Cycle-Removal-Based Priority Policies Coordination for Distributed Intelligent Intersection Management

Kai-En Lin[1], Wan-Ling Weng[1], Eunsuk Kang[2], Chung-Wei Lin[1]
[1]National Taiwan University, Taipei, Taiwan, [2]Carnegie Mellon University, Pittsburgh, PA, USA
r11922065@ntu.edu.tw, b10902060@ntu.edu.tw, eunsukk@andrew.cmu.edu, cwlin@csie.ntu.edu.tw

***Abstract*—Priority policies, which schedule vehicles' passing order in non-signalized intersection management systems, have been a critical task since the development of connected and autonomous vehicles. However, current priority policies lack the ability to coordinate with each other in the same intersection, leaving little flexibility to handle the complex real-world scenarios such as unexpected emergencies, special requirements from manufacturers or service providers, non-uniform computing environments, and faulty control. In this work, a cycle-removal-based coordination strategy is proposed to resolve the deadlock created by the incompatibility between different priority policies. This strategy involves solving an optimization problem—a constrained version of the minimum feedback arc set problem. This newly proposed problem builds upon established algorithmic foundations for the unconstrained version but requires significant adaptations to meet the unique constraints and demands of practical distributed intersection management. Particularly, an exact method and a heuristic are studied and adapted with numerical experiments designed to evaluate their performance. The experimental results show their effectiveness in achieving our goals, which enables the coordination of priority policies and improves an intersection management system's capability for handling complicated requirements.**

***Index Terms*—Cycle Removal, Intelligent Intersection Management, Priority Policies Coordination.**

I. Introduction

The development of Connected and Autonomous Vehicles (CAVs) facilitates the study of non-signalized intersection management, which releases the controlling mechanisms from traditional traffic lights or stop signs and thus enables more efficient intersection management solutions [1]. Existing approaches for CAVs can be categorized into centralized and distributed ones [1]. In centralized approaches, infrastructure is present for gathering vehicles' information and resolving the conflicts between them. Whereas in distributed approaches, vehicles communicate with each other to form a feasible plan to cross the intersection. Although this eliminates the need for infrastructure support, it comes with the cost of synchronizing states among multiple entities, which adds to system complexity and communication overhead.

A. Related Work

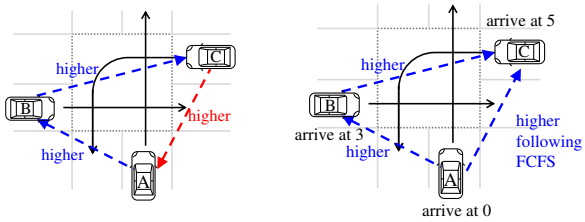
As an intersection is a place where vehicles' trajectories heavily intersect with each other, conflict resolution is an essential part of most intersection control methods. When two vehicles want to go through the same area simultaneously, they must agree on a passing order to avoid collision or circular waiting. In earlier works, simple priority policies like First-Come, First-Served (FCFS) were often used since the main objective was to demonstrate the potential to outperform traditional signal-based methods, such as traffic lights. In the work of Naumann *et al.*, each vehicle computes a priority score based on several features such as idling time and velocity [2]. In the work of VanMiddlesworth *et al.*, the passing orders between vehicles are determined by their arrival lanes, predicted exit times, and whether they have stopped [3]. In the work of Azimi *et al.*, the design of priority policy is less emphasized and FCFS is just used for demonstration [4].

Later, more sophisticated priority policies were proposed for further optimizing the overall system performance. In the work of Liu *et al.*, the passing order decisions are iteratively updated given the broadcasted motion planning results from all the vehicles [5]. The process is proved to converge in finite time and reach a local-optimal solution. In the work of Mirheli *et al.*, the trajectory planning problem is formulated as a distributed mix-integer non-linear program [6]. Each vehicle iteratively solves its optimization task until all the vehicles' solutions converge. This algorithm directly outputs an acceleration profile for each vehicle, but the passing orders between vehicles are still implicitly decided. In the work of Wu *et al.*, decentralized machine learning is adopted, where the movement of vehicles is modeled as a multi-agent Markov decision process [7]. In states where coordination is needed (*i.e.*, vehicles have conflicts), a joint action specifying which vehicle can proceed is selected by the trained model.

Recent research has also integrated decentralized machine learning frameworks into autonomous intersection management. For instance, multi-agent systems incorporating trust mechanisms have been explored to quantify the reliability and trustworthiness of connected vehicles, thus improving decision-making robustness and resilience against misbehavior or faults. Specifically, Zhao *et al.* proposed a trust-aware control approach for intelligent transportation systems, enhancing vehicle cooperation through trust evaluation [8]. Similarly, Fang *et al.* introduced a general trust framework applicable broadly across multi-agent scenarios [9], and Li *et al.* demonstrated trust-aware resilient control specifically for connected automated vehicles at intersections [10]. While these trust-based frameworks show promise, they generally require

This work is partially supported by Ministry of Education (MOE) in Taiwan under Grant Number NTU-113V2003-2 and National Science and Technology Council (NSTC) in Taiwan under Grant Number NSTC-112-2221-E-002-168-MY3.

2325-1301/25/$31.00 ©2025 IEEE
DOI 10.1109/RTCSA66114.2025.00015



(a) (b)

Fig. 1: (a) Vehicles A and C deciding on the passing order, where "Higher" means that vehicle C goes before vehicle A, cause a deadlock co-formed by the passing orders between vehicles A and B as well as vehicles B and C. When the "higher" relations form a cycle, the situation becomes logically infeasible and thus forms a deadlock. (b) FCFS decides on passing orders based on arrival time (annotated beside each vehicle), whose comparison is totally ordered by nature. Thus, vehicles A and C do not need to know the passing orders between other vehicles to avoid a deadlock.

extensive training, complex learning procedures, or explicit trust models, making real-time deployment challenging. More importantly, learning-based approaches do not provide formal guarantees of deadlock-free operation, as their decision-making relies on probabilistic models and adaptive learning. In contrast, our approach explicitly guarantees deadlock-free solutions through deterministic priority adjustments, eliminating the need for complex training and ensuring computational efficiency for real-world deployment.

B. Deadlock-Freedom and Motivating Example

An essential property of a priority policy is the deadlock-freedom of the decided passing orders. Without this property, the resulting cyclic dependencies may prevent the vehicles from forming a valid ordering and thus impossible to execute (*i.e.*, a deadlock). As a result, the passing order between two vehicles cannot be decided without considering the passing orders between the other ones. For example, in Figure 1(a), vehicles A and C decide on a passing order that creates a cycle. However, to be aware of this problem, they need to check the passing order between vehicles A and B as well as vehicles B and C. This issue is especially difficult to handle in distributed approaches, where the decision-making may occur asynchronously in different devices and the sharing of information may be delayed. Every distributed priority policy has its strategy to achieve deadlock-freedom. For example, FCFS relies on the totally-ordered nature of arrival times as Figure 1(b) shows. On the other hand, the priority policy in the work of Liu *et al.* involves identifying and resolving a special "tie" relation that may be formed across multiple vehicles [5]. As the strategies used by different policies are often apparently incompatible, this implicitly requires every vehicle to follow the same priority policy, leaving little flexibility to adjust the passing orders between vehicles. The lack of flexibility is undesired considering the complex and dynamic nature of the real world. Below we give some example scenarios where we may need multiple priority policies to operate together.

