Semi-autonomous Intersection Collision Avoidance through Job-shop Scheduling

Heejin Ahn
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA, 02139
hjahn@mit.edu

Domitilla Del Vecchio
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA, 02139
ddv@mit.edu

ABSTRACT

In this paper, we design a supervisor to prevent vehicle collisions at intersections. An intersection is modeled as an area containing multiple conflict points where vehicle paths cross in the future. At every time step, the supervisor determines whether there will be more than one vehicle in the vicinity of a conflict point at the same time. If there is, then an impending collision is detected, and the supervisor overrides the drivers to avoid collision. A major challenge in the design of a supervisor as opposed to an autonomous vehicle controller is to verify whether future collisions will occur based on the current drivers choices. This verification problem is particularly hard due to the large number of vehicles often involved in intersection collision, to the multitude of conflict points, and to the vehicles dynamics. In order to solve the verification problem, we translate the problem to a job-shop scheduling problem that yields equivalent answers. The job-shop scheduling problem can, in turn, be transformed into a mixed-integer linear program when the vehicle dynamics are first-order dynamics, and can thus be solved by using a commercial solver.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Control theory, Scheduling*; I.2.9 [**Artificial Intelligence**]: Robotics—*Autonomous vehicles*

General Terms

Theory, Algorithms

Keywords

Intersection collision avoidance, multi-vehicle control, supervisory control, collision detection, verification, scheduling

1. INTRODUCTION

In the United States, 33,561 people lost their lives in vehicle crashes in 2012, and 26 % of them occurred at or near intersections [25]. This raises the need for improved safety systems that actively prevent collisions at intersections. For example, a centralized controller could be implemented on the infrastructure to coordinate vehicles near an intersection so as to prevent collisions. However, since a large number of vehicles are often involved in intersection collisions and vehicles are dynamic agents, the design of such systems faces challenges in terms of computational complexity. An additional substantial complication is that the system should override the drivers only when their driving will certainly cause a collision. That is, override actions should be minimally restrictive. This allows drivers to be in control of the vehicle unless unable to handle a dangerous situation. This supervisor can also be used as a safety guard for future fully autonomous vehicles driving in complex environment.

In this paper, we design a supervisor, which can be imple-

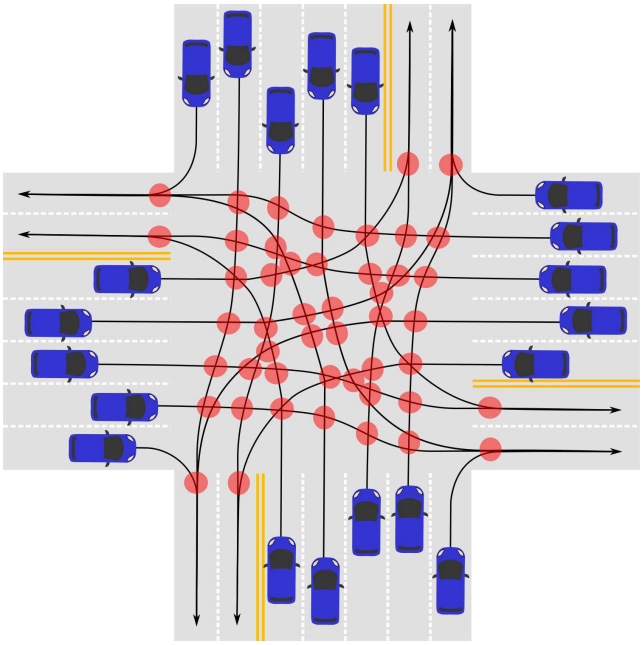


Figure 1: General intersection scenario, taken from [24] to encompass the most dangerous intersections in Massachusetts, USA. This intersection contains forty eight conflict areas (small red circles). The supervisor designed in this paper can prevent collisions at the conflict areas by minimally overriding the vehicles.

mented on an infrastructure, communicating with human-driven vehicles near an intersection as shown in Figure 1. The most important and challenging part in the design is to determine whether vehicles' current driving will cause collisions at some future time. This is important because the exact collision detection, called the *verification problem*, makes the supervisor least restrictive. This problem is not scalable with respect to the number of vehicles near an intersection yet their future safety must be verified every τ seconds, where τ is usually 100 ms [20]. To solve the verification problem in real-time, we formulate a job-shop scheduling problem, and prove that this is equivalent to the former problem. Although the job-shop scheduling problem is NP-hard [15], we can solve this problem using a commercial solver by converting it into a mixed-integer linear programming problem.

Mixed integer programming can handle both discrete and continuous aspects of a system. For example, collision avoidance can be formulated using discrete variables while the dynamic behaviors of vehicles, such as position and speed, are represented by continuous variables. Thus, mixed-integer programming has been employed in various collision avoidance applications such as air traffic control [27, 4, 9] and multi-robot control [13, 17]. Since the decision variables of these works are control inputs, for example, velocity, acceleration, or heading angle, at each time step within a finite time horizon, the discrete-time dynamics of vehicles are considered. As the number of time steps increases, the discretization error is diminished whereas the problem becomes larger and more difficult to solve. Because of this computational complexity, real-time verification is usually not feasible and hence not considered. Moreover, these works are cast in an autonomous framework in which if one input that satisfies the constraints is found, then it is applied. In contrast, in a semi-autonomous framework, such as ours, all admissible inputs need to be examined to determine if at least one feasible input exists.

In collision avoidance confined to an intersection, complexity can be mitigated by exploiting the fact that vehicles tend to follow predetermined paths. Given this, the intersection can be considered as a resource that all vehicles share. In [22, 23, 7], vehicles are assigned time slots during which they can be inside the intersection without conflict. Since the decision variables are the times at which each vehicle enters the intersection, the continuous dynamics are employed to compute these times. Notice that this approach considers n decision variables if n is the number of vehicles, whereas the approach in the previous paragraph considers at least $n * N$ decision variables if N is the number of time steps on a finite time horizon. Because of the significantly smaller number of decision variables, the scheduling approach is computationally more efficient. The above works also assume full autonomy, which is not applicable to the scenarios considered in this paper. A detailed review of autonomous intersection management can be found in [8].

A semi-autonomous framework with the scheduling approach is considered in [10, 11] by proving the equivalence between the verification problem and the scheduling problem. In these works, the authors design a least restrictive supervisor and restrict their attention to a special intersection scenario where all paths of vehicles intersect at one conflict area as indicated by the dashed region in Figure 2. While maintaining the same structure of the supervisor as in [10, 11], we formulate a *job-shop scheduling problem* to account for general scenarios of an intersection, where the paths of vehicles intersect at multiple points as in Figure 1. Considering multiple conflict points enables us to design a less conservative verification problem, but makes it more difficult to translate the problem to a job-shop scheduling problem. In this paper, we prove that our job-shop scheduling problem is equivalent to the verification problem with multiple conflict points. By virtue of this proof, we can solve the verification problem by solving the job-shop scheduling problem, which is computationally tractable. The job-shop scheduling problem is then transformed into a mixed-integer linear programming problem by assuming the single integrator dynamics of vehicles. Although a mixed-integer linear programming problem is NP-hard [15], it can be solved by commercial solvers such as CPLEX [21] or Gurobi [18].

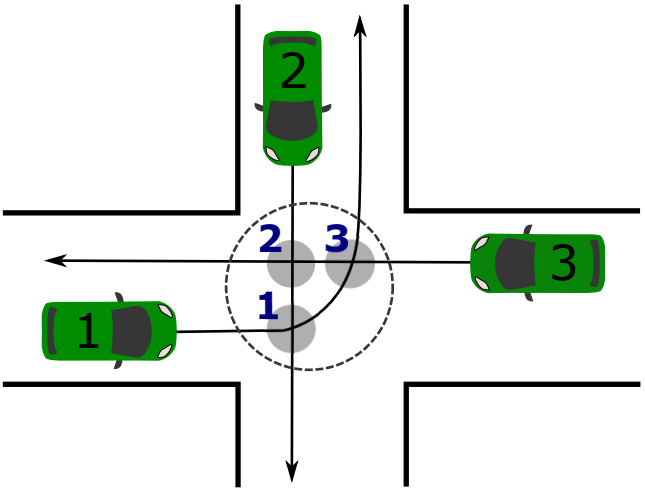


Figure 2: Example of three vehicles with three conflict areas. The dashed circle represents the intersection model used in [10, 11]. In this paper, the intersection is modeled as multiple conflict areas as represented by the three shaded circles.

