag{13}$$

Note that $j = j'$ is always true in this case. The last maximum term is to make sure that $\Delta_{i'}$ leaves $\Xi_{j'}$ for $\Xi_{j''}$ so that Δ_i can enter Ξ_j. For easier understanding, we can also set the intersection-entering point of each source lane as a conflict zone so that it is the first conflict zone of the trajectory of each vehicle from the source lane.

- Otherwise,

$$s_{i,j} = \max \left\{ \max_{k|e_k=(v_{i',j'},v_{i,j})\in G'} \left\{ s_{i',j'} + p_{i',j'} + w_k \right\}, \right.$$
$$\left. \max_{k'|e_k=(v_{i',j'},v_{i,j})\in G', e_{k'}=(v_{i',j'},v_{i',j''})\in G', e_k\neq e_{k'}} \left\{ s_{i',j''} - w_{k'} + w_k \right\} \right\} \tag{14}$$

Note that either $i = i'$ or $j = j'$ is always true in this case. If $i = i'$, the last maximum term is not needed.

Problem Formulation Given a conflict graph G, the earliest arrival time a_i of each vehicle Δ_i, the edge waiting time w_k of each edge e_k, and the vertex passing time $p_{i,j}$ of each vertex $v_{i,j}$, the problem is to

1. Compute an acyclic subgraph G' of G, where
   - For each vertex v_i in G, v_i is also in G',
   - For each Type-1 edge e_k in G, e_k is also in G',
   - For each Type-2 edge e_k in G, e_k is also in G',[3] and
   - For each pair of vertices $v_{i,j}$ and $v_{i',j}$ in G, there exists a path either from $v_{i,j}$ to $v_{i',j}$ or from $v_{i',j}$ to $v_{i,j}$ in G',
2. Guarantee no deadlock,
3. Assign the vertex entering time $s_{i,j}$ of each vertex $v_{i,j}$ (as the paragraph above), and
4. Minimize

$$\max_{v_{i,j}} \left(s_{i,j} + p_{i,j} \right), \tag{15}$$

which is the total time needed for all vehicles to go through the intersection.

[3] We do not consider overtaking in this section; otherwise, we can relax the constraint to potentially change Type-2 edges.

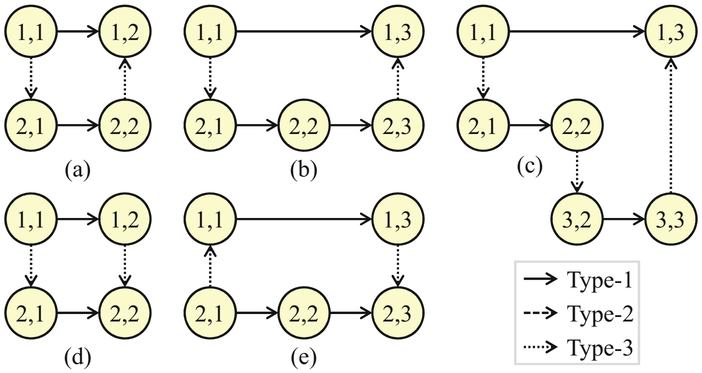


Fig. 9 (**a**)–(**c**) Examples with deadlocks and (**d**)–(**e**) examples without deadlocks

The item 1 is the safety (*collision-freeness*) property to guarantee an order for vehicles having a conflict. The item 3 follows the order to schedule vehicles, and the item 4 is the objective function. The item 2 is the liveness (*deadlock-freeness*) property. To this point, we have not detailed how to guarantee no deadlock—it will be demonstrated in the following section.

3.2 Deadlock-Freeness Verification

In this section, we will demonstrate a graph-based verification approach which can guarantee deadlock-freeness. A tailored Petri net [10] can also verify the deadlock-freeness. The verification can serve as a routine for the scheduling in Sect. 3.3 to verify deadlock-freeness for G'.

Having no cycle in G' or G does not guarantee deadlock-freeness.[4] Some examples are shown in Fig. 9.[5] All of them have no cycle in G', but Fig. 9a–c have deadlocks, and Fig. 9d–e are deadlock-free. In Fig. 9a, Δ_1 needs to enter Ξ_2 after Δ_2. However, Δ_2 even cannot enter Ξ_1(also, Ξ_2) because it is waiting Δ_1 to leave Ξ_1. That causes a deadlock. Similarly, there are deadlocks in Fig. 9b, c. On the contrary, in Fig. 9d, there is no deadlock as Δ_1 enters both Ξ_1 and Ξ_2 before Δ_2. In Fig. 9e, even if Δ_2 enters Ξ_1 first, Δ_2 can enter Ξ_2 after that so that Δ_1 is able to enter Ξ_1 (after Δ_2) and Ξ_3 (before Δ_2) without a deadlock.

[4] This is the reason that we need the item 2 in the problem formulation.

[5] To demonstrate the examples concisely, the examples in Fig. 9 are not associated with any intersection modeling in Fig. 7.