A possible scenario is that some vehicles want to yield those with reasonable emergencies. While forcefully yielding all the emergency vehicles (*e.g.*, ambulances, fire trucks) has been implemented in existing works [11], some normal vehicles might also be worth yielding based on the other driver's judgment (*e.g.*, carrying a kid with a fever or a severely injured person). Another possible scenario is when some vehicles do not join the process of forming a deadlock-free passing order in the first place for some reason. For example, some manufacturers or service providers may design a proprietary priority policy used only between their owned vehicles. On the other hand, the selected policy could be too resource-intensive so some vehicles have insufficient computing power to execute it considering the non-uniformity of the computing environment among vehicle models. There could also be cases where the scheduled passing order is accidentally violated due to the inherent uncertainty in the real world like faulty control, faulty communication, or human interference [12]. Overall, with the ability to dynamically adjust the passing orders between vehicles based on different policies, we can satisfy more complicated requirements and improve the fault recovery of the system.

C. Contributions

Thus, in this paper, we propose to provide a general strategy for ensuring deadlock-freedom and enabling the coordination between multiple priority policies for a distributed intersection management system. Specifically, for any pair of vehicles, they can decide their passing order by whatever means they want. To achieve this, several challenges need to be solved. First, as a deadlock indicates the logical infeasibility of the passing orders, it is inevitable to modify some of the decisions made by the priority policies for maintaining deadlock-freedom. However, since this defies our purpose—letting the vehicles decide which policy to use, the number of modified decisions needs to be as small as possible. Ideally, the modification should be negligible so that the goals of all the policies (*e.g.*, yielding a vehicle, reducing traveling time) can still be achieved to some extent. Second, the passing orders are often not free to be modified due to physical limits or traffic regulations. For example, a narrow lane could prohibit vehicles from overtaking and laws could enforce yielding to emergency vehicles. As a result, there should be a mechanism to ensure that these priorities are not subject to modification in the deadlock resolution process. Third, the aforementioned challenges need to be solved in real time. This drives us to be aware of the time complexity and execution-time boundability of the adopted algorithms.

To solve these challenges, we propose a priority policy coordination strategy to resolve a possible deadlock after the vehicles decide the passing orders between them with whatever policy they prefer. The deadlock resolution process is abstracted as a constrained version of the NP-hard *minimum*

feedback arc set problem (FAS). This newly proposed but fundamental algorithmic problem meets the unique constraints and demands of practical distributed intersection management. We demonstrate that several exact methods and heuristics for FAS are suitable for solving the proposed problem with necessary modifications. After numerically examining these algorithms, we find that it is feasible to solve the problem optimally when the number of vehicles is low despite its NP-hardness. Additionally, the performance of the heuristics is generally acceptable compared with the optimal solution, which makes them useful in resource-limited environments.

D. Organization

The remaining paper is organized as follows. The deadlock resolution problem is formulated as an algorithmic optimization problem with proper assumptions in Section II. The adopted algorithms for solving this optimization problem, including exact methods and heuristics, are described in Section III. In Section IV, we design several numerical experiments and present the results to show the effectiveness of the proposed algorithms. In Section V, limitations and future directions of this work are discussed. Section VI concludes the paper.

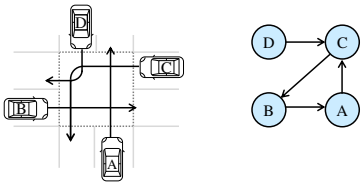
II. Problem Formulation

In this section, we first describe the fundamental assumptions that clarify the scope and usage of the proposed strategy in Section II-A. Then, the abstracted algorithmic problem definition and the reasoning behind it are described in Section II-B.

A. Assumptions

1) System Support: The proposed strategy targets only the process of deciding the passing orders between vehicles in a distributed intersection management system. The other aspects of the system (*e.g.*, conflict detection, path planning, motion planning) are not in our scope. We assume that the proposed strategy operates after the passing order is decided for each pair of conflicting vehicles (*i.e.*, the vehicle that should go first is clear and the same for both of them). This decision may be made by either a default priority policy, a dynamically established consensus (*e.g.*, a proposal from one vehicle is agreed by another), or enforcement of regulations. Note that the process of making the decision is unrestricted. For example, multiple vehicles may cooperate to achieve better results. The settlement of the passing orders may be signaled by an additional field in the periodically broadcasted message to determine when to initiate the proposed strategy.

2) Communication Capability: We assume that the communication capability allows all the vehicles to broadcast the decided passing orders associated with themselves so that every vehicle has complete information for computing the result of deadlock resolution. The passing order information may be encoded as an array of boolean variables whose length is the number of conflicting vehicles. Although this makes the size of the field grow with the number of vehicles in the system, it will not cause serious overhead considering it is limited by the physical space of an intersection. This assumption may be relaxed by adopting a leader election mechanism for delegating information collection and computation to a single vehicle.



(a) (b)

Fig. 2: (a) A set of vehicles with their trajectories illustrated in black arrows. (b) A corresponding priority graph, where the vertices represent the vehicles and the edges indicate the passing orders. There is no edge between B and D (A and D) since their trajectories do not intersect, so there is no need for them to decide on a passing order.

3) Cooperative Behavior: We assume that vehicles behave cooperatively and normally. However, realistic deployments inevitably include scenarios where vehicles misbehave, fail to follow assigned priorities, or break down within intersections. While explicitly modeling these situations is beyond the primary scope of this paper, our proposed approach inherently allows adaptation by dynamically incorporating these incidents as fixed constraints within the priority graph. For instance, if a vehicle breaks down, it can be treated as an obstacle, generating immutable constraints (static edges) within the priority graph. Other vehicles can temporarily utilize their onboard sensors and local decision-making capabilities to establish interim passing orders until the priority graph is dynamically recalculated. Robustly detecting and responding to misbehavior, such as through anomaly detection algorithms or periodic heartbeat signals, merit further investigation.

B. Abstraction

1) Priority Graph: It is natural to represent the relative order over the set of vehicles as a directed graph, which will be referred to as a priority graph henceforth. Let $G = (V, E)$ denote a priority graph. There is a vertex in V for each vehicle in the system and $(u, v) \in E$ if and only if the vehicle corresponding to u has priority over the one corresponding to v. An example of vehicles and the corresponding priority graph are shown in Figure 2. Note that there is not necessarily an edge between each pair of vertices as two vehicles' trajectories may not be in conflict with each other (*e.g.*, vehicles B and D in Figure 2(b)). It is straightforward that any cycle in a priority graph corresponds to a logical deadlock, which cannot be eliminated without modifying the graph. Since an edge indicates a conflict that must be resolved, the edges cannot be simply removed. Instead, we can only *reverse* the edges to achieve our goal.

2) Constraint: When we enumerate the challenges to be solved in Section I, we urge the need to ensure that some

passing orders are not modified in the deadlock resolution process. This requirement is for modeling government regulations or physical limitations to avoid chaotic traffic conditions or impossible schedules. Some examples of regulations are the forbiddance of overtaking and the enforcement of yielding to emergency vehicles. For physical limitations, an inapparent example is to satisfy fait accompli. That is, if a vehicle has already entered the conflicting area, then the other vehicles automatically yield to that vehicle assuming rollback is never worth it. This is essential to handle fault recovery scenarios where some vehicles physically violate the schedule. Therefore, we conclude that constraint is necessary for the proposed strategy. The constraint is represented as a subset F of the edge E in the priority graph, which contains the edges that cannot be reversed. The construction of this set on the computing device may be implemented by hard-coded rules or authenticated requests sent by special vehicles. Note that the subgraph $G = (V, F)$ must not contain any cycle, or the deadlock will be logically unresolvable. Therefore, the design of the constraint should be done with caution, just as designing a priority policy.