The rest of this paper is organized as follows. In Section 2, we introduce the intersection model and the dynamic model of vehicles. In Section 3, we formally state the verification problem and the supervisor-design problem. The verification problem can be solved by formulating and solving a job-shop scheduling problem, which plays the most important role in the design of the supervisor. We then transform the job-shop scheduling problem into a mixed-integer linear programming problem to solve the job-shop scheduling problem using a commercial solver. These solutions will be given in Section 4. We conclude this paper by presenting the results of computer simulations in Section 5 and conclusions in Section 6.

2. SYSTEM DEFINITION

Let us consider n vehicles approaching an intersection. The vehicles follow their predetermined paths, and a point at which at least two of the paths intersect is defined as a *conflict point*. Around a conflict point, we define a *conflict area* to account for the size of vehicles. The intersection is modeled as a set of m conflict areas as in Figures 1 and

2. Throughout this paper, vehicles and conflict areas are distinguished by integer indexes $1,\dots,n$ and $1,\dots,m$, respectively. In order to focus only on intersection collision, we assume that there is only one vehicle per road.

To model the longitudinal dynamics of vehicles, let $x_j \in X_j$ be the dynamic state of vehicle j. Let $u_j \in U_j \subset \mathbb{R}$ the control input of vehicle j. Then, the longitudinal dynamics are as follows:

$$\dot{x}_j = f_j(x_j, u_j), \qquad y_j = h_j(x_j). \tag{1}$$

The output of the system is the position $y_j \in Y_j$ along the path. Here, u_j is in a compact set, i.e., $u_j \in U_j := [u_{j,min}, u_{j,max}]$. We assume that the output y_j continuously depends on the input u_j. With abuse of notation, let u_j denote the input signal as well as the input value in $\mathbb{R}$. The input signal $u_j \in \mathcal{U}_j$ is a function of time defined as $\{u_j(t) : u_j(t) \in U_j \text{ for } t \geq 0\}$.

Let $x_j(t, u_j, x_j(0))$ denote the state reached after time t with input signal u_j starting from $x_j(0)$. Similarly, let $y_j(t, u_j, x_j(0))$ denote the position reached after time t with input signal u_j starting from $x_j(0)$. The aggregate state, output, input, and input signal are denoted by $\mathbf{x} \in \mathbf{X}, \mathbf{y} \in \mathbf{Y}, \mathbf{u} \in \mathbf{U}$, and $\mathbf{u} \in \mathcal{U}$, respectively.

One of the most important properties of the dynamic model (1) is the order-preserving property. That is, for $u_j(t) \leq u'_j(t)$ for all t, we have $x_j(t, u_j, x_j(0)) \leq x_j(t, u'_j, x_j(0))$ and $y_j(t, u_j, x_j(0)) \leq y_j(t, u'_j, x_j(0))$ for all $t \geq 0$. We will exploit this property in the design of the supervisor, particularly in formulating the job-shop scheduling problem.

3. PROBLEM STATEMENT

Let $(\alpha_{ij}, \beta_{ij}) \subset \mathbb{R}$ denote the location of conflict area i along the longitudinal path of vehicle j. A conflict area is defined around a conflict point such that a collision occurs if more than one vehicle stay in a conflict area at the same time. That is, a collision occurs if $\mathbf{y} \in B$ where

$$\begin{aligned} B := \{\mathbf{y} \in Y :\ & \text{for some } j \text{ and } j', \\ & y_j \in (\alpha_{i,j}, \beta_{i,j}) \text{ and } y_{j'} \in (\alpha_{i,j'}, \beta_{i,j'})\}. \end{aligned} \tag{2}$$

This subset of output B is called the *bad set*, and if $\mathbf{y}(t) \notin B$ for all $t \geq 0$, we consider the system *safe*.

The verification problem is to determine if collisions at an intersection can be prevented at all future time given an initial state. We formally state this problem using the bad set (2) as follows.

PROBLEM 1 (VERIFICATION). *Given $\mathbf{x}(0)$, determine if there exists $\mathbf{u} \in \mathcal{U}$ such that $\mathbf{y}(t, \mathbf{u}, \mathbf{x}(0)) \notin B$ for all $t \geq 0$.*

Now, we design a supervisor as follows. Every time τ, the supervisor receives the measurements of current states of vehicles and drivers' inputs. Based on the measurements, the supervisor determines whether it must override the vehicles at this time step because otherwise there will be no admissible input to avoid collisions at the next time step. This decision can be made by solving the verification problem.

The supervisor-design problem is formulated as follows.

PROBLEM 2 (SUPERVISOR-DESIGN). *At time $k\tau$, given state $\mathbf{x}(k\tau)$ and drivers' input $\mathbf{u}^k_{driver} \in U$, design a supervisor that satisfies the following specifications.*

Spec 1. For time $[k\tau, (k+1)\tau)$, it returns $\mathbf{u}^k_{driver}$ if there exists $\mathbf{u} \in \mathcal{U}$ such that for all $t \geq 0$

$$\mathbf{y}(t, \mathbf{u}, \mathbf{x}(\tau, \mathbf{u}^k_{driver}, \mathbf{x}(k\tau))) \notin B,$$

or returns $\mathbf{u}^k_{safe} \in \mathcal{U}$ otherwise. Here, $\mathbf{u}^k_{safe}$ is defined as the safe input that guarantees the existence of $\mathbf{u}' \in \mathcal{U}$ such that for all $t \geq 0$,

$$\mathbf{y}(t, \mathbf{u}', \mathbf{x}(\tau, \mathbf{u}^k_{safe}, \mathbf{x}(k\tau))) \notin B. \tag{3}$$

Spec 2. It is non-blocking, that is, $\mathbf{u}^k_{safe}$ must exist for any $k > 0$ if $\mathbf{u}^0_{safe}$ exists.

In Problem 2, Spec 1 guarantees that the supervisor is least restrictive, and Spec 2 guarantees that the supervisor always has an input to override vehicles to ensure safety.

4. PROBLEM SOLUTION

In this section, we solve the two problems: the verification problem (Problem 1) and the supervisor-design problem (Problem 2). As a main result, we formulate a job-shop scheduling problem and prove that this problem is equivalent to Problem 1. Before formulating the job-shop scheduling problem in Section 4.2, we introduce classical job-shop scheduling in Section 4.1. In Section 4.2, we also convert the job-shop scheduling problem into a mixed-integer linear programming problem with the assumption of first-order vehicle dynamics. In Section 4.3, the supervisor algorithm satisfying the specifications of Problem 2 is given.

4.1 Classical job-shop scheduling

In classical job-shop scheduling [26], n jobs are processed on m machines subject to the constraints that (a) each job has its own prescribed sequence of machines to follow, and (b) each machine can process at most one job at a time. This can be represented by a disjunctive graph with a set of nodes $\mathcal{N}$ and two sets of arcs $\mathcal{C}$ and $\mathcal{D}$. Here, the sets are defined as follows.

$$\begin{aligned} \mathcal{N} := \{&(i,j) : (i,j) \text{ is the process of job } j \text{ on machine } i \\ &\text{for all } j \in \{1,\dots,n\}\}, \\ \mathcal{C} := \{&(i,j) \to (i',j) : \text{job } j \text{ is must be processed on machine } i \\ &\text{and then on machine } i' \text{ for all } j \in \{1,\dots,n\}\}, \\ \mathcal{D} := \{&(i,j) \leftrightarrow (i,j') : \text{two jobs } j \text{ and } j' \text{ are to be processed} \\ &\text{on machine } i \text{ for all } i \in \{1,\dots,m\}\}. \end{aligned}$$

The arcs in $\mathcal{C}$, called the conjunctive arcs, represent the routes of the jobs, and the arcs in $\mathcal{D}$, called the disjunctive arcs, connect two operations processed on a same machine.

Let $\mathcal{F} \subseteq \mathcal{N}$ denote a set of the first operations of jobs, and $\mathcal{L} \subseteq \mathcal{N}$ denote a set of the last operations of jobs. If each job has only one operation on its route, $\mathcal{N} = \mathcal{F} = \mathcal{L}$.

The scenario in Section 2 can be described in job-shop scheduling by considering vehicles as jobs and conflict areas as machines. For instance, each vehicle in Figure 2 has its own prescribed route. Vehicle 1 crosses conflict area 1 first and then conflict area 3. At most one vehicle can be inside each conflict area at a time, because otherwise collisions occur. The corresponding disjunctive graph is shown in Figure 3.

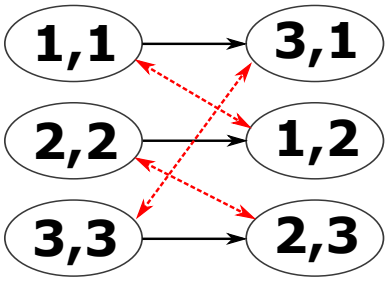


Figure 3: Disjunctive graph of the example in Figure 2. The black solid lines are the conjunctive arcs, and the red dotted lines are the disjunctive arcs.