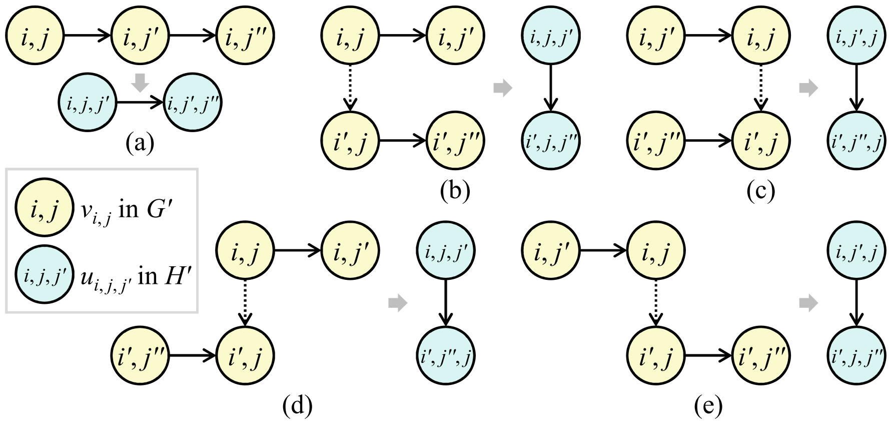


Fig. 10 The construction rules of resource conflict graphs

As illustrated above, having no cycle in G' cannot verify that there is no deadlock. Therefore, we introduce *resource conflict graphs* as follows:

Resource Conflict Graph The directed resource conflict graph H' of G' is constructed by the following rules:

- There is a *vertex* $u_{i,j,j'}$ if and only if there is a Type-1 edge $(v_{i,j}, v_{i,j'})$ in G'.
- If there are edges $(v_{i,j}, v_{i,j'})$ and $(v_{i,j'}, v_{i,j''})$ in G', then there is an edge $(u_{i,j,j'}, u_{i,j',j''})$ in H' (illustrated in Fig. 10a).
- If there are edges $(v_{i,j}, v_{i',j})$, $(v_{i,j}, v_{i,j'})$, and $(v_{i',j}, v_{i',j''})$ in G', then there is an edge $(u_{i,j,j'}, u_{i',j,j''})$ in H' (illustrated in Fig. 10b).
- If there are edges $(v_{i,j}, v_{i',j})$, $(v_{i,j'}, v_{i,j})$, and $(v_{i',j''}, v_{i',j})$ in G', then there is an edge $(u_{i,j',j}, u_{i',j'',j})$ in H' (illustrated in Fig. 10c).
- If there are edges $(v_{i,j}, v_{i',j})$, $(v_{i,j}, v_{i,j'})$, and $(v_{i',j''}, v_{i',j})$ in G', then there is an edge $(u_{i,j,j'}, u_{i',j'',j})$ in H' (illustrated in Fig. 10d).
- If there are edges $(v_{i,j}, v_{i',j})$, $(v_{i,j'}, v_{i,j})$, and $(v_{i',j}, v_{i',j''})$ in G', then there is an edge $(u_{i,j',j}, u_{i',j,j''})$ in H' (illustrated in Fig. 10e).

The general concept of the last four rules is that, if there is an edge $(v_{i,j}, v_{i',j})$ in G', then there is an edge from each vertex (which corresponds to an edge in G') involving $v_{i,j}$ to each vertex (which corresponds to an edge in G') involving $v_{i',j}$ in H'. It implies that, if Δ_i enters Ξ_j before $\Delta_{i'}$ enters Ξ_j, then Δ_i must leave Ξ_j before $\Delta_{i'}$ enters Ξ_j. The resource conflict graphs of the examples in Fig. 9 are shown in Fig. 11. We can observe that they are cyclic in Fig. 11a–c, while they are acyclic in Fig. 11d–e.

Theorem 4 *H' is cyclic if and only if G' has a deadlock.*

Proof From left-hand side (LHS) to right-hand side (RHS): If there is a cycle in H', we assume the cycle as $((i_0, j_0, j'_0), (i_1, j_1, j'_1), \ldots, (i_k, j_k, j'_k), \ldots, (i_l, j_l, j'_l))$,

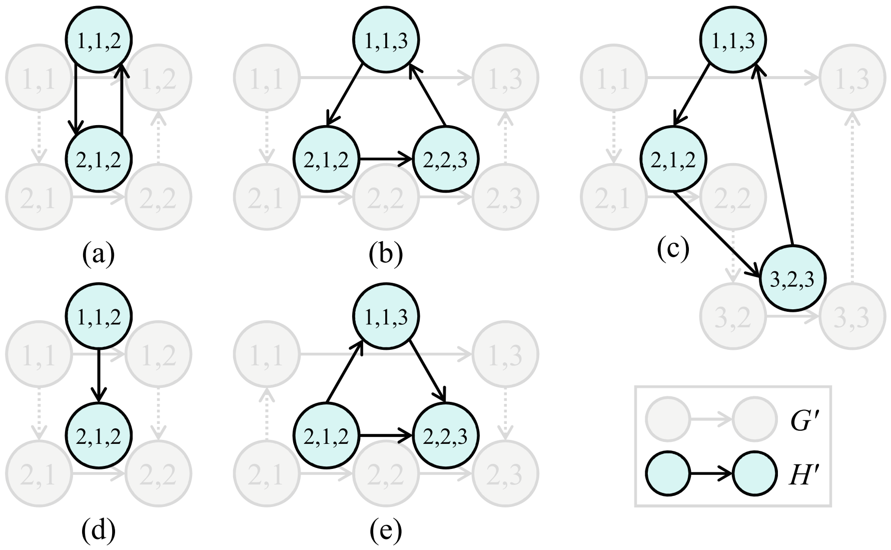


Fig. 11 The resource conflict graphs of the examples in Fig. 9

where $(i_l, j_l, j'_l) = (i_0, j_0, j'_0)$. By the construction rules of H', for any pair of (i_k, j_k, j'_k) and $(i_{k+1}, j_{k+1}, j'_{k+1})$, at least one equality of $j_k = j_{k+1}$, $j_k = j'_{k+1}$, $j'_k = j_{k+1}$, and $j'_k = j'_{k+1}$ is true. Assume that it is equal to j^* in the true equality. By the definition of a conflict zone (that two vehicles cannot be at the same conflict zone at the same time), Δ_{i_k} must leave Ξ_{j^*} before $\Delta_{i_{k+1}}$ enters Ξ_{j^*}. This means that (i_k, j_k, j'_k) blocks $(i_{k+1}, j_{k+1}, j'_{k+1})$, and thus, considering $0 \leq k \leq l-1$, the cycle forms a deadlock.