C. Problem Statements

Combining both aforementioned elements, we give the formal statement of the resulting algorithmic abstraction, which is a constrained cycle removal problem on a directed graph (*i.e.*, priority graph). We first define a *swap arc set* [13] on a directed graph.

Definition 1 (Swap Arc Set)**.** *Given a directed graph, a swap arc set of the graph is a subset of the edges whose reversal makes the graph acyclic.*

By definition, we can resolve a deadlock by modifying the passing orders corresponding to a swap arc set of the priority graph. A swap arc set is first defined in the work of Oliva *et al.* [13]. It can also be called a *feedback set* in the field of graph drawing [14]. Since the first naming reflects the definition better, we choose to use the naming from the work of Oliva *et al.* [13] despite the fact that it was published later. Then, the problem statement is defined as follows.

Problem 1 (Constrained Minimum Swap Arc Set Problem)**.** *Given a directed graph $G = (V, E)$ and a subset of the edges $F \subseteq E$ where $G_F = (V, F)$ is acyclic, find the minimum swap arc set of G that does not contain any edges in F.*

Problem 1 is the constrained version of the *minimum swap arc set problem* studied in the work of Oliva *et al.* [13]. As the unconstrained minimum swap src set problem has been proved to be NP-hard in the work, it is trivial to prove that this problem is also NP-hard by reduction (let $F = \emptyset$).

III. Algorithms

The unconstrained version of Problem 1 has been studied by Oliva *et al.* [13], who proposed a distributed cycle removal algorithm that is efficient in communication cost. However, their algorithm cannot be easily adapted to deal with the constraint we introduce because it involves reversing all the incoming edges of a vertex. After this work, there is still no further research on this problem.

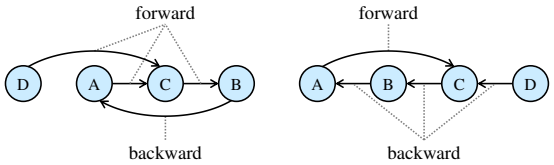


Fig. 3: These two figures are two possible orderings of the priority graph in Figure 2(b) with the forward and backward edges marked. Although both orderings create feasible solutions to the minimum feedback arc set problem, (a) is better since it induces fewer backward edges than (b).

There is another problem that is less similar to our problem but much more studied, called the *minimum feedback arc set problem*. A *feedback arc set* of a graph is a subset of the edges whose *removal* makes the graph acyclic. In the work of Tamassia *et al.*, it has been proved that a minimal feedback arc set is also a swap arc set of the same graph [14]. As a result, most of the algorithms for the minimum feedback arc set problem can also be directly used to solve the swap arc set variant. However, there is another gap to our problem: the constraint. Compared with the original problem, the constrained minimum feedback arc set problem is much less studied in the literature. The only work we can find is an approximation algorithm that specializes in tournaments [15], a type of graph where there is exactly one edge between each pair of vertices. Thus, additional effort is still needed to examine and adapt existing solutions.

It is worth noting that there is an equivalent problem of the minimum feedback arc set problem called the *linear ordering problem* [16], which aims to rank a set of elements such that the total weights from higher-ranked elements to the lower-ranked elements are maximized. This equivalence comes from a simple observation that an acyclic graph, hence a feedback arc set, always corresponds to an ordering of the vertices. To construct a feedback arc set from an ordering, one just needs to find the "backward" edges, which are those edges pointing from lower-ranked vertices to higher-ranked vertices (see Figure 3). A lot of methods approach the minimum feedback arc set problem from this perspective. That is, instead of directly finding a good feedback arc set, they attempt to find a good ordering of the vertices and construct a feedback arc set from it.

A. Exact Methods

1) Overview: In this section, the possibility of using exact methods for the minimum feedback arc set problem to solve Problem 1 is investigated. Considering its NP-hardness, these methods are unlikely to be scalable. However, since the number of vehicles that want to cross an intersection is limited

by the physical space, the problem instance can be mostly small and tractable.

It should be noted that the purpose of this section is not to pursue the state-of-the-art performance but to provide tools to demonstrate the effectiveness of the proposed formulation in solving the challenges introduced in Section I. As a result, we only explore the latest exact method based on *integer linear programming (ILP)*, which has a relatively simple structure as most of the complexity is delegated to the ILP solvers. On the other hand, another broad category of exact methods, branch and bound, utilize various techniques to efficiently enumerate the set of feasible solutions [16], [17], [18], [19]. This kind of method has the potential to outperform ILP-based methods according to existing benchmarks [20]. However, due to its variety and complexity in technical details, we choose not to study this category in this paper and leave it for future research.

2) Lazy-Cycle-Enumeration-Based Formulation (LCE-ILP): This formulation [21] is based on the reduction from the *minimum set cover problem* to the minimum feedback arc set problem. The concept is to find the minimum set of edges that covers all the cycles in the graph, and the removal of these edges naturally breaks all the cycles. Let binary variable y_i indicate whether edge i is selected in the minimum feedback arc set, *i.e.*, $y_i = 1$ represents the decision to reverse edge i, and the set of such edges defines the swap-arc set $\mathcal{S}$. Suppose there are c simple cycles and m edges, the cycles are encoded in a *cycle matrix* $A_{c\times m} = (a_{ij})_{c\times m}$, where $a_{ij} = 1$ if edge j is in the i-th cycle; otherwise, $a_{ij} = 0$.

The problem with this formulation is the construction of the cycle matrix, which should be obtained by enumerating all the cycles in the graph. However, the number of simple cycles in a graph is $\Omega(2^n)$ [21] even for sparse graphs so it must take at least $\Omega(2^n)$ time complexity to achieve this. As a result, this formulation works only when the number of cycles is small.

To adapt this formulation to solve Problem 1, we just need to fix $y_j = 0$ for all $j \in F$, which is written as follows:

$$\begin{aligned}
&\text{minimize} && \sum_{j\in E\setminus F} y_j, \\
&\text{subject to} \\
&&& \sum_{j\in E\setminus F} a_{ij}y_j \geq 1, && \forall\, i \in \{1,2,\ldots,c\}. \\
&&& y_j \in \{0,1\}, && \forall\, j \in E \setminus F.
\end{aligned}$$

Here, the minimum-modification cost refers to the number of reversed priority relations, *i.e.*, the size of the swap-arc set $\mathcal{S}$. To overcome the intractability of cycle enumeration, Baharev *et al.* [21] proposed a method that lazily enumerates cycles. Their method iteratively solves the minimum set cover formulation with a "partial" cycle matrix. If the solution of this partial formulation does not cover all the cycles, the graph with the edges in this solution removed, denoted by $G^{(k)}$ for the k-th iteration, must contain some cycles. To discover these uncovered cycles, a heuristic algorithm for the minimum feedback arc set problem is run on $G^{(k)}$, which gives a set of edges that cover these cycles. Then, for each edge in this set, breadth-first-search is used to find one cycle that contains this edge. The discovered cycle can thus augment the partial cycle matrix. This process is repeated until the solution of the current partial formulation leads to a valid feedback arc set. A concrete example of how an iteration is executed is illustrated in Figure 4. Note that this method requires the adopted ILP solver to support the feature of incrementally building a linear program so that each run of the solver does not need to start from scratch. In the work of Baharev *et al.*, Gurobi with the *Lazy Constraint* feature is used [21].