Example 1. The disjunctive graph in Figure 3 consists of the set of nodes

$$\mathcal{N} = \{(1,1),(3,1),(2,2),(1,2),(3,3),(2,3)\},$$

and the sets of conjunctive and disjunctive arcs

$$\mathcal{C} = \{(1,1) \to (3,1),(2,2) \to (1,2),(3,3) \to (2,3)\},$$
$$\mathcal{D} = \{(1,1) \leftrightarrow (1,2),(2,2) \leftrightarrow (2,3),(3,3) \leftrightarrow (3,1)\},$$

respectively.

The sets of the first and the last operations are $\mathcal{F} = \{(1,1),(2,2),(3,3)\}$ and $\mathcal{L} = \{(3,1),(1,2),(2,3)\}$, respectively.

In [3], as a variant of job-shop scheduling, release times and deadlines are considered such that jobs must start after given release times and be finished before given deadlines. The release time r_j and the deadline d_j are defined for each job j, not for each operation (i,j). The process time p_j is a constant for all operations of job j independent of the machines. Then, the classical job-shop scheduling problem with deadline is formulated as follows.

PROBLEM 3 (CLASSICAL JOB-SHOP). *Given the release times r_j, the deadlines d_j, and the process time p_j, determine if there exists the operation starting times t_{ij} for all $(i,j) \in \mathcal{N}$ such that*

$$\begin{aligned}
&\textit{for all } (i,j) \in \mathcal{F}, && r_j \le t_{ij},\\
&\textit{for all } (i,j) \in \mathcal{L}, && t_{ij} + p_j \le d_j,\\
&\textit{for all } (i,j) \to (i',j) \in \mathcal{C}, && t_{ij} + p_j \le t_{i'j},\\
&\textit{for all } (i,j) \leftrightarrow (i,j') \in \mathcal{D}, && t_{ij} \le t_{ij'} \Rightarrow t_{ij} + p_j \le t_{ij'}.
\end{aligned}$$

In the next section, a new job-shop scheduling problem similar to Problem 3 is formulated to solve Problem 1.

4.2 Solution of Problem 1

4.2.1 Job-shop scheduling

In contrast to classical job-shop scheduling, our problem must account for the dynamic model of vehicles (1). Thus, process times, release times, and deadlines are not initially given and not constant with operation starting times. Also, they are defined for each operation, that is, depending on the jobs and the machines as follows.

Definition 1. Given initial condition $\mathbf{x}(0)$ and schedule $\mathbf{T} := \{T_{ij} \in \mathbb{R} : y_j(T_{ij}, u_j, x_j(0)) = \alpha_{ij}$ for some $u_j \in \mathcal{U}_j,\ \forall (i,j) \in \mathcal{N}\}$, process time P_{ij} is defined for operation $(i,j) \in \mathcal{N}$ as follows.

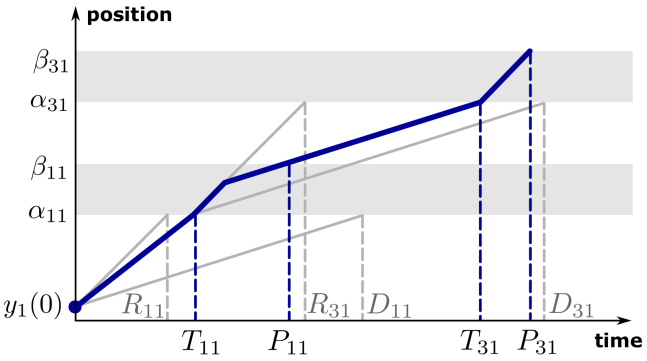


Figure 4: Process time, release time, and deadline of vehicle 1 in the example in Figure 2. The thick blue line represents the position of vehicle 1 on its longitudinal path with the schedules T_{11} and T_{31}. For $(1,1) \notin \mathcal{L}$, P_{11} is a function of T_{11} and T_{31}, while for $(3,1) \in \mathcal{L}$, P_{31} is a function of T_{31} only. Also, for $(3,1) \notin \mathcal{F}$, R_{31} and D_{31} are functions of T_{11}.

- If $(i,j) \in \mathcal{L}$, for $y_j(0) < \alpha_{ij}$,

$$\begin{aligned}
P_{ij} := \min_{u_j \in \mathcal{U}_j} \{t : y_j(t, u_j, x_j(0)) = \beta_{ij}\\
\text{with constraint } y_j(T_{ij}, u_j, x_j(0)) = \alpha_{ij}\}.
\end{aligned} \tag{4}$$

  For $\alpha_{ij} \le y_j(0) < \beta_{ij}$, set $P_{ij} := \min_{u_j}\{t : y_j(t, u_j, x_j(0)) = \beta_{ij}\}$. For $\beta_{ij} \le y_j(0)$, set $P_{ij} = 0$. If the constraint is not satisfied, set $P_{ij} = \infty$.

- If $(i,j) \notin \mathcal{L}$, that is, $\exists (i',j)$ such that $(i,j) \to (i',j) \in \mathcal{C}$, for $y_j(0) < \alpha_{ij}$,

$$\begin{aligned}
P_{ij} := \min_{u_j \in \mathcal{U}_j} \{t : y_j(t, u_j, x_j(0)) = \beta_{ij}\\
\text{with constraint } y_j(T_{ij}, u_j, x_j(0)) = \alpha_{ij}\\
\text{and } y_j(T_{i'j}, u_j, x_j(0)) = \alpha_{i'j}\}.
\end{aligned} \tag{5}$$

  For $\alpha_{ij} \le y_j(0) < \beta_{ij}$, set $P_{ij} := \min_{u_j}\{t : y_j(t, u_j, y_j(0)) = \beta_{ij}$ with constraint $y_j(T_{i'j}, u_j, x_j(0)) = \alpha_{i'j}\}$. For $\beta_{ij} \le y_j(0)$, set $P_{ij} = 0$. If the constraints are not satisfied, set $P_{ij} = \infty$.

By the above definition, process time P_{ij} is the earliest time at which vehicle j can exit conflict area i.

Definition 2. Given initial condition $\mathbf{x}(0)$ and schedule $\mathbf{T} := \{T_{ij} \in \mathbb{R} : y_j(T_{ij}, u_j, x_j(0)) = \alpha_{ij}$ for some $u_j \in \mathcal{U}_j,\ \forall (i,j) \in \mathcal{N}\}$, release time R_{ij} and deadline D_{ij} are defined for operation $(i,j) \in \mathcal{N}$ as follows.

- If $(i,j) \in \mathcal{F}$, for $y_j(0) < \alpha_{ij}$,

$$\begin{aligned}
R_{ij} &:= \min_{u_j \in \mathcal{U}_j} \{t : y_j(t, u_j, x_j(0)) = \alpha_{ij}\},\\
D_{ij} &:= \max_{u_j \in \mathcal{U}_j} \{t : y_j(t, u_j, x_j(0)) = \alpha_{ij}\}.
\end{aligned} \tag{6}$$

  For $\alpha_{ij} \le y_j(0)$, set $R_{ij} = 0$ and $D_{ij} = 0$.

- If $(i,j) \notin \mathcal{F}$, that is, $\exists (i',j)$ such that $(i',j) \to (i,j) \in$

$\mathcal{C}$, for $y_j(0) < \alpha_{ij}$,

$$R_{ij} := \min_{u_j \in \mathcal{U}_j} \{t : y_j(t, u_j, x_j(0)) = \alpha_{ij} \text{ with constraint } y_j(T_{i'j}, u_j, x_j(0)) = \alpha_{i'j}\},$$
$$D_{ij} := \max_{u_j \in \mathcal{U}_j} \{t : y_j(t, u_j, x_j(0)) = \alpha_{ij} \text{ with constraint } y_j(T_{i'j}, u_j, x_j(0)) = \alpha_{i'j}\}. \quad (7)$$

For $\alpha_{ij} \leq y_j(0)$, set $R_{ij} = 0$ and $D_{ij} = 0$. If the constraint cannot be satisfied by any $u_j \in \mathcal{U}_j$, set $R_{ij} = \infty$ and $D_{ij} = -\infty$.

By definition, release time R_{ij} is the earliest time at which vehicle j can enter conflict area i, and deadline D_{ij} is the latest such time.

If an intersection is modeled as a single conflict point as in [10, 11], the process time is defined by (4), and the release time and deadline by (6). This is because each vehicle has a single operation so that $\mathcal{F} = \mathcal{L} = \mathcal{N}$. As for multiple conflict points, we have to include the effect of preceding and succeeding operations in the definition. Notice that the process time P_{ij} in (5) depends on the schedules T_{ij} and $T_{i'j}$ where (i', j) is the succeeding operation of (i, j), and the release time R_{ij} and deadline D_{ij} in (7) depend on $T_{i'j}$ where (i, j) is the preceding operation of (i, j). An example of these definitions is illustrated in Figure 4.