From RHS to LHS: If there is a deadlock, without loss of generality, we assume that Δ_i cannot move from Ξ_j to $\Xi_{j'}$. The conditions that Δ_i cannot move from Ξ_j to $\Xi_{j'}$ include[6] (1) Δ_i cannot move from another conflict zone $\Xi_{j''}$ to Ξ_j, (2) another vehicle $\Delta_{i'}$ scheduled to enter Ξ_j earlier cannot enter Ξ_j, (3) another vehicle $\Delta_{i'}$ scheduled to leave Ξ_j earlier cannot leave Ξ_j, (4) another vehicle $\Delta_{i'}$ scheduled to enter $\Xi_{j'}$ earlier cannot enter $\Xi_{j'}$, and (5) another vehicle $\Delta_{i'}$ scheduled to leave $\Xi_{j'}$ earlier cannot leave $\Xi_{j'}$. By the construction rules of H', each of the conditions constructs an edge to vertex (i, j, j') in H'. Repeating applying the same conditions, those edges must form a cycle[7] since the numbers of vehicles and conflict zones are finite. □

[6] If all of the conditions are false, then Δ_i can move from Ξ_j to $\Xi_{j'}$. A similar claim is not true for G', so having no cycle in G' cannot guarantee deadlock-freeness.

[7] Though it may not go back to (i, j, j').

By Theorem 4, H' is acyclic if and only if G' has no deadlock (deadlock-freeness). Note that we construct H' from G'. After the construction, we do not need G' in the verification.

3.3 Centralized Scheduling Approach

In this section, we develop a cycle removal algorithm based on the graph model in Sect. 3.1 and the verification approaches in Sect. 3.2.

A greedy strategy, a *First-Come-First-Serve* approach, can be adopted here to schedule the vehicles based on their earliest arrival times. However, this approach ignores the interactions between vehicles and conflict zones, and thus possibly leads to extra waiting time. To address this problem, with the graph-based model and the verification approaches, we can decide the passing order for vehicles to go through the intersection safely and efficiently by removing all cycles in the graph.

The most common method to detect and remove cycles in a directed graph is the Depth-First Search (DFS) algorithm [11]. There is a cycle in a graph only if a *back edge*, which is an edge from a vertex to itself or its ancestors, is found during the DFS traversal of the graph. Then, the method can remove any edge in the cycle to avoid having cycle in the graph. However, without optimization objective, the DFS method may not remove "good" edges to perform optimization. Furthermore, to decide a passing order, we cannot remove some edges because of the safety property (item 1) in our problem formulation, and thus the direct use of a DFS method is not feasible. On the other hand, the minimum feedback arc set problem, a special case of our problem, is NP-hard [12] and has not known to be approximable within a constant [13].

Our objective is to minimize the total time needed for all vehicles to go through the intersection, equivalent to the leaving time of the last vehicle. To remove cycles while considering the edge costs, finding a minimum spanning tree (MST) of the graph can be a potential solution, and one approach is the Kruskal's algorithm [14]. The Kruskal's algorithm repeatedly chooses a minimum-cost edge which does not form any cycle with those already-chosen edges. Kruskal also proposed the backward version of the original one, and it repeatedly removes a maximum-cost edge whose removal does not disconnect the graph. Inspired by this method, we do intend to remove the edge which results in the largest delay to the objective. This can remove cycles and benefit the objective minimization at the same time.

Based on our graph model, we develop a cycle removal algorithm. First, we compute the vertex entering time of each vertex without considering Type-3 edges. Next, the costs of Type-3 edges are estimated by their impacts on the objective. Then, we remove a Type-3 edge which has the largest cost from the graph. The impact of $(v_{i,j}, v_{i',j})$ on the objective is measured by considering $(v_{i,j}, v_{i',j})$ when recomputing the vertex entering time of each vertex. Repeating those steps, we can remove cycles and compute the vertex entering time of each vertex in the graph. It should be noted that, sometimes, we cannot remove an edge because of the last constraint of the item 1 in the problem formulation. In this case, we divide the problem into sub-problems and solve the sub-problems.

3.3.1 Definitions

We first provide some definitions which will be used in our algorithm as follows.

Edge State There are four possible states for an edge:

- An edge is **ON** if it has been decided to be kept (in G'). By the item 1 in the problem formulation, a Type-1 or Type-2 edge is always **ON**. When discussing the graph G', we only consider **ON** edges.
- An edge is **OFF** if it has been decided to be removed.
- An edge is **UNDECIDED** if it is going to be decided in the current sub-problem.
- An edge is **DONTCARE** if it is not considered in the current sub-problem.