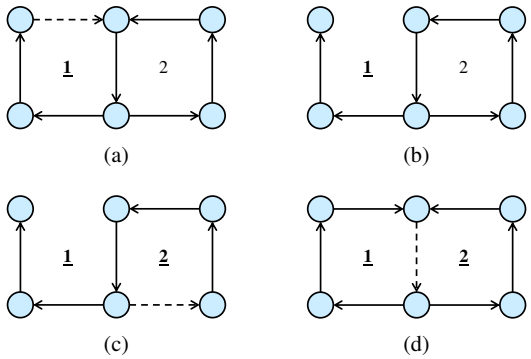
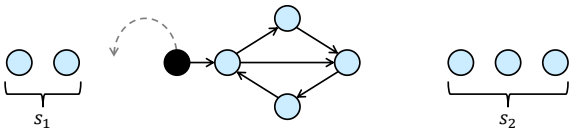


Fig. 4: (a) Suppose only cycle 1 is known at the beginning of the current (k-th) iteration. As a result, the ILP solver returns an edge (the dashed one) that only covers cycle 1. (b) This figure shows the graph $G^{(k)}$, which is still cyclic. (c) A heuristic algorithm is run on $G^{(k)}$ and it finds an edge (the dashed one) that maybe covers some cycles. Leveraging this edge, we can discover cycle 2 using BFS. (d) In the next iteration, both cycles are known. Thus, the ILP solver returns the optimal solution that covers both cycles with the minimum number of edges (one edge).

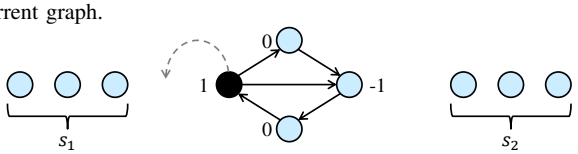
B. Heuristic Algorithms

1) Overview: Although Problem 1 may be tractable most of the time considering the sizes of real-world intersections, we still need to study heuristic algorithms for the following reasons. First, some exact methods (*e.g.*, LCE-ILP described in Section III-A2) use heuristic algorithms as part of their procedures to obtain a temporary feasible solution. Thus, at least a working heuristic is necessary, and a better one may further reduce the running time of those exact methods. Second, exact methods often have a weak running time guarantee due to the NP-hardness of the problem. Considering the time-sensitive nature of this application, it is necessary to have a fast and execution-time-boundable heuristic as a backup in case the exact method cannot meet the real-time constraints.

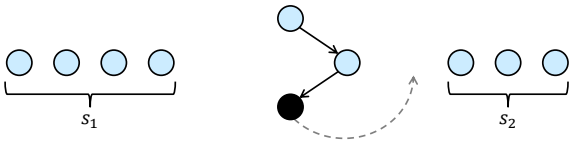
In this paper, a heuristic algorithm named GreedyFAS [22] serves as a representative of the heuristics. This algorithm is selected according to the following process: First, we survey existing algorithms for the minimum feedback arc set problem



(a) The black vertex is a source. When it is appended to s_1, it induces no backward edges with respect to the rest of the vertices in the current graph.



(b) There are no sources or sinks, so the δ function is calculated. The black vertex has the greatest value (1) so it is appended to s_1.



(c) Now there is one source and one sink. Suppose the sink (black vertex) is chosen. When prepended to s_2, it induces no backward edges since it is a sink.

Fig. 5: An illustration of the GreedyFAS algorithm. The black vertex is the one that is selected in each iteration according to the greedy criteria.

and implement only the suitable ones for Problem 1. Then, we conduct experiments on these implementations and select the one with the best overall performance. The basic requirement for a suitable algorithm is the ability to deal with the constraint F introduced in Problem 1 without severely compromising the intuition behind the heuristic. Additionally, we also exclude those heavyweight meta-heuristics like simulated annealing considering that heuristics are meant to be fast according to the reasons mentioned in the previous paragraph. We adapt and implement only two algorithms: GreedyFAS [22] and InsertionSortFAS [23]. However, InsertionSortFAS showed poor performance in our initial experiments, so we focus solely on GreedyFAS in this paper. Non-experimented heuristics are briefly summarized in Section III-B3 as a guide for future research.

The surveyed algorithms are mainly collected by consulting two papers [16], [24] which cover most of the existing algorithms. The first work [24] empirically studies several heuristic algorithms [22], [23], [25] with scalability as the main objective and provides efficient implementations for them. The second work comprehensively studies the linear ordering problem (which is equivalent to the minimum feedback arc set problem as mentioned in the opening of Section III). Despite lacking implementation details, it contains a broader collection of heuristics than [24]. It also conducts a benchmark of the studied algorithms, which is a great reference for comparing

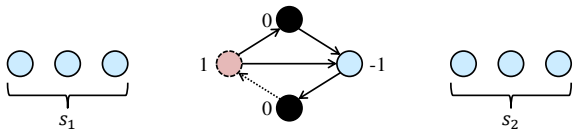


Fig. 6: Suppose there is one edge (marked in a dotted line) in the constraint F in the situation of Figure 5(b). Appending the vertex with the greatest δ value 1 will make this edge backward, violating the constraint. As a result, we should instead select from the two vertices with zero δ value in this iteration.

them. The next section describes how GreedyFAS works and how it can be adapted to solve Problem 1.

2) GreedyFAS: GreedyFAS [22] iteratively constructs a linear ordering of the vertices based on the degrees of each vertex. Let $\delta^-(v)$ denote the indegree of vertex v and $\delta^+(v)$ denotes its outdegree. At the beginning of the algorithm, two empty sequences s_1 and s_2 are initialized. These two sequences will be concatenated as $s_1\, s_2$ to form the final linear ordering after processing all the vertices. In each iteration, it first tries to remove a source (*i.e.*, $\delta^-(v) = 0$) or a sink (*i.e.*, $\delta^+(v) = 0$) from the current graph. The removed vertex is then appended to s_1 if it is a source or prepended to s_2 if it is a sink. If there are no sources or sinks, then the vertex with the greatest $\delta(v) = \delta^+(v) - \delta^-(v)$ in the current graph is removed and appended to s_1. These criteria for selecting vertices are illustrated with concrete examples in Figure 5. The iteration stops when the graph becomes empty, and the concatenated sequence $s_1\, s_2$ is returned as the linear ordering, which corresponds to a feedback arc set as described in the opening of Section III.

The intuition behind this heuristic is to greedily add the vertex that induces no backward edges (*i.e.*, a source or a sink) with respect to the remaining vertices currently. When a vertex is appended to s_1, it means all the remaining vertices will be ranked lower than it. Thus, all the vertex's indegrees contribute to the backward edges. Similarly, a vertex's outdegrees also contribute to the backward edges when prepended to s_2. This is why sinks and sources induce zero backward edges. However, sinks or sources may not exist. In this case, the δ function is used as the greedy criterion. Although the design philosophy of this function remains unexplained in the original work [22]. Simpson *et al.* gave an intuitive interpretation that it is to select the most "source-like" vertex [24].