Using the above definitions, we formulate the job-shop scheduling problem as follows.

PROBLEM 4 (JOB-SHOP SCHEDULING). *Given* $\mathbf{x}(0)$*, determine the existence of a schedule* $\mathbf{T} := \{T_{ij} : (i, j) \in \mathcal{N}\} \in \mathbb{R}_+^{|\mathcal{N}|}$ *that satisfies the following constraints.*

$$\text{for all } (i, j) \in \mathcal{N}, \quad R_{ij} \leq T_{ij} \leq D_{ij}, \quad (8)$$
$$\text{for all } (i, j) \leftrightarrow (i, j') \in \mathcal{D}, \quad T_{ij} \leq T_{ij'} \Rightarrow P_{ij} \leq T_{ij'}. \quad (9)$$

Constraint (9) implies avoidance of intersection collisions between vehicles j and j' by ensuring that vehicle j must exit conflict area i before vehicle j' enters it.

We now prove that Problem 1 is equivalent to Problem 4. Before this, we introduce the formal definition of the equivalence between two problems, and prove a lemma that relates Constraint (8) to the existence of an input u_j such that $y_j(T_{ij}, u_j, x_j(0)) = \alpha_{ij}$ and $y_j(P_{ij}, u_j, x_j(0)) = \beta_{ij}$ for $(i, j) \in \mathcal{N}$.

Definition 3. [12] An instance I_A of Problem A is the information required to solve the problem. If I_A satisfies Problem A, we write $I_A \in$ Problem A.

Problem A is reducible to Problem B if for any instance I_A of Problem A, an instance I_B of Problem B can be constructed in polynomial time, and $I_A \in$ Problem A if and only if $I_B \in$ Problem B. If Problem A is reducible to Problem B, and Problem B is reducible to Problem A, then Problem A is equivalent to Problem B.

LEMMA 1. *If* $R_{ij} \leq T_{ij} \leq D_{ij}$ *for all* $(i, j) \in \mathcal{N}$ *with* $y_j(0) < \alpha_{ij}$*, there exists* $u_j \in \mathcal{U}_j$ *such that* $y_j(T_{ij}, u_j, x_j(0)) = \alpha_{ij}$ *and* $y_j(P_{ij}, u_j, x_j(0)) = \beta_{ij}$*.*

PROOF. By the definitions of R_{ij} and D_{ij} in (6), for the first operation (i_1, j) on the route of vehicle j, there exists an input signal u_j such that $y_j(T_{i_1 j}, u_j, x_j(0)) = \alpha_{i_1 j}$. This is because the input space is path-connected, and the output y_j continuously depends on u_j. Then, for the next operation (i_2, j), that is, $(i_1, j) \to (i_2, j) \in \mathcal{C}$, since the constraint in definition (7) is satisfied by the input signal u_j, there is an input signal u_j such that $y_j(T_{i_2 j}, u_j, x_j(0)) = \alpha_{i_2 j}$. By induction on the sequence of operations, for all $(i, j) \in \mathcal{N}$, there exists an input signal u_j such that $y_j(T_{ij}, u_j, y_j(0)) = \alpha_{ij}$.

This input signal u_j satisfies the constraints in the definition of P_{ij} in (4) and (5). Since there exists at least one input signal that satisfies the constraints, an input signal u_j exists such that $y_j(P_{ij}, u_j, x_j(0)) = \beta_{ij}$ for all $(i, j) \in \mathcal{N}$. □

THEOREM 1. *Problem 1 is equivalent to Problem 4.*

PROOF. An instance of Problem 1 is $\{\mathbf{x}(0), \Theta\}$, where $\Theta = \{\{\alpha_{ij}, \beta_{ij} : \forall (i, j) \in \mathcal{N}\}, d, X, Y, U, \mathcal{U}, \mathcal{N}, \mathcal{F}, \mathcal{L}, \mathcal{C}, \mathcal{D}\}$. Notice that an instance of Problem 4 is $\{\mathbf{x}(0), \Theta\}$, which is identical to an instance of Problem 1. Thus, the construction of an instance takes $O(1)$ time. All we have to show is that given $I = \{\mathbf{x}(0), \Theta\}$, $I \in$ Problem 1 if and only if $I \in$ Problem 4.

Suppose $I \in$ Problem 1. Then, there exists $\tilde{\mathbf{u}} \in \mathcal{U}$ such that $\mathbf{y}(t, \tilde{\mathbf{u}}, \mathbf{x}(0)) \notin B$ for all $t \geq 0$. In this proof, we assume $y_j(0) < \alpha_{ij}$. For all $(i, j) \in \mathcal{N}$, let $\tilde{T}_{ij} = \{t : y_j(t, \tilde{u}_j, x_j(0)) = \alpha_{ij}\}$ and $\tilde{P}_{ij} = \{t : y_j(t, \tilde{u}_j, x_j(0)) = \beta_{ij}\}$. We will show that $\{\tilde{T}_{ij} : (i, j) \in \mathcal{N}\}$ satisfies the constraints in Problem 4 so that $\{\mathbf{x}(0), \Theta\} \in$ Problem 4.

By the definitions of R_{ij} and D_{ij}, we have $R_{ij} \leq \tilde{T}_{ij} \leq D_{ij}$ (Constraint (8)). For all $(i, j) \leftrightarrow (i, j') \in \mathcal{D}$, assume without loss of generality vehicle j enters conflict area i before vehicle j'. Then we know that at $t = \tilde{P}_{ij}$, since $y_j(t, \tilde{u}_j, x_j(0)) = \beta_{ij}$, we have $y_{j'}(t, \tilde{u}_j, x_j(0)) \leq \alpha_{ij'}$. That is, $\tilde{P}_{ij} \leq \tilde{T}_{ij'}$. Since $\tilde{u}_j$ satisfies all the constraints given in the definitions of P_{ij}, we have $P_{ij} \leq \tilde{P}_{ij}$. Therefore, $P_{ij} \leq \tilde{T}_{ij'}$ (Constraint (9)).

Suppose $I \in$ Problem 4. Then, there exists $\hat{\mathbf{T}}$ satisfying the constraints in Problem 4. By Lemma 1, there exists $\hat{\mathbf{u}}$ that satisfies $y_j(\hat{T}_{ij}, \hat{u}_j, x_j(0)) = \alpha_{ij}$ and $y_j(P_{ij}, \hat{u}_j, x_j(0)) = \beta_{ij}$ for all $(i, j) \in \mathcal{N}$. In Constraint (9), for all $(i, j) \leftrightarrow (i, j') \in \mathcal{D}$, we have $P_{ij} \leq \hat{T}_{ij'}$ if $\hat{T}_{ij} \leq \hat{T}_{ij'}$. Then, at $t = P_{ij'}$, we have $y_j(t, \hat{u}_j, x_j(0)) = \beta_{ij}$ while $y_{j'}(t, \hat{u}_{j'}, x_{j'}(0)) \leq \alpha_{i'j}$. This implies that any two vehicles never meet inside a conflict area, that is, $\mathbf{y}(t, \hat{\mathbf{u}}, \mathbf{x}(0)) \notin B$ for all $t \geq 0$.

Therefore, there exists $\hat{\mathbf{u}}$ such that $\mathbf{y}(t, \hat{\mathbf{u}}, \mathbf{x}(0)) \notin B$ for all $t \geq 0$. □

By Theorem 1, we can solve Problem 1 by solving Problem 4. One may notice that Problem 4 is similar to the classical job-shop scheduling problem (Problem 3) if $D_{ij} = d_j - p_j$ and $P_{ij} = t_{ij} + p_j$. However, in Problem 4, the release times, deadlines, and process times are defined for each operation as functions of the schedules. The fact that they vary depending on the schedules significantly complicates the problem. We thus cannot directly employ the solutions from the scheduling literature. Instead, we have to formulate a mixed-integer linear programming problem, which is proved to yield the equivalent answers to Problem 4 by assuming that the vehicle dynamics are single integrator dynamics.