Vertex State There are three possible states for a vertex:

- A vertex is **BLACK** if its vertex entering time has been scheduled. If $v_{i,j}$ is **BLACK**, then each edge $e_k = (v_{i,j}, v_{i',j'})$ or $(v_{i',j'}, v_{i,j})$ must be **ON** or **OFF**. On the other hand, if $e_k = (v_{i,j}, v_{i',j'})$ is **ON**, then $v_{i,j}$ must be **BLACK**.
- A vertex is **GRAY** if its vertex entering time can still be influenced by Type-3 edges. If $v_{i,j}$ is **GRAY**, then for each Type-1 or Type-2 edge $e_k = (v_{i',j'}, v_{i,j})$, $v_{i',j'}$ must be **BLACK**. When we remove edges, we only estimate the cost of an edge $e_k = (v_{i,j}, v_{i',j'})$, where at least one of $v_{i,j}$ and $v_{i',j'}$ is **GRAY**.
- A vertex is **WHITE** if its vertex entering time can be influenced by any type of edges.

Vertex Slack The vertex slack is the maximum time which can be delayed at the vertex without increasing the objective. We consider **ON** edges only. Similar to the computation of the vertex entering time, if G' is acyclic, we follow a reverse topological order and compute the vertex slack of each vertex $v_{i,j}$ as follows:

- If Ξ_j is the last conflict zone on the trajectory of Δ_i,

$$slack\,[v_{i,j}] = \min\left\{ \max_{v_{i',j'} \in G'} \left(s_{i',j'} + p_{i',j'}\right) - \left(s_{i,j} + p_{i,j}\right), \right. \\ \left. \min_{k | e_k = (v_{i,j}, v_{i',j'}) \in G'} \left\{slack\,[v_{i',j'}]\right\} \right\} \tag{16}$$

- Otherwise,

$$slack\,[v_{i,j}] = \min_{k | e_k = (v_{i,j}, v_{i',j'}) \in G'} \left\{slack\,[v_{i',j'}]\right\} \tag{17}$$

Algorithm 2 Cycle-removal-based scheduling

```
Input: $G$
Output: $G'$
 1: Initialization;
 2: for each vertex $v_{i,j} \in V$ do
 3:     $state[v_{i,j}] \leftarrow$ WHITE;
 4:     $slack[v_{i,j}] \leftarrow \infty$;
 5: end for
 6: for each edge $e_k \in E$ do
 7:     if $e_k$ is a Type-3 edge then
 8:         $state[e_k] \leftarrow$ UNDECIDED;
 9:     else
10:         $state[e_k] \leftarrow$ ON;
11:     end if
12: end for
13: Update-Time-Slack ($G$);
14: Remove-Type-3-Edges ($G$, 0, $m$);
15: Output the resultant graph as $G'$;
```

Edge Cost The edge cost of a Type-3 edge is the delay time of the objective caused by this edge if we keep it. For the edge $e_k = (v_{i,j}, v_{i',j})$, $(s_{i,j} + p_{i,j} + w_k)$ and $s_{i',j}$ are the vertex entering times of $v_{i',j}$ with and without considering e_k, respectively. If the delay time caused by e_k is larger than the slack of $v_{i',j}$, the objective will increase if we keep e_k. The edge cost of a Type-3 edge $e_k = (v_{i,j}, v_{i',j})$ is defined as follows:

$$cost\ [e_k] = (s_{i,j} + p_{i,j} + w_k) - s_{i',j} - slack\ [v_{i',j}] \tag{18}$$

Note that, although the edge cost may be negative, the objective will never decrease.

3.3.2 Cycle-Removal-Based Scheduling

To solve the cycle removal problem, we follow the steps listed in Algorithm 2. First, based on the problem formulation in Sect. 3.1, Type-1 and Type-2 edges must be included in G'. Thus, we set the states of all Type-1 and Type-2 edges to **ON** and the states of all Type-3 edges to **UNDECIDED**.

Next, we apply Algorithm 3 to compute the vertex entering times and slacks of vertices. At this moment, the graph G' contains only Type-1 and Type-2 edges and thus is an acyclic graph. According to its topological order, we compute the vertex entering time of each vertex by Eqs. (13) and (14) and the leaving time of the last vehicle. We also compute the slack of each vertex according to reverse topological order by Eqs. (16) and (17).

Then, we decide which edges to be removed by Algorithm 4. First, in the process of *Find-Leaders*, a vertex $v_{i,j}$ is identified as a *leader vertex* if Δ_i is the first vehicle of its source lane and Ξ_j the first conflict zone on the trajectory of Δ_i.

Algorithm 3 Update-time-slack

```
Input: $G$
1: Initialization;
2: Topological-Sort ($G$)
3: for each vertex $v_{i,j}$ in topological order do
4:     Compute $s_{i,j}$ by Eq. (13) or (14);
5: end for
6: $maxLeavingTime \leftarrow \max_{v_{i,j}}(s_{i,j} + p_{i,j})$;
7: for each vertex $v_{i,j}$ in reverse topological order do
8:     $slack[v_{i,j}] \leftarrow maxLeavingTime - s_{i,j} - p_{i,j}$;
9:     for each edge $e_k = (v_{i,j}, v_{i',j'}) \in E$ do
10:        $slack[v_{i,j}] \leftarrow \min\{slack[v_{i,j}], slack[v_{i',j'}]\}$;
11:    end for
12: end for
```