This intuition also works with the constraint introduced in Problem 1. In each iteration of the algorithm, a vertex inducing no backward edges in F should be selected as illustrated in Figure 6. Since $G_F = (V, F)$ is acyclic by the problem definition, such vertex always exists. This ensures that the final ordering induces no backward edges in F, so the resulting swap arc set and F are disjoint. The disjointness between the swap arc set and F is guaranteed by the algorithm's design: because G_F is acyclic by definition, there always exists a removable vertex that does not induce backward edges over

F, and thus the heuristic never needs to skip or block edges in F in implementation explicitly.

3) Non-Experimented Heuristics: In this section, we list the non-experimented heuristics and the corresponding reasons. The first obvious reason is the incapability of handling the constraint F. For example, some heuristics may rely on removing all the incoming/outgoing edges of a vertex, which cannot be done in the presence of non-changeable edges. The randomized heuristics proposed by Berger & Shor [25], *KwikSort* by Brandenburg [23] *et al.*, and the simple DFS-based approach [24] fall into this category. Another common reason is high or unbounded time complexity, which defies our purpose of studying heuristics (see the opening of Section III-B). The various meta-heuristics like GRASP, tabu search, and simulated annealing described in the work of Marti *et al.* [16] as well as the divide-and-conquer algorithm in the work of Saab *et al.* [26] are not selected due to this reason. Other reasons include obvious inferiority in existing benchmark [16] (Aujac & Masson [27]) and similarity to one of the experimented algorithms (SiftFAS [23]).

IV. Numerical Experiments

From our problem formulation in Section II, it is clear that we can resolve the deadlock and ensure that the non-changeable passing orders are respected. However, it is still unclear if the first and the third challenges are well-solved—that is, the amount of interference imposed by the proposed approach with the priority policies and the ability of the studied algorithms to respond in real time. Thus, we will try to evaluate these items by numerically measuring several metrics with randomly generated traffic data. In each piece of traffic data, two different priority policies are applied to a batch of vehicles that intend to pass the same intersection. Then, the proposed algorithms: LCE-ILP and GreedyFAS are executed on the abstracted problem instance. All the experiments are run on a machine with an Apple M1 CPU and 8 GB memory. For the ILP solver, we use Gurobi [28], which has the lazy constraint feature required by LCE-ILP. For the heuristic required by LCE-ILP, GreedyFAS is used.

A. Intersection Topology and Traffic Generation

The traffic data is generated on a four-way intersection with three lanes for each incoming and outgoing direction as shown in Figure 7. Each piece of data consists of a finite number of vehicles with different attributes such as trajectory and arrival time. For each generated vehicle, its trajectory is randomly selected from the set of legal trajectories depending on its incoming lane as illustrated in the figure. For simplicity in conflict detection, the intersection is uniformly partitioned into 36 disjoint conflict zones, and a trajectory is encoded as a sequence of conflict zones. If two vehicles' trajectories have any common conflict zones, then these two vehicles are regarded as conflicting. It should be noted that the approach to perform conflict detection is not limited by the proposed method and it can even be priority-policy-dependent. In addition to the trajectory, some other information is often required for a priority policy to make decisions. For example, FCFS needs the arrival time of a vehicle. Thus, we also generate all the necessary attributes, including arrival time and speed, with proper random distributions. To demonstrate the algorithms' performance under different sizes of problem instances, the possible numbers of vehicles in each piece of data are only ten's multiples ranging from 10 to 60 and there are 50 pieces of data for each number. Note that the setting of up to 60 vehicles is based on the communication range as well as the reasonable number of vehicles near the intersection, and it does not represent a scalability limit.

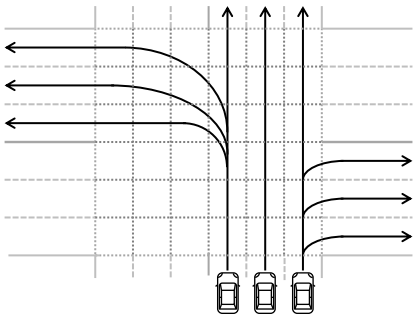


Fig. 7: The four-way intersection with three incoming and outgoing lanes for each direction was used in the numerical experiments. Black arrows mark the legal trajectories for the northbound lanes. The trajectories for the other lanes can be inferred from symmetry.

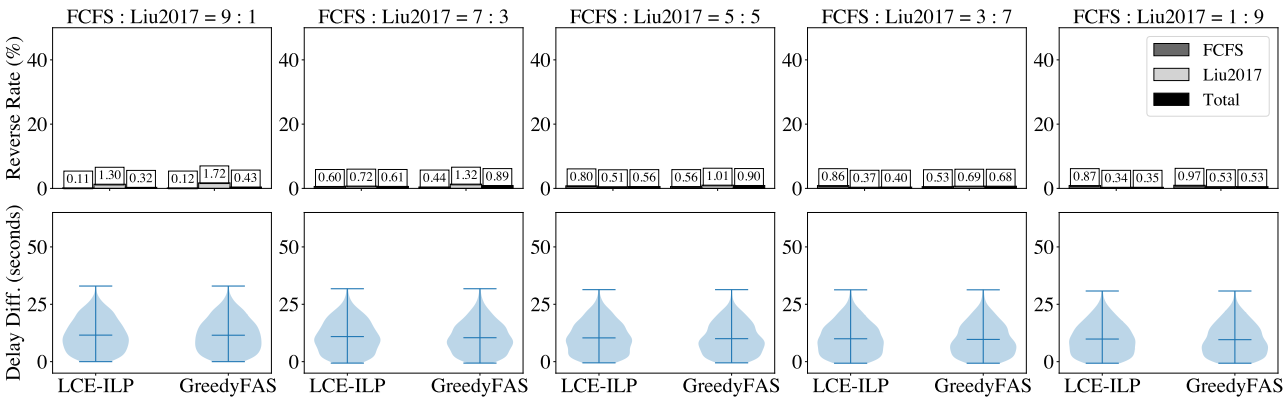
We also assume that the width of the lanes prevents vehicles from overtaking. As a result, within the same incoming lane, the priority of vehicles with earlier arrival time over those arriving later is enforced by the constraint F in Problem 1.