4.2.2 Mixed-integer programming

Problem 4 can be transformed into a mixed-integer programming problem, which is an optimization problem subject to equality and inequality constraints in the presence of continuous and discrete variables. Notice that Constraint (8)

is already an inequality constraint. However, Constraint (9) contains a disjunctive constraint, which can be converted into linear inequalities by introducing a binary variable $k_{ijj'} \in \{0,1\}$ and using the big-M method [16]. In particular, define

$$k_{ijj'} := \begin{cases} 1 & \text{if vehicle } j \text{ crosses conflict area } i \\ & \text{before vehicle } j', \\ 0 & \text{otherwise.} \end{cases}$$

Also, let M be a large positive constant in $\mathbb{R}$. Then Constraint (9) can be rewritten as follows:

$$\begin{aligned} &\text{for all } (i,j) \leftrightarrow (i,j') \in \mathcal{D}, \\ &\qquad k_{ijj'} + k_{ij'j} = 1, \quad k_{ijj'}, k_{ij'j} \in \{0,1\} \\ &\qquad P_{ij} \le T_{ij'} + M(1 - k_{ijj'}), \\ &\qquad P_{ij'} \le T_{ij} + M(1 - k_{ij'j}), \end{aligned} \tag{10}$$

for M sufficiently larger than T_{ij} and P_{ij} for all $(i,j) \in \mathcal{N}$. If $k_{ijj'} = 1$ and $k_{ij'j} = 0$, vehicle j crosses conflict area i before vehicle j' so that $T_{ij} \le T_{ij'}$. Then, $P_{ij} \le T_{ij'}$ is imposed while $P_{ij'} \le T_{ij} + M$ is automatically satisfied because of a sufficiently large M. Thus, (10) encodes the same constraint as (9).

Notice that because R_{ij}, D_{ij}, and P_{ij} are functions of variable T_{ij}, Problem 4 with Constraint (8) and (10) is a general mixed-integer program (MIP). Due to its high complexity, this formulation is usually difficult to solve [6]. If the constraints can be expressed in a linear function of variables, the problem becomes a mixed-integer linear program (MILP). Although MILP are combinatorial, several algorithmic approaches are available to solve medium to large size application problems [14].

To this end, we assume that the longitudinal dynamics of vehicles are modeled as a single integrator as follows. For vehicle j,

$$\dot{x}_j = u_j, \qquad y_j = x_j. \tag{11}$$

Notice that the dynamic state $x_j \in X_j \subseteq \mathbb{R}$ is the position, and the control input $u_j \in U_j$ is the speed. Since vehicles do not go in reverse, we let $u_{j,min} > 0$.

With the first order dynamic model (11), we can transform Problem 4 into a mixed-integer linear programming problem. Let us write $P_{ij} = T_{ij} + \min_{u_j}\{t : y_j(t, u_j, \alpha_{ij}) = \beta_{ij}\}$ so that the constraint that $y_j(T_{ij}, u_j, x_j(0)) = \alpha_{ij}$ is automatically satisfied. By defining

$$p_{ij} := \{t : y_j(t, u_j, \alpha_{ij}) = \beta_{ij}\},$$

p_{ij} corresponds to the time spent inside conflict area i, independent of T_{ij}. Then, the variables for the mixed-integer linear programming problem are as follows:

- T_{ij} for $(i,j) \in \mathcal{N}$, continuous variables,
- p_{ij} for $(i,j) \notin \mathcal{L}$, continuous variables,
- $k_{ijj'}$ and $k_{ij'j}$ for $(i,j) \leftrightarrow (i,j') \in \mathcal{D}$, binary variables.

Notice that p_{ij} for $(i,j) \in \mathcal{L}$ is excluded from the variables because we can set $p_{ij} = (\beta_{ij} - \alpha_{ij})/u_{max}$. This is possible because $P_{ij} = T_{ij} + (\beta_{ij} - \alpha_{ij})/u_{max}$ by definition (4), and the minimum p_{ij} is most likely to satisfy the problem formulated in the following paragraph.

Given the single integrator dynamics, we formulate the mixed-integer linear programming problem as follows.

PROBLEM 5. *Given $\mathbf{x}(0)$, determine if there exists a feasible solution subject to the following constraints.*

A. If $(i,j) \in \mathcal{F}$, for $y_j(0) < \alpha_{ij}$

$$\frac{\alpha_{ij} - y_j(0)}{u_{j,max}} \le T_{ij} \le \frac{\alpha_{ij} - y_j(0)}{u_{j,min}}.$$

For $\alpha_{ij} \le y_j(0)$, consider $T_{ij} = 0$.

B. If $(i,j) \notin \mathcal{F}$, that is $\exists (i',j)$ such that $(i',j) \to (i,j) \in \mathcal{C}$,

$$T_{i'j} + p_{i'j} + \frac{\alpha_{ij} - \beta_{i'j}}{u_{j,max}} \le T_{ij} \le T_{i'j} + p_{i'j} + \frac{\alpha_{ij} - \beta_{i'j}}{u_{j,min}}.$$

C. If $(i,j) \notin \mathcal{L}$, for $y_j(0) < \alpha_{ij}$,

$$\frac{\beta_{ij} - \alpha_{ij}}{u_{j,max}} \le p_{ij} \le \frac{\beta_{ij} - \alpha_{ij}}{u_{j,min}}.$$

For $\alpha_{ij} \le y_j(0)$, consider instead $\frac{\beta_{ij} - y_j(0)}{u_{j,max}} \le p_{ij} \le \frac{\beta_{ij} - y_j(0)}{u_{j,min}}$. If $\beta_{ij} \le y_j(0)$, the schedule of operation (i,j) is not of interest.

D. For all $(i,j) \leftrightarrow (i,j') \in \mathcal{D}$, with a large number $M \in \mathbb{R}_+$,

$$\begin{aligned} &T_{ij} + p_{ij} \le T_{ij'} + M(1 - k_{ijj'}), \\ &T_{ij'} + p_{ij'} \le T_{ij} + M(1 - k_{ij'j}), \\ &k_{ijj'} + k_{ij'j} = 1. \end{aligned}$$

We now prove that this problem yields equivalent answers to the job-shop scheduling problem (Problem 4) with the first-order dynamics.

THEOREM 2. *If the vehicle dynamics* (1) *are modeled as* (11)*, Problem 4 is equivalent to Problem 5.*

PROOF. Problem 4 and Problem 5 have an identical instance $I = \{\mathbf{x}(0), \Theta\}$. Thus, we need to show that $I \in$ Problem 4 if and only if $I \in$ Problem 5. We will prove that $I \in$ Problem 4 if $I \in$ Problem 5, and $I \notin$ Problem 4 if $I \notin$ Problem 5.

Suppose $I \in$ Problem 5. Then there exist a feasible solution $(\tilde{\mathbf{T}}, \tilde{\mathbf{p}}, \tilde{\mathbf{k}})$ where $\tilde{\mathbf{T}} = \{\tilde{T}_{ij} : \forall (i,j) \in \mathcal{N}\}$, $\tilde{\mathbf{p}} = \{\tilde{p}_{ij} : \forall (i,j) \notin \mathcal{L}\}$, and $\tilde{\mathbf{k}} = \{\tilde{k}_{ijj'}, \tilde{k}_{ij'j} : \forall (i,j) \leftrightarrow (i,j') \in \mathcal{D}\}$.

For $(i,j) \in \mathcal{F}$, $R_{ij} = (\alpha_{ij} - y_j(0))/u_{j,max}$ and $D_{ij} = (\alpha_{ij} - y_j(0))/u_{j,min}$ by definition (6). For $(i,j) \notin \mathcal{F}$, that is, $\exists (i',j) \to (i,j) \in \mathcal{C}$, there is the constraint in definition (7) that $y_j(\tilde{T}_{i'j}) = \alpha_{i'j}$. Thus, R_{ij} and D_{ij} are as follows.

$$\begin{aligned} R_{ij} &= \tilde{T}_{i'j} + \frac{\alpha_{ij} - \alpha_{i'j}}{u_{j,max}} = \tilde{T}_{i'j} + \tilde{p}_{i'j} + \frac{\alpha_{ij} - \beta_{i'j}}{u_{j,max}}, \\ D_{ij} &= \tilde{T}_{i'j} + \frac{\alpha_{ij} - \alpha_{i'j}}{u_{j,min}} = \tilde{T}_{i'j} + \tilde{p}_{i'j} + \frac{\alpha_{ij} - \beta_{i'j}}{u_{j,min}}. \end{aligned}$$

The second equalities in both equations result from Constraint C. Therefore, Constrains A and B imply $R_{ij} \le \tilde{T}_{ij} \le D_{ij}$ for all $(i,j) \in \mathcal{N}$ (Constraint (8)).

In Constraint D, we have $P_{ij} \le \tilde{T}_{ij} + \tilde{p}_{ij}$ because P_{ij} is the minimum time to reach β_{ij}. Therefore, we have $P_{ij} \le \tilde{T}_{ij} + \tilde{p}_{ij} \le \tilde{T}_{ij'} + M(1 - \tilde{k}_{ijj'})$. Similarly, $P_{ij'} \le \tilde{T}_{ij'} + \tilde{p}_{ij'} \le \tilde{T}_{ij} + M(1 - \tilde{k}_{ij'j})$ (Constraint (10)).