Second, an UNDECIDED edge $e_k = (v_{i,j}, v_{i',j})$, i.e., a Type-3 edge, is identified as a *candidate edge* if $v_{i,j}$ or $v_{i',j}$ is a leader vertex. Third, we compute the edge cost of each candidate edge by Eq. (18). Fourth, we try to remove Type-3 edges in descending order of edge cost. Removing edge $e_k = (v_{i,j}, v_{i',j})$ means its reverse edge $e_{k'} = (v_{i',j}, v_{i,j})$ must be included in G' and cannot be removed. As a result, we temporarily set the state of e_k to OFF and $e_{k'}$ to ON. Then, we verify deadlock-freeness for the current G' by the verification approaches in Sect. 3.2. If G' is not deadlock-free, we recover e_k and remove $e_{k'}$ by exchanging their states and verify deadlock-freeness for G' again. If G' is deadlock-free after we decide the states of e_k and $e_{k'}$, we update the states of related vertices, identify newly set GRAY vertices as *leader vertices*, and recompute vertex entering times and slacks. Then, we perform the same process to the next highest cost edge.

However, sometimes G' may have a deadlock no matter we remove either e_k or $e_{k'}$. The reason is that the previous assignments of edges conflict with the decision of choosing e_k or $e_{k'}$. Backtracking the already removed edges is a solution for resolving the dilemma. Unfortunately, the backtracking suffers from a long runtime of finding a valid assignment. Therefore, instead of backtracking the removed edges, we divide the original problem to sub-problems. We partition all vehicles into two parts according to the ascending order of their earliest arrival times. The first part contains vehicles ordered before i_{end}, the second part contains the rest. Consider each pair of vehicles Δ_i and $\Delta_{i'}$, where Δ_i is in the first part, while $\Delta_{i'}$ the second. If Ξ_j is a common conflict zone on the trajectories of Δ_i and $\Delta_{i'}$, we assume that Δ_i will pass zone Ξ_j before $\Delta_{i'}$. The assumption implies the state of edge $(v_{i,j}, v_{i',j})$ is ON and the state of edge $(v_{i',j}, v_{i,j})$ is OFF. Therefore, when solving the sub-problem associated with the first part, we consider only Type-3 edges in between two vehicles belonging to the first part. For both Δ_i and $\Delta_{i'}$ in the first part, the state of their Type-3 edge $(v_{i,j}, v_{i',j})$ is set to UNDECIDED (to be decided in the current sub-problem); for both Δ_i and $\Delta_{i'}$ in the second part, the state of their Type-3 edge $(v_{i,j}, v_{i',j})$ is set to DONTCARE (ignored in the current sub-problem). After we have solved the sub-problem associated with the first part, we turn to the sub-problem

Algorithm 4 Remove-type-3-edges

```
Input: $G, i_{start}, i_{end}$
1: Initialization
2: for each vertex $v_{i,j} \in V$ do
3:     if $order[\Delta_i] \geq i_{start}$ then
4:         $state[v_{i,j}] \leftarrow$ WHITE;
5:     end if
6: end for
7: for each Type-3 edge $e_k = (v_{i,j}, v_{i',j}) \in E$ do
8:     case 1: $order[\Delta_i], order[\Delta_{i'}] < i_{start}$ do   do nothing;
9:     case 2: $i_{end} \leq order[\Delta_i], order[\Delta_{i'}]$ do   $state[e_k] \leftarrow$ DONTCARE;
10:    case 3: $i_{start} \leq order[\Delta_i], order[\Delta_{i'}] < i_{end}$ do   $state[e_k] \leftarrow$ UNDECIDED;
11:    case 4: $i_{start} \leq order[\Delta_i] < i_{end} \leq order[\Delta_{i'}]$ do   $state[e_k] \leftarrow$ ON;
12:    case 5: $i_{start} \leq order[\Delta_{i'}] < i_{end} \leq order[\Delta_i]$ do   $state[e_k] \leftarrow$ OFF;
13: end for
14: $f_{fail} \leftarrow$ FALSE;
15: $LeaderVertices \leftarrow$ Find-Leaders ($i_{start}, i_{end}$);
16: while $LeaderVertices \neq \emptyset$ do
17:     $CandidateEdges \leftarrow$ Find-Candidates ($LeaderVertices$);
18:     for each edge $e_k = (v_{i,j}, v_{i',j})$ in $CandidateEdges$ do
19:         $cost[e_k] \leftarrow s_{i,j} + p_{i,j} + w_k - s_{i',j} - slack[v_{i',j}]$;
20:     end for
21:     $e_{max} \leftarrow$ Find-Max-Cost-Edge ($CandidateEdges$);
22:     $e_{max'} \leftarrow (v_{i',j}, v_{i,j})$ when $e_{max} = (v_{i,j}, v_{i',j})$;
23:     $state[e_{max}] \leftarrow$ OFF;
24:     $state[e_{max'}] \leftarrow$ ON;
25:     if VerifyGraph ($G$) is FALSE then
26:         $state[e_{max}] \leftarrow$ ON;
27:         $state[e_{max'}] \leftarrow$ OFF;
28:         if VerifyGraph ($G$) is FALSE then
29:             $f_{fail} \leftarrow$ TRUE; break;
30:         end if
31:     end if
32:     $LeaderVertices \leftarrow$ Update-Leaders ($LeaderVertices$);
33:     Update-Time-Slack ($G$);
34: end while
35: if $f_{fail}$ is TRUE then
36:     $i_{mid} \leftarrow \frac{1}{2}(i_{start} + i_{end})$;
37:     Remove-Type-3-Edges ($G, i_{start}, i_{mid}$);
38:     Remove-Type-3-Edges ($G, i_{mid}, i_{end}$);
39: end if
```

associated with the second part based on the result derived in previously solved sub-problems. We set the Type-3 edges within the second part to **UNDECIDED** and keep those in the first part unchanged. These procedure is repeated until there are no **UNDECIDED** edges and all the vertices are **BLACK**.Finally, we obtain an acyclic graph G' and schedule the vertex entering time of each vertex in G' by Eqs. (13) and (14).