B. Scenarios

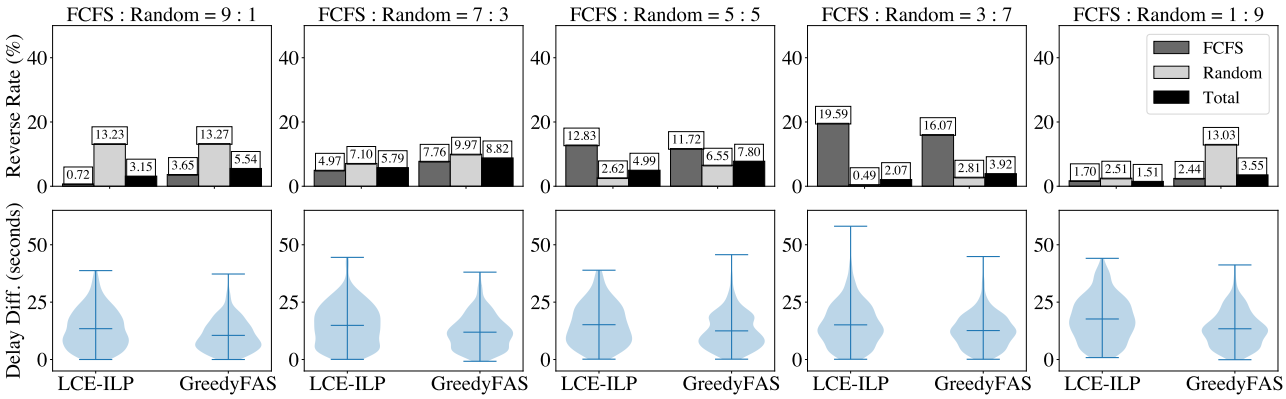
To create passing orders where the proposed strategy is required to resolve a deadlock, we simulate two of the sample scenarios mentioned in Section I:

1) Scenario 1: The first scenario is when a random portion of vehicles uses a priority policy different from the default one. This different priority policy is used only among this portion of vehicles, while the default one is still used between the two portions of vehicles. This scenario may happen when vehicles have different intentions to be fulfilled, incompatible computing environments, or nonidentical sets of available information. The selected default priority policy is FCFS, which is intuitive and common in existing literature. For the other policy, we experiment on two different ones to compare the results when their "similarity" to the default policy varies. The first one is from Liu's work[1] (denoted by Liu2017 henceforth) [5], which represents the case where the intentions of the two policies are more aligned. That is, they both intend to improve the

[1]Due to the lack of specification of some scenarios encountered in our experiments, the implemented priority policy is the extended version of the priority policy from [5]. We adopt a similar tie-breaking methodology to handle the cases where at least one vehicle of a conflicting vehicle pair from different lanes is in the state IL. For the details, please see the Appendix.



(a) FCFS with Liu2017



(b) FCFS with random ordering

Fig. 8: The results of the experiments in Scenario 1. All the metrics are obtained by averaging across all the traffic data that matches the corresponding settings. The lower parts of both figures are violin plots, where the width of the curve shows the frequency of at different values. The three vertical bars of each plot indicate the maximum, mean, and minimum.

overall delay time. The second one is a hypothetical policy where the priority is decided according to a random ordering of the vehicles. This represents a policy whose intention is orthogonal to FCFS, which is designed to demonstrate how the proposed algorithms perform under extreme circumstances.

2) Scenario 2: The second scenario is when one vehicle requests the other vehicles to yield to itself, but only a random portion of vehicles accepts this request. Due to the presence of the no-overtaking constraint mentioned previously, those vehicles that arrive earlier than the requesting vehicle and come from the same lane are also yielded. This scenario may happen when a vehicle has a reasonable emergency like going to a hospital. Whether the reason is acceptable can be up to the other drivers' judgment or decided by other systems (*e.g.*, payment). To reflect the effect of yielding better, the requesting vehicle is always the latest arriving vehicle in the experiments. For the default priority policy, FCFS is selected.

C. Metrics

1) Reverse Rate: The reverse rate is defined as the percentage of the reversible edges reversed by an algorithm after deadlock resolution. This shows how much the priority policies' original decision is forced to be changed and thus reflects the effectiveness of the algorithm. If most of the decisions are changed, then it will be meaningless to let the priority policies make decisions in the first place. In addition to the total reverse rate that considers all the reversible edges, we also report the *separate reverse rates* that consider only those edges whose orientations are decided by each individual priority policy. This tells us if some policies are "sacrificed" too much in the deadlock resolution process. If a policy's separate reverse rate is almost 100%, then it will also be meaningless to allow this policy to operate even if the total reverse rate is low enough. However, it is also natural that policies used by more vehicles should be respected more. Thus, a higher reverse rate for less-used policies is still reasonable.

2) Policy-Specific Metrics: It may be unreasonable for a policy to fully achieve its objective when it is a minority of the decision-makers. However, if a policy can control most of the decisions, then it may be better if its objective can be roughly achieved. Our objective in the process of deadlock resolution, minimization of reverse rate, only reduces the number of

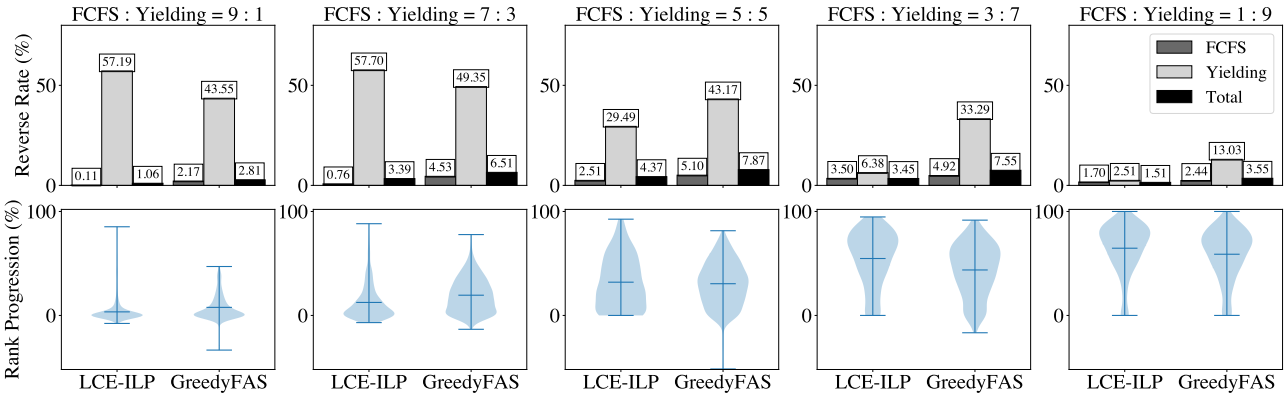


Fig. 9: The results of the experiments in Scenario 2. The upper part of the figure shows the average reverse rate, while the right part shows the rank progression. The rank progression is demonstrated in violin plots, where the width of the curve shows the frequency of at different values. The three vertical bars of each plot indicate the maximum, mean, and minimum.

changed decisions, but it does not necessarily correlate to the change in the objectives pursued by each individual policy. Thus, for each scenario, we design a special metric trying to get an empirical view of this correlation, which may end up revealing a limitation, or a surprising advantage of our method.

a) Scenario 1: Delay Time Difference: The delay time difference is obtained by subtracting the average delay time when all vehicles use FCFS from the average delay time after deadlock resolution. Since Liu2017 is an efficiency-oriented priority policy, we would like to measure the possible improvement in the delay time when more vehicles adopt Liu2017. On the other hand, the random-ordering-based policy may damage the efficiency of the intersection, thus increasing the delay time. To obtain delay time from passing orders, we apply a simple speed profile planning algorithm—each vehicle maintains the maximum acceleration of 2.5 m/s until it reaches the reference speed of 5 m/s. If a vehicle with higher priority blocks its path, it immediately stops. That is, vehicles are assumed to have unbounded deceleration, which is reasonable given the low reference speed.

b) Scenario 2: Rank Progression: This metric is designed to more intuitively demonstrate the effect of yielding. The rank progression is the amount of increase in the yielded vehicle's rank, which is defined as follows:

$$\text{rank progression} = \frac{\text{original rank} - \text{after rank}}{\text{original rank}} \times 100\%.$$

The rank of a vehicle is defined as the number of vehicles that can pass the intersection before it. Thus, the highest possible rank is zero. The original rank means the rank of the vehicle when all the passing orders are decided by FCFS.