Thus, $\tilde{\mathbf{T}}$ satisfies the constraints in Problem 4. That is, $I \in$ Problem 4.

Suppose $I \notin$ Problem 5. Notice that if Constraint C is ignored and let $p_{ij} = 0$, the problem is always feasible because for $(i_1, j) \in \mathcal{F}$ and $(i_1, j) \to (i_2, j), \ldots, (i_{d-1}, j) \to (i_d, j) \in \mathcal{C}$,

$$T_{i_1,j} = \frac{\alpha_{i_1 j} - y_j(0)}{u_{j,max}},\ T_{i_2 j} = T_{i_1 j} + \frac{\alpha_{i_2 j} - \beta_{i_1 j}}{u_{j,max}},$$
$$\ldots,\ T_{i_d j} = T_{i_{d-1} j} + \frac{\alpha_{i_d j} - \beta_{i_{d-1} j}}{u_{j,max}}$$

becomes a feasible solution for any j. Constraint D is also satisfied because either $T_{ij} \leq T_{i'j}$ or $T_{i'j} \leq T_{ij}$ is always true. We can thus find the maximum process time that is a feasible solution for the problem without Constraint C. Since $I \notin$ Problem 5, this solution violates Constraint C. Thus, there is no $p_{ij} \geq (\beta_{ij} - \alpha_{ij})/u_{j,max}$ for any $(i, j) \notin \mathcal{L}$ such that Constraints A, B, and D are satisfied. This, in turn, implies that given the definition $P_{ij} = T_{ij} + \min p_{ij}$, there is no $P_{ij} \geq T_{ij} + (\beta_{ij} - \alpha_{ij})/u_{j,max}$ such that the constraints in Problem 4 are satisfied. Since P_{ij} is not feasible, neither are T_{ij} and $k_{ijj'}$. Thus, $I \notin$ Problem 4. □

We solve Problem 5 using CPLEX. The procedure that solves Problem 5 given an instance $I = \{\mathbf{x}(0), \Theta\}$ is referred to as $\texttt{Jobshop}(I)$. If $I \in$ Problem 5, that is, $I \in$ Problem 1 by Theorems 1 and 2, $\texttt{Jobshop}(I)$ returns $\{\mathbf{T}, \mathbf{p}, yes\}$. Otherwise, it returns $\{\emptyset, \emptyset, no\}$.

4.3 Solution of Problem 2

The supervisor runs in discrete time with a time step τ. At time $k\tau$ where $k > 0$, it receives the measurements of the states $\mathbf{x}(k\tau)$ and drivers' inputs $\mathbf{u}^k_{driver} \in \mathbf{U}$ of the vehicles near an intersection. By assuming that $\mathbf{u}^k_{driver}$ is constant for time $[k\tau, (k+1)\tau)$, we predict a state at the next time step, called a state prediction and denoted by $\hat{\mathbf{x}}(\mathbf{u}^k_{driver})$, as follows.

$$\hat{\mathbf{x}}(\mathbf{u}^k_{driver}) = \mathbf{x}(\tau, \mathbf{u}^k_{driver}, \mathbf{x}(k\tau)).$$

Notice that $\texttt{Jobshop}(\hat{\mathbf{x}}(\mathbf{u}^k_{driver}), \Theta)$ determines whether or not collisions can be avoided at all future time given the state prediction. If it returns $\{\mathbf{T}, \mathbf{p}, yes\}$, then the supervisor allows the vehicles to drive with input $\mathbf{u}^k_{driver}$ for time $[k\tau, (k+1)\tau)$. The schedule $\mathbf{T}$ and the process time $\mathbf{p}$ are used to generate a safe input signal $\mathbf{u}^{k+1,\infty}_{safe}$, defined on time $[(k+1)\tau, \infty)$. We define a safe input operator $\sigma(\hat{\mathbf{x}}(\mathbf{u}^k_{driver}), \mathbf{T}, \mathbf{p})$ as follows.

$$\sigma(\hat{\mathbf{x}}(\mathbf{u}^k_{driver}), \mathbf{T}, \mathbf{p})$$
$$\in \{(u_1, \ldots, u_n) \in \mathcal{U} : y_j(T_{ij}, u_j, \hat{x}_j(u^k_{driver,j})) = \alpha_{ij}$$
$$\text{and } y_j(p_{ij}, u_j, \alpha_{ij}) = \beta_{ij}\ \forall (i,j) \in \mathcal{N}\}, \quad (12)$$

where $u^k_{driver,j}$ is the j^{th} entry of $\mathbf{u}^k_{driver}$, and $\hat{x}_j(u^k_{driver,j})$ is the j^{th} entry of $\hat{\mathbf{x}}(\mathbf{u}^k_{driver})$. This safe input signal is stored for possible uses at the next time step.

If $\texttt{Jobshop}(\hat{\mathbf{x}}(\mathbf{u}^k_{driver}), \Theta)$ returns $\{\emptyset, \emptyset, no\}$, then the supervisor overrides the vehicles using the safe input signal stored at the previous step, $\mathbf{u}^{k,\infty}_{safe}$. Since $\mathbf{u}^{k,\infty}_{safe}$ is defined on time $[k\tau, \infty)$, let $\mathbf{u}^k_{safe} \in \mathcal{U}$ be $\mathbf{u}^{k,\infty}_{safe}$ restricted to time $[k\tau, (k+1)\tau)$. The supervisor blocks the drivers' inputs $\mathbf{u}^k_{driver}$ and returns the safe input $\mathbf{u}^k_{safe}$ for time $[k\tau, (k+1)\tau)$ to prevent future collisions.

This procedure is written as an algorithm as follows.

Algorithm 1 $\texttt{Supervisor}(\mathbf{x}(k\tau), \mathbf{u}^k_{driver})$

1: $\{\mathbf{T}_1, \mathbf{p}_1, answer_1\} = \texttt{Jobshop}(\hat{\mathbf{x}}(\mathbf{u}^k_{driver}), \Theta)$
2: **if** $answer_1 = yes$ **then**
3: $\quad \mathbf{u}^{k+1,\infty} \leftarrow \sigma(\hat{\mathbf{x}}(\mathbf{u}^k_{driver}), \mathbf{T}_1, \mathbf{p}_1)$
4: $\quad \mathbf{u}^{k+1}_{safe} \leftarrow \mathbf{u}^{k+1,\infty}(t)$ for $t \in [(k+1)\tau, (k+2)\tau)$
5: $\quad$ **return** $\mathbf{u}^k_{driver}$
6: **else**
7: $\quad \{\mathbf{T}_2, \mathbf{p}_2, answer_2\} = \texttt{Jobshop}(\hat{\mathbf{x}}(\mathbf{u}^k_{safe}), \Theta)$
8: $\quad \mathbf{u}^{k+1,\infty} \leftarrow \sigma(\hat{\mathbf{x}}(\mathbf{u}^k_{safe}), \mathbf{T}_2, \mathbf{p}_2)$
9: $\quad \mathbf{u}^{k+1}_{safe} \leftarrow \mathbf{u}^{k+1,\infty}(t)$ for $t \in [(k+1)\tau, (k+2)\tau)$
10: $\quad$ **return** $\mathbf{u}^k_{safe}$
11: **end if**

If $answer_1 = yes$, then the supervisor generates and stores the safe input $\mathbf{u}^{k+1}_{safe}$ in lines 3-4, and does not intervene in line 5. If $answer_1 = no$, the supervisor solves the verification problem in line 7 given the state predicted with the safe input $\mathbf{u}^k_{safe}$. It will be proved in the following theorem that $answer_2$ is always yes, which implies the non-blocking property of the supervisor. Based on $\mathbf{T}_2$ and $\mathbf{p}_2$, the supervisor generates and stores the safe input $\mathbf{u}^{k+1}_{safe}$ in lines 8-9, and overrides the vehicles in line 10.

THEOREM 3. *Algorithm 1 solves Problem 2.*

PROOF. To prove that Algorithm 1 is a solution of Problem 2, we check if the algorithm satisfies the specifications in Problem 2.

Specification 1 is met by the design of the algorithm. If there exists $\mathbf{u} \in \mathcal{U}$ such that $\mathbf{y}(t, \mathbf{u}, \hat{\mathbf{x}}(\mathbf{u}^k_{driver})) \notin B$ for all $t \geq 0$, then $\texttt{Jobshop}(\hat{\mathbf{x}}(\mathbf{u}^k_{driver}), \Theta)$ returns *yes*. In this case, the supervisor returns $\mathbf{u}^k_{driver}$. Otherwise, it returns $\mathbf{u}^k_{safe} \in \mathcal{U}$. The fact that this input makes $\texttt{Jobshop}(\hat{\mathbf{x}}(\mathbf{u}^k_{safe}), \Theta)$ return *yes* will be clear in the proof of the non-blocking property.