Theorem 5 *Our scheduling algorithm always finds a feasible solution.*

Proof A feasible solution should satisfy items (i) and (ii) in the problem formulation. Type-1 and Type-2 edges must be included in G', and they do not generate cycles or deadlocks. For Type-3 edges between any pair of $v_{i,j}$ and $v_{i',j}$, only one edge (either $e_k = (v_{i,j}, v_{i',j})$ or $e_{k'} = (v_{i',j}, v_{i,j})$) is selected by our algorithm. If we cannot determine all Type-3 edges at a time, the original problem of all vehicles is recursively divided into two sub-problems according to the ascending order of their earliest arrival times. For Type-3 edges in between two parts, we select only Type-3 edges from the first part to the second part. Hence, only Type-3 edges within one sub-problem have to be discussed. Every time, including a Type-3 edge in one sub-problem is verified by our verification approaches in Sect. 3.2 to guarantee cycle-freeness and deadlock-freeness. In the worst case of sub-problem division, each sub-problem solves only one vehicle. In this case, no Type-3 edges exist in between two vertices belonging to the same vehicle. As a result, the resultant G' is guaranteed to be acyclic and deadlock-free. □

Theorem 6 *The time complexity of our scheduling algorithm is* $O(E^2 \log V)$.

Proof Our scheduling algorithm (Algorithm 2) contains three parts: vertex and edge state initialization, updating vertex entering times and slacks (Algorithm 3), and Type-3 edge removal (Algorithm 4). Vertex/edge state initialization can be done by graph traversal in $O(V+E)$ time. Vertex entering times and slacks can be computed in $O(V+E)$ time based on topological sort and graph traversal. For Type-3 edge removal, assume the induced subgraph for a sub-problem covers a vertex subset $V_s \subseteq V$ and an edge subset $E_s \subseteq E$. The running time of each sub-problem is dominated by the while loop and sub-problem division in Algorithm 4. The while loop examines each Type-3 edge at most once, and the verifier takes $O(V+E)$ time. Thus, the while loop takes a total of $O(E_s(V+E))$ time. The recurrence for the running time $T(V_s, E_s, V, E)$ of Algorithm 4 can be written as $T(V_s, E_s, V, E) = T\left(\frac{V_s}{2}, \alpha E_s, V, E\right) + T\left(\frac{V_s}{2}, \beta E_s, V, E\right) + O(E_s(V+E))$, where $\alpha + \beta \leq 1$. In the base case, every sub-problem contains only one vehicle, and $T(1, E_s, V, E)$ takes $O(E)$ time. The overall running time $T(V, E, V, E)$ of Algorithm 4 is $O(VE + E(V+E)\log V)$. Therefore, our scheduling algorithm takes $O(E^2 \log V)$ time. □

In practical cases, the number of vehicles near an intersection is less than 100, and the experimental results will show the efficiency applicable in real time.

3.4 A Special Case: Lane Merging

Lane merging is the process that vehicles from different incoming lanes merge into one outgoing lane and is one of the major sources causing traffic congestion and delay. For example, in a two-lane merging problem, we have two incoming lanes merging into one outgoing lane. There is no priority for each lane (i.e., no main or secondary lane), and vehicles are not allowed to overtake other vehicles during the process. For two-lane merging, the merging intersection is the sole conflict zone, the merging point is a representative point of the merging intersection. We can optimally solve the two-lane merging scenario by a dynamic programming algorithm. It decomposes the problem into a series of sub-problems to schedule the passing order for vehicles while minimizing the time needed for all vehicles to go through the merging point (equivalent to the time that the last vehicle goes through the merging point). We can extend the problem to a consecutive lane-merging scenario, which is fundamental to further generalization.

3.5 Experimental Results

We implemented the verification approach and scheduling algorithms in the C++ programming language. The experiments were run on a macOS mojave notebook with 2.3 GHz Intel CPU and 8 GB memory. The traffic is generated at every source lane by Poisson distribution where the parameter of Poisson distribution λ is set to as 0.1, 0.3, 0.5, 0.6, and 0.7. The higher λ, the higher traffic density. When $\lambda = 0.1$, the average time interval between two incoming vehicles is 10 s, while it is 2 s when $\lambda = 0.5$. The respective edge waiting time of a Type-1 edge, Type-2 edge, and Type-3 edge is 0.1, 0.2, and 0.2, respectively. The minimum time for a vehicle to pass a conflict zone is set to 1 second, which means a vehicle takes 1 s to pass a conflict zone without considering other vehicles.

3.5.1 Scheduling Effectiveness and Efficiency

In the first experiment, a four-way intersection is considered. For each direction, there is only one incoming lane and one outgoing lane. Four conflict zones are generated according to the crossing locations of four incoming lanes. Two traffic settings are generated. In the first setting, the earliest arrival time of the last vehicle is 30 s, meaning that the intersection manager is required to have a communication range covering all vehicles that will arrive in 30 s. In the second setting, the earliest arrival time of the last vehicle is 60 s. For each vehicle, the probability of going straight, taking a right turn, or taking a left turn is generated by a uniform distribution.