D. Results

1) Scenario 1: The results for Scenario 1 are shown in Figure 8. As described in Section IV-B1, this scenario includes two combinations of priority policies: FCFS with Liu2017 and FCFS with the hypothetical policy based on random ordering. For each combination, we vary the ratio of the FCFS portion to another portion in terms of the number of vehicles in order to show the difference as more vehicles select a priority policy.

First, we observe only the total reverse rate. It should be noted that LCE-ILP is an exact method that always outputs optimal solutions. The optimal total reverse rate is generally low ($\leq$1% for FCFS with Liu2017, $\leq$6% for FCFS with random ordering), which indicates the basic effectiveness of the proposed method. The better heuristic (GreedyFAS) is 30-100% worse than the exact methods but still preserves most of the decisions with a total reverse rate lower than 10% in the worst case. The lower total reverse rate of the first combination also shows that a deadlock becomes easier to resolve when the intentions of the used policies are more aligned. Nevertheless, the nonzero optimal reverse rate means that deadlocks still exist, hence making the proposed strategy necessary. The separate reverse rate is generally higher when the corresponding policy accounts for fewer decisions. However, even in extreme cases, the reverse rates of the minority policies are still under 2% for FCFS with Liu2017 and under 20% for FCFS with random ordering using either exact methods or heuristics. This indicates that both policies are respected enough and thus gain fair shares of control in the intersection, which again demonstrates the effectiveness of the proposed method.

As for the delay time difference, we can see that the delay time after deadlock resolution is worse than plain FCFS in almost all cases. It can also be observed that the differences are similar for all ratios, which may indicate that the problem is in the proposed method but not in the policies themselves.

2) Scenario 2: The results for Scenario 2 are shown in Figure 9. For the reverse rate, the trend for either kind of algorithm is the same so we will discuss them together. The total reverse rate is slightly higher than the FCFS with random ordering in Scenario 1 but still lower than 10% in the worst case. The separate reverse rate still becomes higher when a policy gains control over less vehicles, which is the same as Scenario 1. However, different from Scenario 1, the separate reverse rates of the "yielding" policy are always higher than

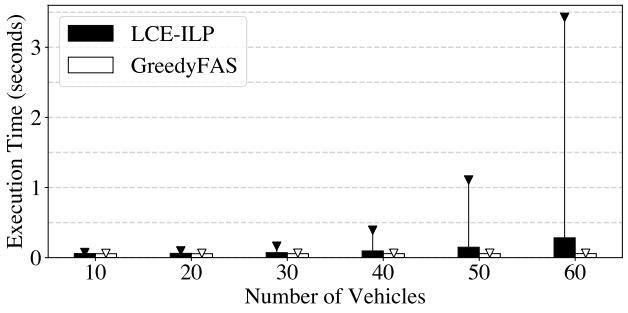


Fig. 10: The execution time of the proposed algorithms when handling different numbers of vehicles. The bars show the means, while the triangles over the bars indicate the maximums.

those of FCFS even when most of the vehicles choose to yield. The reason could be that the yielding policy only takes effect on those decisions that are related to that one yielded vehicle and the vehicles ahead of it. Thus, even if all vehicles choose to yield, there will still be a good portion of passing orders decided by FCFS. For the rank progression, it is apparent that the yielded vehicle's rank advances more on average as a larger portion of vehicles yield it. This shows that the proposed strategy does a better job of reflecting the overall intention of the vehicles. However, we can still observe the extreme values that diverge a lot from the means (sometimes the yielded vehicle's rank even increases), which means that this is definitely not a guarantee.

3) Execution Time: Figure 10 shows the execution time statistics of all the aforementioned experiments. The mean and maximum values are organized in terms of different numbers of vehicles in the generated traffic to illustrate how well each algorithm scales. Looking at the mean values, it is clear that the heuristic (*i.e.*, GreedyFAS) is always the faster one. As expected, the differences between the exact method and the heuristic are tiny with a small number of vehicles but increasing quickly as the number grows. When there are 60 vehicles, LCE-ILP runs for approximately 0.3 seconds on average, which is hardly acceptable considering that our machine is a relatively powerful one compared to common automotive chips. Moreover, the maximum values of LCE-ILP are even far more than the means in some cases, which proves our consideration that exact methods alone are not reliable enough to be used in time-sensitive applications. In contrast, GreedyFAS stably terminates in 0.1 seconds with maximum values approximately equal to the mean values, which makes them good for operating under real-time constraints. The selection between LCE-ILP and GreedyFAS should depend on the CPU frequency and the number of vehicles.

V. Discussions

Although the proposed method effectively solves the challenges enumerated in Section I, it does come with some limitations and leaves some directions for future research. First, it introduces communication and computation overhead to the intersection management system. For communication, each vehicle needs to gather passing order information that may not be related to itself as described in Section II-A2. This adds an additional requirement that all the participating vehicles should be within each other's communication range, or more complicated mechanisms need to be introduced to propagate the information or computed results. For computation, all of the studied algorithms are heavier than typical priority policies like FCFS, which poses a burden to computing devices. Both problems may be solved by developing some distributed algorithms for Problem 1 so that less information or less computation is required for each participating vehicle.

Second, the current objective is to minimize the number of modified decisions (*i.e.*, reverse rate), whose main goal is to ensure fairness. However, this objective does not directly correlate to the objective each priority policy pursues. Although the reverse rate is generally low when a priority policy makes most of the decisions in our experiments, this small amount of reversing is still possible to seriously undermine the objective of that policy. This problem has been shown in the policy-specific metrics in our experiments, where the major policies generally cannot perform as well as they control all the vehicles in the intersection. A possible solution to this problem might be studying the weighted version of Problem 1. By properly designing the weights of the edges, a more complicated objective can be pursued.

Third, the proposed method operates on the decided passing orders of priority policies. Although passing order must be decided to avoid collision no matter what policy is used, some methods may not explicitly produce outputs in the form of passing orders. For example, the work by Mirheli *et al.* [6] directly outputs acceleration profiles for the vehicles, which must be parsed into passing orders to fit the proposed method. However, this not only induces computation overhead but also wastes the motion planning efforts of this policy. Thus, the policies that separate the deciding of passing orders from the other modules (*e.g.*, motion planning) will match the proposed method better.

VI. Conclusion

In this paper, we proposed to enable the coordination of multiple priority policies, which provided a distributed intelligent intersection management system with essential flexibility to deal with complex and dynamic real-world scenarios. To achieve this, the deadlock problem was identified, and a general deadlock-resolution strategy that works for all priority policies was proposed. This strategy involved solving a constrained optimization problem for keeping minimum interference with the operating priority policies with mandatory passing orders unchanged. Building upon foundational concepts from the minimum feedback arc set problem, our work introduced essential modifications to existing algorithms, enabling them to meet the requirements of practical distributed intersection management. In particular, an exact method and a heuristic were studied and adapted. These algorithms were

numerically evaluated with randomly generated traffic data and the results indicated that they could resolve a deadlock in real time while respecting most of the original decisions, demonstrating the effectiveness of the proposed strategy in coordinating priority policies. As connected and autonomous vehicles are designed by different manufacturers or service providers, compatibility, where priority policies coordination is one case, is still an issue. Future directions include the corresponding verification, requirements engineering, and runtime monitoring.