To prove the non-blocking property, we use mathematical induction on k where $t = k\tau$. At $t = 0$, we assume $\mathbf{u}^0_{safe} \neq \emptyset$. At $t = (k-1)\tau$, suppose there exists $\mathbf{u}^{k-1}_{safe}$. That is, by definition, there exists $\mathbf{u}' \in \mathcal{U}$ such that $\mathbf{y}(t, \mathbf{u}', \hat{\mathbf{x}}(\mathbf{u}^{k-1}_{safe})) \notin B$ for all $t \geq 0$. If $\texttt{Jobshop}(\hat{\mathbf{x}}(\mathbf{u}^{k-1}_{driver}), \Theta)$ returns *yes*, then then there exists $\mathbf{u} \in \mathcal{U}$ such that $\mathbf{y}(t, \mathbf{u}, \hat{\mathbf{x}}(\mathbf{u}^{k-1}_{driver})) \notin B$ for all $t \geq 0$ by Problem 1.

Now at $t = k\tau$, we want to prove that there exists $\mathbf{u}^k_{safe}$. Notice that $\mathbf{x}(k\tau)$ is either $\hat{\mathbf{x}}(\mathbf{u}^{k-1}_{driver})$ or $\hat{\mathbf{x}}(\mathbf{u}^{k-1}_{safe})$. In the former case, let $\mathbf{u}^k$ be $\mathbf{u}$ restricted to time $[k\tau, (k+1)\tau)$, and $\mathbf{u}^{k+1,\infty}$ be $\mathbf{u}$ restricted to time $[(k+1)\tau, \infty)$. Then, we have

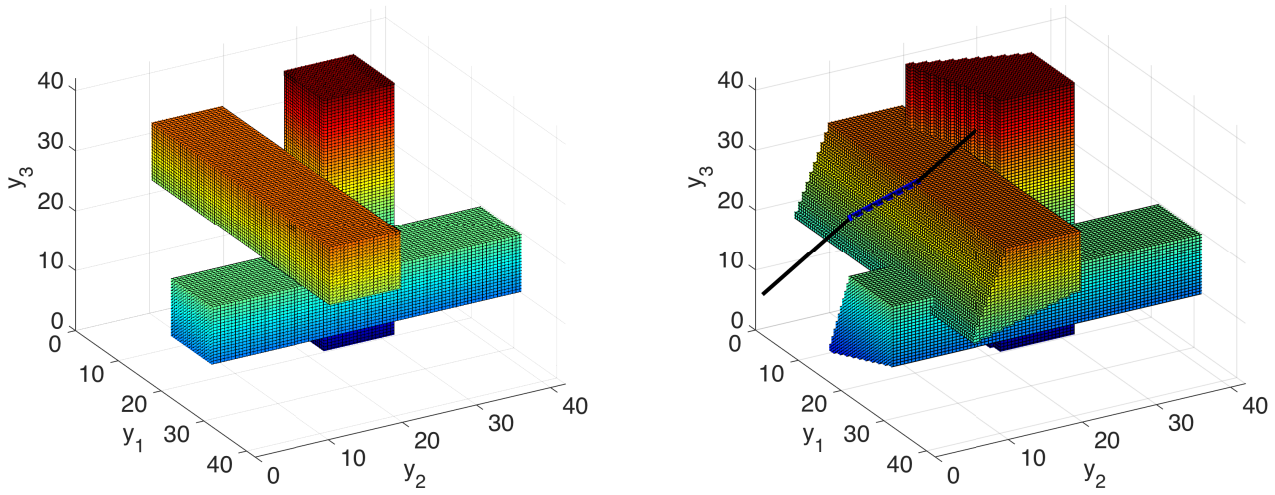
$$\mathbf{y}(t, \mathbf{u}^{k+1,\infty}, \mathbf{x}(\tau, \mathbf{u}^k, \hat{\mathbf{x}}(\mathbf{u}^{k-1}_{driver}))) \notin B.$$

Thus there exists $\mathbf{u}^k_{safe} = \mathbf{u}^k$. Similarly for the latter case, let $\mathbf{u}'^k$ be $\mathbf{u}'$ restricted to time $[k\tau, (k+1)\tau)$, and $\mathbf{u}'^{k+1,\infty}$ be $\mathbf{u}'$ restricted to time $[(k+1)\tau, \infty)$. Then, we have

$$\mathbf{y}(t, \mathbf{u}'^{k+1,\infty}, \mathbf{x}(\tau, \mathbf{u}'^k, \hat{\mathbf{x}}(\mathbf{u}^{k-1}_{safe}))) \notin B.$$

Thus there exists $\mathbf{u}^k_{safe} = \mathbf{u}'^k$. Therefore, in any case, there exists a safe input $\mathbf{u}^k_{safe}$.

If $\mathbf{u}^0_{safe}$ exists, there exists $\mathbf{u}^k_{safe}$ for any $k > 0$. The supervisor is thus, non-blocking. □



(a) Bad set

(b) Capture set

Figure 5: Position space of the three vehicles in the scenario of Figure 2. Subfigure (a) shows the bad set defined in (2), and subfigure (b) shows the resulting capture set defined in (13). In (b), the black line is the trajectory of the system, and the blue thick line highlights the positions at times when the supervisor overrides the vehicles. Notice that the supervisor prevents them from entering the capture set, thereby averting collision.

5. SIMULATION RESULTS

This section presents simulation results of the supervisor. In particular, considering the intersection scenarios illustrated in Figures 1 and 2, we validate that the supervisor prevents impending collisions by minimally overridng vehicles. Also, the simulations illustrate that for a system with a large number of vehicles, the computation time required for the supervisor algorithm (Algorithm 1) at each step is within the allotted 100 ms.

We implement Algorithm 1 using MATLAB, in which mixed-integer programming in Problem 5 is solved by using CPLEX. To speed up the process of generating the constraints of the problem, MATLAB Coder$^{\text{TM}}$[28] is used to replace the code written in MATLAB with the C code and compile it into a MATLAB executable function. Simulations are performed on a personal computer, which runs Windows 7 Home Premium and consists of an Intel Core i7-3770s processor at 3.10 GHz and 8 GB random-access memory.

Consider first Figure 2, in which three vehicles are approaching the intersection containing three conflict points. The parameters used in the simulations are τ =0.1, U_j = $[0.1, 0.3]$ for all $j \in \{1, \ldots, n\}$, $(\alpha_{ij}, \beta_{ij})$ =(10,20) for $(i,j) \in \mathcal{F}$, and $(\alpha_{ij}, \beta_{ij}) = (\alpha_{i'j} + 22, \alpha_{ij} + 10)$ for $(i,j) \notin \mathcal{F}$, where $(i',j) \rightarrow (i,j) \in \mathcal{C}$.

To solve the verification problem (Problem 1), the work in [19] considers the set of initial states such that no input exists to avoid a collision. This subset of the state space is called the *capture set* and defined as follows.

$$\mathcal{CS} := \{\mathbf{x} \in \mathbf{X} : \forall \mathbf{u} \in \mathcal{U},\ \exists t \text{ such that } \mathbf{y}(t, \mathbf{u}, \mathbf{x}) \in B\}. \tag{13}$$

The capture set resulting from the bad set in Figure 5(a) is shown in Figure 5(b). Given an instance $I = \{\mathbf{x}(0), \Theta\}$ of Problem 1, $I \notin$ Problem 1 if and only if $\mathbf{x}(0) \in \mathcal{CS}$ by definition. By Theorems 1 and 2, if $\mathbf{x}(0) \in \mathcal{CS}$, $I \notin$ Problems 4 and 5.

In Figure 5(b), the black line represents the trajectory of the system given an initial condition $\mathbf{x}(0)$ =(-2.8,-3.7,-1.2). When the supervisor overrides the vehicles, the trajectory is shown in blue. The drivers' inputs are set to be $\mathbf{u}^k_{driver} = (0.15, 0.11, 0.25)$ and constant for all $k \geq 0$ where $t = k\tau$, so that without override actions of the supervisor, the trajectory would enter the bad set in Figure 5(a). Notice that the supervisor overrides the vehicles right before the trajectory enters the capture set and makes the trajectory ride on the boundary of the capture set. The drivers regain the control of their vehicles once the dangerous situation is resolved. This confirms that the supervisor is least restrictive because it intervenes only when the state prediction $\hat{\mathbf{x}}(\mathbf{u}^k_{driver})$ enters the capture set. The computation of the supervisor algorithm (Algorithm 1) takes less than 4 ms per iteration in the worst case.