As listed in Tables 5 and 6, the proposed scheduling algorithm is compared with three approaches: (1) 3D-Intersection, (2) First-Come-First-Serve, and (3) Priority-Based. In the 3D-Intersection approach, vehicles do not consider the conflicts with vehicles on other lanes so that a vehicle is delayed only by vehicles on the same lane and in front of it. Thus, the 3D-Intersection approach provides a lower bound for the objective (T_L), although it may not be collision-free. The First-Come-First-Serve approach was introduced in Sect. 3.3. The distributed priority-based approach in [7] is modified to fit in our graph-based model and problem formulation. The Priority-Based approach iteratively decides the passing order of vehicles by their

Table 5 Results under different λ when the earliest arrival time of the last vehicle is 30 s where T_L, T_D, and RT are the leaving time of the last vehicle, the average delay time of all vehicles, and the runtime, respectively (all units are in second)

		3D-intersection			First-come-first-serve		
λ	m	T_L	T_D	RT	T_L	T_D	RT
0.1	11	33.40	0	0.001	33.40	0.00	0.003
0.3	34	40.70	0	0.003	50.70	5.85	0.005
0.5	58	42.40	0	0.006	82.40	19.58	0.009
0.6	66	40.50	0	0.009	90.39	24.65	0.011
0.7	77	46.10	0	0.010	90.20	23.68	0.013
		Priority-Based			Ours		
λ	m	T_L	T_D	RT	T_L	T_D	RT
0.1	11	33.40	0.00	0.009	33.40	0.00	0.002
0.3	34	44.50	3.17	0.007	40.80	2.23	0.015
0.5	58	68.20	10.62	0.013	60.40	6.91	0.057
0.6	66	70.10	12.31	0.020	68.70	13.65	0.119
0.7	77	74.90	13.44	0.024	72.80	13.46	0.174

Table 6 Results under different λ when the earliest arrival time of the last vehicle is 60 s where T_L, T_D, and RT are the leaving time of the last vehicle, the average delay time of all vehicles, and the runtime, respectively (all units are in second)

		3D-Intersection			First-Come-First-Serve		
λ	m	T_L	T_D	RT	T_L	T_D	RT
0.1	25	66.30	0	0.002	68.80	0.48	0.005
0.3	66	68.80	0	0.009	89.19	10.84	0.013
0.5	104	74.00	0	0.015	131.10	26.75	0.020
0.6	129	71.50	0	0.026	149.20	37.62	0.033
0.7	157	72.90	0	0.039	176.50	54.67	0.049
		Priority-Based			Ours		
λ	m	T_L	T_D	RT	T_L	T_D	RT
0.1	25	68.80	0.48	0.008	66.90	0.32	0.006
0.3	66	73.50	2.36	0.015	71.10	1.78	0.070
0.5	104	105.30	12.30	0.052	98.40	11.80	0.229
0.6	129	133.00	27.64	0.091	116.90	20.77	0.626
0.7	157	157.80	38.49	0.157	139.50	34.22	1.825

priorities, and the priorities may change after each iteration. In our experiment, for every 1.0 second, the priorities are updated according to the newly estimated earliest arrival times to intersection.

All approaches are evaluated by two criteria: (1) the leaving time of the last vehicle T_L and (2) the average delay time of all vehicles T_D. T_L is equivalent to the total time needed for all vehicles to go through the intersection. On the other hand, since the 3D-Intersection approach provides the lower bound of T_L, the average delay time of all vehicles T_D is computed as the average of the difference between each vehicle's leaving time and its leaving time in the 3D-Intersection solution. The average delay time of the 3D-Intersection approach itself is always 0.

We demonstrate the effectiveness and efficiency of our algorithm by changing (1) the traffic density and (2) the communication range.

Different Traffic Densities Table 5 shows the impact of traffic density on scheduling. Note that 3D-Intersection provides the lower bounds of T_L and T_D. When λ is 0.1, all approaches can achieve the optimal solution on T_L and T_D due to low traffic density. However, when λ becomes higher, the T_L and T_D of the First-Come-First-Serve approach increase rapidly, and our algorithm can always achieve better results than the First-Come-First-Serve approach. This is because our algorithm considers more vehicles and their interactions, i.e., a global view, and provides a systematic approach to optimize the objective. Only few cases, e.g., $\lambda = 0.6$ or 0.7 when the earliest arrival time of the last vehicle is 30 s, the priority-based approach achieves better T_D than ours. The main reason is that its frequent updates (every 1.0 second) on the earliest arrival times can sometimes mend the lack of a global view. If the update is not fast enough, its effectiveness will decline.

Different Communication Ranges The communication range of an intersection manager is an important factor. To show the flexibility of communication ranges of our algorithm, we compare Table 5 with 6 to observe the results generated by different communication ranges under same λ. In Table 5, the communication range of the intersection manager covers all vehicles that will arrive in 30 s. In Table 6, the communication range of the intersection manager covers all vehicles that will arrive in 60 s. As the communication range becomes twice larger, the T_L of our algorithm also becomes approximately twice larger, which means different communication ranges do not affect the solution quality of our algorithm.

Overall, the proposed scheduling algorithm always achieves better solutions than the First-Come-First-Serve approach under different scenarios. Our algorithm is sufficiently efficient for real-time use even when the number of vehicles reaches 100, which can be completed in around 1 second.[8] As the number of vehicles exceeds 100, the runtime grows up. However, the number of vehicles in an intersection will not exceed 100 in most cases. Even if the number of vehicles is large, we can still split the traffic and schedule the front vehicles first because it is impossible for 100 vehicles to go through the intersection in 1 second.