APPENDIX

In the original work of Liu2017 priority policy [5], the vehicles in the state IL (*i.e.*, in an incoming lane and not the first vehicle) are just instructed to "yield all vehicles that its front vehicle yields". However, this is not enough to decide the passing order between a vehicle in IL state and any other vehicles in a different incoming lane. In our experiments, conflicts can happen between any vehicles with intersecting trajectories. Thus, we add the following extensions to this priority policy:

At time step n, suppose vehicles i and j are from different incoming lanes, the conditions for determining if vehicle $j \in \mathcal{U}_i$ has temporal advantage over vehicle i are augmented with the following rules:

(1) $\mathcal{S}_j = FIL, \mathcal{S}_i = IL$ and vehicle i leaves some conflict zones later than vehicle j enters.
(2) $\mathcal{S}_j = IL, \mathcal{S}_i = FIL$ and vehicle j leaves all conflict zones earlier than vehicle i enters.
(3) $\mathcal{S}_j = \mathcal{S}_i = IL$ and vehicle j enters some conflict zones earlier than vehicle i.

$\mathcal{S}_i$ and $\mathcal{S}_j$ denote the states of vehicle i and j respectively. A vehicle is in state FIL if it is in an incoming lane and is the first vehicle. These rules are constructed by replicating the strategy for handling the states FIL and I (*i.e.*, at the intersection). In rules (1) and (2), state IL is regarded as the "low-priority" state with respect to state FIL, just as state FIL with respect to state I in the original rules. Rule (3) is exactly the same as how the original rules handle the cases where two vehicles are in the same state.

REFERENCES

[1] M. Khayatian, M. Mehrabian, E. Andert, R. Dedinsky, S. Choudhary, Y. Lou, and A. Shirvastava, "A survey on intersection management of connected autonomous vehicles," *ACM Transactions on Cyber-Physical Systems (TCPS)*, vol. 4, no. 4, pp. 1–27, 2020.

[2] R. Naumann, R. Rasche, and J. Tacken, "Managing autonomous vehicles at intersections," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 3, pp. 82–86, 1998.

[3] M. VanMiddlesworth, K. Dresner, and P. Stone, "Replacing the stop sign: Unmanaged intersection control for autonomous vehicles," in *International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, 2008, pp. 1413–1416.

[4] S. Azimi, G. Bhatia, R. Rajkumar, and P. Mudalige, "Reliable intersection protocols using vehicular networks," in *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2013, pp. 1–10.

[5] C. Liu, C.-W. Lin, S. Shiraishi, and M. Tomizuka, "Distributed conflict resolution for connected autonomous vehicles," *IEEE Transactions on Intelligent Vehicles (T-IV)*, vol. 3, no. 1, pp. 18–29, 2017.

[6] A. Mirheli, M. Tajalli, L. Hajibabai, and A. Hajbabaie, "A consensus-based distributed trajectory control in a signal-free intersection," *Transportation Research Part C: Emerging Technologies*, vol. 100, pp. 161–176, 2019.

[7] Y. Wu, H. Chen, and F. Zhu, "DCL-AIM: Decentralized coordination learning of autonomous intersection management for connected and automated vehicles," *Transportation Research Part C: Emerging Technologies*, vol. 103, pp. 246–260, 2019.

[8] D. Zhao, Z. Wang, G. Wu, and M. J. Barth, "Trust-aware control for intelligent transportation systems," in *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2021, pp. 377–384.

[9] M. Fang, J. Zhang, J. Thangarajah, and T. Miller, "A general trust framework for multi-agent systems," in *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2021, pp. 332–340.

[10] Y. Li, G. Wu, and M. J. Barth, "Trust-aware resilient control and coordination of connected and automated vehicles," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2023, pp. 4309–4314.

[11] K. Dresner and P. Stone, "Human-usable and emergency vehicle-aware control policies for autonomous intersection management," in *International Workshop on Agents in Traffic and Transportation (ATT)*, vol. 12, 2006, p. 14.

[12] K.-E. Lin, K.-C. Wang, Y.-H. Chen, L.-H. Lin, Y.-H. Lee, C.-W. Lin, and I. H.-R. Jiang, "Graph-based deadlock analysis and prevention for robust intelligent intersection management," *ACM Transactions on Cyber-Physical Systems (TCPS)*, 2023.

[13] G. Oliva, R. Setola, L. Glielmo, and C. N. Hadjicostis, "Distributed cycle detection and removal," *IEEE Transactions on Control of Network Systems*, vol. 5, no. 1, pp. 194–204, 2016.

[14] R. Tamassia, *Handbook of graph drawing and visualization*. CRC press, 2013.

[15] A. Van Zuylen and D. P. Williamson, "Deterministic pivoting algorithms for constrained ranking and clustering problems," *Mathematics of Operations Research*, vol. 34, no. 3, pp. 594–620, 2009.

[16] R. Martí and G. Reinelt, *The linear ordering problem: exact and heuristic methods in combinatorial optimization*. Springer Science & Business Media, 2011, vol. 175.

[17] M. Grötschel, M. Jünger, and G. Reinelt, "A cutting plane algorithm for the linear ordering problem," *Operations research*, vol. 32, no. 6, pp. 1195–1220, 1984.

[18] T. Orenstein, Z. Kohavi, and I. Pomeranz, "An optimal algorithm for cycle breaking in directed graphs," *Journal of Electronic Testing*, vol. 7, no. 1, pp. 71–81, 1995.

[19] R. Kaas, "A branch and bound algorithm for the acyclic subgraph problem," *European Journal of Operational Research*, vol. 8, no. 4, pp. 355–362, 1981.

[20] M. Grötschel, M. Jünger, and G. Reinelt, "Comments on "an exact method for the minimum feedback arc set problem"," *ACM Journal of Experimental Algorithmics (JEA)*, vol. 27, pp. 1–4, 2022.

[21] A. Baharev, H. Schichl, A. Neumaier, and T. Achterberg, "An exact method for the minimum feedback arc set problem," *Journal of Experimental Algorithmics (JEA)*, vol. 26, pp. 1–28, 2021.

[22] P. Eades, X. Lin, and W. F. Smyth, "A fast and effective heuristic for the feedback arc set problem," *Information Processing Letters*, vol. 47, no. 6, pp. 319–323, 1993.

[23] F. J. Brandenburg and K. Hanauer, "Sorting heuristics for the feedback arc set problem," 2011.

[24] M. Simpson, V. Srinivasan, and A. Thomo, "Efficient computation of feedback arc set at web-scale," *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 133–144, 2016.

[25] B. Berger and P. W. Shor, "Approximation alogorithms for the maximum acyclic subgraph problem," in *ACM-SIAM Symposium on Discrete algorithms*, 1990, pp. 236–243.

[26] Y. Saab, "A fast and effective algorithm for the feedback arc set problem," *Journal of Heuristics*, vol. 7, pp. 235–250, 2001.

[27] H. Aujac, "La hiérarchie des industries dans un tableau des échanges interindustriels: et ses conséquences sur la mise en œuvre d'un plan national décentralisé," *Revue économique*, pp. 169–238, 1960.

[28] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com