We then run Algorithm 1 for the intersection instance shown in Figure 1, which contains twenty vehicles and forty eight conflict points. Then, we inserted additional vehicles per road (far enough so to ensure that rear-end collsions do not occur) to determine how many vehicles the supervisor can handle within the 100 ms. In Figure 6, the computation time required for one iteration of Algorithm 1 is shown with respect to the number of vehicles. Notice that as the number of vehicles increases, the computation time increases exponentially. Although the problem is not scalable, about twenty five vehicles can be managed by the supervisor within the 100 ms even in the complicated intersection scenario.

The intersection scenario of Figure 1 is created from the top 20 crash intersection locations in the report of the Massachusetts Department of Transportation [24] such that it can represent each intersection topology by removing or com-

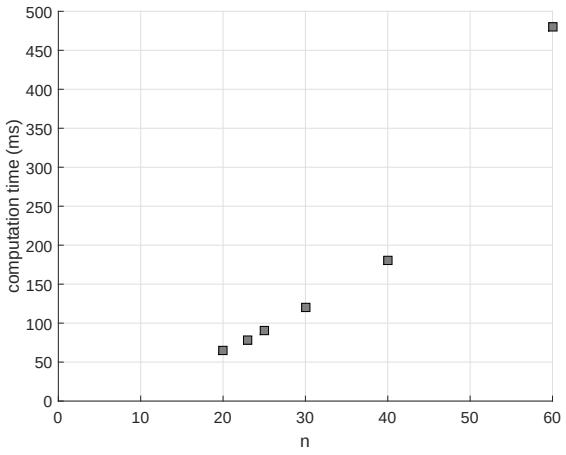


Figure 6: Computation time for one iteration of Algorithm 1 in the worst case with respect to the number of vehicles.

bining its lanes. That is, this intersection scenario consisting of twenty lanes and forty eight conflict points is more complicated than the twenty most dangerous intersections in Massachusetts. If we do not consider rear-end collisions and assume that there is only one vehicle per road, the number of vehicles in typical intersection scenarios usually does not exceed twenty. We can thus conclude that this supervisor is practical for typical intersection scenarios. How accounting for rear-end collisions affects computational complexity will be investigated in future work. It is shown in [11] that additional vehicles on the same lane increase computational complexity less than those on different lanes due to precedence constraints. Since in Figure 6, we did not consider these precedence constraints, we expect that the computation time will be lower than that shown in Figure 6.

6. CONCLUSIONS

We have designed a supervisor that overrides human-driven vehicles only when a future collision is detected and has a non-blocking property. To this end, we have formulated the verification problem and the job-shop scheduling problem and proved that they are equivalent. To solve the job-shop scheduling problem, we have converted it into a mixed-integer linear programming problem by assuming the single integrator vehicle dynamics. The computer simulations confirm that the supervisor guarantees safety while overriding vehicles only when a future collision is unavoidable otherwise. Also, the computational studies show that despite the combinatorial complexity of the verification problem, the supervisor can deal with a complicated intersection scenario as in Figure 1 within the allotted 100 ms per iteration.

While this paper considers a general intersection model in terms of conflict areas, the inclusion of rear-end collisions in the scenario makes it more practical. Moreover, to account for more realistic dynamic behaviors of vehicles, a nonlinear second-order model will be considered. In particular, for second-order linear dynamics, the job-shop scheduling problem may be reformulated as a mixed-integer quadratic programming problem. Also, as considered in [5, 1, 2] in which an intersection is modeled as a single conflict area, measurement and process uncertainty and the presence of unequipped vehicles will be investigated in future work. Undetermined routes of vehicles will also be investigated by considering possible decisions of steering inputs.

7. ACKNOWLEDGMENTS

The authors would like to thank Alessandro Colombo and Gabriel Campos at Politecnico di Milano for the helpful discussions and suggestions. This work was in part supported by NSF CPS Award No. 1239182.

8. REFERENCES

[1] H. Ahn, A. Colombo, and D. Del Vecchio. Supervisory control for intersection collision avoidance in the presence of uncontrolled vehicles. In *American Control Conference (ACC)*, June 2014.

[2] H. Ahn, A. Rizzi, A. Colombo, and D. Del Vecchio. Experimental testing of a semi-autonomous multi-vehicle collision avoidance algorithm at an intersection testbed. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[3] E. Balas, G. Lancia, P. Serafini, and A. Vazacopoulos. Job shop scheduling with deadlines. *Journal of Combinatorial Optimization*, 1(4):329–353, Dec. 1998.

[4] F. Borrelli, D. Subramanian, A. Raghunathan, and L. Biegler. MILP and NLP techniques for centralized trajectory planning of multiple unmanned air vehicles. In *American Control Conference (ACC)*, June 2006.

[5] L. Bruni, A. Colombo, and D. Del Vecchio. Robust multi-agent collision avoidance through scheduling. In *IEEE Conference on Decision and Control (CDC)*, Dec. 2013.

[6] M. R. Bussieck and S. Vigerske. MINLP Solver Software. In *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2010.

[7] G. Campos, P. Falcone, H. Wymeersch, R. Hult, and J. Sjoberg. Cooperative receding horizon conflict resolution at traffic intersections. In *IEEE Conference on Decision and Control (CDC)*, Dec. 2014.

[8] L. Chen and C. Englund. Cooperative Intersection Management: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2015.

[9] M. Christodoulou and S. Kodaxakis. Automatic commercial aircraft-collision avoidance in free flight: the three-dimensional problem. *IEEE Transactions on Intelligent Transportation Systems*, 7(2):242–249, 2006.

[10] A. Colombo and D. Del Vecchio. Efficient algorithms for collision avoidance at intersections. In *Hybrid Systems: Computation and Control (HSCC)*, 2012.

[11] A. Colombo and D. Del Vecchio. Least restrictive supervisors for intersection collision avoidance: A scheduling approach. *IEEE Transactions on Automatic Control*, 2014.

[12] T. H. Cormen. *Introduction to algorithms*. MIT Press, 2009.

[13] M. Earl and R. D'Andrea. Modeling and control of a multi-agent system using mixed integer linear programming. In *IEEE Conference on Decision and Control (CDC)*, Dec. 2002.

[14] C. A. Floudas. *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press, 1995.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco, 1979.

[16] I. E. Grossmann and J. P. Ruiz. Generalized disjunctive programming: A framework for formulation and alternative algorithms for MINLP optimization. In *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 93–115. Springer New York, 2012.

[17] E. I. Grotli and T. A. Johansen. Path planning for UAVs under communication constraints using SPLAT! and MILP. *Journal of Intelligent and Robotic Systems*, 65(1-4):265–282, 2011.

[18] Gurobi Optimization, Inc. Gurobi optimizer reference manual. `http://www.gurobi.com/documentation/6.0/refman/`, 2014.

[19] M. Hafner, D. Cunningham, L. Caminiti, and D. Del Vecchio. Cooperative Collision Avoidance at Intersections: Algorithms and Experiments. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1162–1175, 2013.

[20] Intellient Tranportation Systems, Joint Program Office, U.S. Department of Transportation. ITS Strategic plan 2015-2019. `http://www.its.dot.gov/strategicplan.pdf`, 2014.

[21] International Business Machines Corporation. IBM ILOG CPLEX V12.1: Users manual for CPLEX. `ftp://public.dhe.ibm.com/software/websphere/ilog/docs/optimization/cplex/ps_usrmancplex.pdf`, 2009.

[22] H. Kowshik, D. Caveney, and P. Kumar. Provable systemwide safety in intelligent intersections. *IEEE Transactions on Vehicular Technology*, 60(3):804–818, 2011.

[23] J. Lee and B. Park. Development and Evaluation of a Cooperative Vehicle Intersection Control Algorithm Under the Connected Vehicles Environment. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):81–90, 2012.

[24] Massachusetts Department of Transportation. 2012 Top Crash Locations Report. `https://www.massdot.state.ma.us/Portals/8/docs/traffic/CrashData/12TopCrashLocationsRpt.pdf`, 2014.

[25] National Highway Traffic Safety Administration, U.S. Department of Transportation. 2012 Motor Vehicle Crashes: Overview, Traffic Safety Facts. `http://www-nrd.nhtsa.dot.gov/Pubs/811856.pdf`, 2013.

[26] M. Pinedo. *Scheduling theory, algorithms, and systems*. Springer, 4th edition, 2012.

[27] A. Richards, T. Schouwenaars, J. P. How, and E. Feron. Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. *Journal of Guidance, Control, and Dynamics*, 25(4):755–764, 2002.

[28] The MathWorks, Inc. MATLAB Coder™User's Guide. `http://de.mathworks.com/help/pdf_doc/coder/coder_ug.pdf`, 2015.