3.5.2 Modeling Expressiveness

In the second experiment, we show the expressiveness and generality of our modeling for different granularities of an intersection. The four-way intersection is modeled by 1 (like the previous work [7]), 4, and 16 conflict zone(s) as shown in Fig. 7. As shown in Table 7, when λ is low, different granularities of an intersection lead to near-optimal solutions because of few conflicts between vehicles. However,

[8] It is believed that an intersection manager has much better computational capability than a current vehicle.

Table 7 Results of the proposed algorithm under different numbers of conflict zones, where T_L, T_D, and RT are the leaving time of the last vehicle, the average delay time of all vehicles, and the runtime, respectively (all units are in second)

		1 Conflict Zone			4 Conflict Zones			16 Conflict Zones		
λ	m	T_L	T_D	RT	T_L	T_D	RT	T_L	T_D	RT
0.1	11	33.40	0.09	0.004	33.40	0.00	0.002	34.50	0.72	0.004
0.3	34	50.30	6.29	0.016	40.80	2.23	0.014	44.20	3.05	0.020
0.5	58	77.70	16.87	0.092	60.40	6.91	0.057	51.40	5.13	0.103
0.6	66	89.00	24.03	0.188	68.70	13.65	0.119	55.80	6.34	0.134
0.7	77	100.70	28.84	0.284	72.80	13.46	0.174	64.20	9.37	0.769

when λ becomes higher, the intersection modeled by 4 conflict zones always has better solutions than that modeled by 1 conflict zone. Similarly, intersection modeled by 16 conflict zones has better solution than those modeled by 1 and 4 conflict zone(s) in most cases. The finer granularity of an intersection, the more delicate intersection modeling and solution space, and thus the better scheduling results. It should be mentioned that we provide general modeling, scheduling, and verification for intersection management, and they can further assist intersection designers (i.e., governments or city planners) to design intersections (e.g., the number of conflict zones, the passing speed, the safety gap, the communication range, etc.).

3.6 Conclusion

In this section, we propose a timing conflict graph model for centralized intersection management. The model is very general and applicable to different granularities of intersections and other conflicting scenarios. We devise a resource conflict graph for formally verifying deadlock-freeness. Based on the graph-based models, we develop a cycle removal algorithm to schedule vehicles to go through the intersection safely (without collisions) and efficiently without deadlocks. The algorithm is sufficiently efficient to consider more conflict zones and more vehicles in real time. Experimental results demonstrate the expressiveness of the proposed model and the effectiveness and efficiency of the proposed algorithm.

4 Summary

In this chapter, we consider connected and autonomous vehicles at intersections and introduce distributed and centralized approaches solving the problem of intersection management. The approaches provide feasibility, safety (collision-freeness), liveness (deadlock-freeness), stability, efficiency, and real-time decision. Distributed and centralized approaches have their own advantages and disadvantages. We believe that they are suitable for different intersections. For example, a distributed approach for a small intersection; a centralized approach for a large intersection. The trade-offs between different factors and properties should be handled to match the real-world scenarios.

References

1. Liu, C., Tomizuka, M.: Enabling safe freeway driving for automated vehicles. In: 2016 American Control Conference, pp. 3461–3467. IEEE, Piscataway (2016)
2. Liu, C., Chen, J., Nguyen, T.-D., Tomizuka, M.: The robustly-safe automated driving system for enhanced active safety. Technical report, SAE Technical Paper (2017)
3. Dresner, K., Stone, P.: Multiagent traffic management: an improved intersection control mechanism. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 471–477. ACM, New York (2005)
4. Azimi, S., Bhatia, G., Rajkumar, R., Mudalige, P.: Reliable intersection protocols using vehicular networks. In: Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems (ICCPS'13), pp. 1–10. ACM, New York (2013). https://doi.org/10.1145/2502524.2502526
5. Pandit, K., Ghosal, D., Zhang, H.M., Chuah, C.-N.: Adaptive traffic signal control with vehicular ad hoc networks. IEEE Trans. Veh. Technol. **62**(4), 1459–1471 (2013)
6. Liu, C., Zhan, W., Tomizuka, M.: Speed profile planning in dynamic environments via temporal optimization. In: 2017 IEEE Intelligent Vehicles Symposium (IV), pp. 154–159. IEEE, Piscataway (2017)
7. Liu, C., Lin, C.-W., Shiraishi, S., Tomizuka, M.: Distributed conflict resolution for connected autonomous vehicles. IEEE Trans. Intell. Veh. **3**(1), 18–29 (2017)
8. Zhou, H., Liu, C.: Distributed motion coordination using convex feasible set based model predictive control. In: International Conference on Robotics and Automation (ICRA 2021) (2021)
9. An, J., Giordano, G., Liu, C.: Flexible MPC-based conflict resolution using online adaptive ADMM. In: European Control Conference (ECC 2021) (2021)
10. Peterson, J.L.: Petri nets. ACM Comput. Surv. **9**(3), 223–252 (1977)
11. Kleinberg, J., Tardos, E.: Algorithm Design. Pearson, Addison Wesley, London (2006)
12. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations. The IBM Research Symposia Series, pp. 85–103 (1972)
13. Kann, V.: On the approximability of np-complete optimization problems. Ph.D. Thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm (1992)
14. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Am. Math. Soc. **7**(1), 48–50 (1956)