A Polynomial-time Decentralised Algorithm for Coordinated Management of Multiple Intersections

Tatsuya Iwase\(^1\), Sebastian Stein\(^2\), Enrico H. Gerding\(^2\), Archie Chapman\(^3\)

\(^1\)Toyota Motor Europe NV/SA, Zaventem, Belgium (on leave from Toyota CRDL Inc., Japan)
\(^2\)University of Southampton, Southampton, United Kingdom
\(^3\)The University of Queensland, Brisbane, Australia

tiwase@mosk.tytlabs.co.jp, \{ss2, eg\}@ecs.soton.ac.uk, archie.chapman@uq.edu.au

Abstract

Autonomous intersection management has the potential to reduce road traffic congestion and energy consumption. To realize this potential, efficient algorithms are needed. However, most existing studies locally optimize one intersection at a time, and this can cause negative externalities on the traffic network as a whole. Here, we focus on coordinating multiple intersections, and formulate the problem as a distributed constraint optimisation problem (DCOP). We consider three utility design approaches that trade off efficiency and fairness. Our polynomial-time algorithm for coordinating multiple intersections reduces the traffic delay by about 41% compared to independent single intersection management approaches.

1 Introduction

Autonomous intersection management (AIM) is the intelligent coordination of autonomous vehicles at intersections [Dresner and Stone, 2008]. Here, an intersection manager agent allocates slots in advance to vehicles to minimize the total travel time. AIM has the potential to reduce traffic congestion, the number of accidents, fuel consumption as well as CO\(_2\) emissions [Namazi et al., 2019] compared to using traditional traffic signals. However, many studies of AIM focus on the management of a single intersection and do not cover the coordination of traffic flows across the entire road network [Namazi et al., 2019]. Indeed, the local optimization of one intersection can cause negative externalities on the surrounding intersections. For example, a vehicle passing an intersection can conflict with a platoon of vehicles at a downstream intersection and cause a cascade of braking. Hence, the coordinated management of multiple intersections (CMMI) is required to solve this issue.

However, a key challenge in optimizing traffic over multiple intersections is its inherent computational complexity. As we will discuss later, CMMI contains an NP-hard packing problem. Due to this, centralized approaches such as a mixed integer linear program (MILP) [Bredström and Rönqvist, 2008], microscopic models [Li et al., 2020], queue models [Wu et al., 2012] and agent-based simulations [Jin et al., 2012; Wang et al., 2020] solve the problem for each intersection independently or sacrifice scalability.

Decentralized solutions are generally more scalable, robust without having a single point of failure, and preferable from a privacy perspective, as they do not require sharing the route information of all cars. They are also suitable for the dynamic nature of CMMI, where vehicles newly appear or change their routes, because agents only have to check the local changes, while a centralized controller has to monitor the entire system. However, a naive application of Distributed Constraint Optimization (DCOP) [Fioretto et al., 2018] will again result in controlling each individual intersection independently [Vu et al., 2020], leading to the same issue of ignoring externality mentioned above. Also, DCOP algorithms such as Max-sum [Farinelli et al., 2008] or MGM-k [Pearce et al., 2005] are not scalable for multiple intersections due to their computational and communication complexity [Chapman et al., 2011]. DCOP games [Chapman et al., 2011] are a scalable and suitable approach for large problems. However, their application to CMMI is not trivial because the feasibility of the solution is not guaranteed due to the presence of hard constraints that are typically solved in mathematical programming problems using Lagrange multipliers.

To address the issues above, we formulate a new CMMI game, which is an extension of DCOP games. We cast CMMI as a multiagent system (MAS) where agents need to be coordinated without global knowledge of the entire network or excessive communication between agents. Since the performance of a MAS depends on the computational ability of each agent, we consider three specific formulations: (1) intersection agents, where each agent controls the allocation of vehicles at a particular intersection; (2) vehicle agents, where each agent controls the allocation of slots for a particular vehicle; (3) atomic agents, where each agent corresponds to a vehicle at a particular intersection. We then compare this to an optimal solution computed using an MILP approach.

In short, we provide the first theoretical analysis of CMMI from the perspective of MAS, making the following novel theoretical and empirical contributions: (1) we prove that computing the optimal solution of CMMI is NP-hard, which shows the essential difficulty of coordination in intersection management; (2) we propose the CMMI game and a polynomial-time algorithm to find a feasible Nash equilibrium instead of the optimal solution (given some weak as-
sumptions); and (3) we also propose a heuristic that coordinates multiple intersections. The empirical result shows that our algorithm typically achieves a social cost that is only 2.6% greater than the optimal solution, while being 41% smaller than the traditional AIM in case of car agents.

The remainder of the paper is structured as follows: Section 2 formalises the problem of CMMI. Section 3 presents an algorithm that solves CMMI, along with a theoretical analysis of its efficiency. Section 4 evaluates our algorithm with a real-world data, and Section 5 concludes the paper.

2 Model

In this section, we formulate our model of the coordinated management of multiple intersections (CMMI) as a mixed integer program (MIP). Then we transform CMMI into a distributed formulation based on a DCOP game [Chapman et al., 2011]. The notation is summarised in Appendix A, and all proofs of theorems are provided in Appendix H.

2.1 Coordinated Management of Multiple Intersections

We assume finite time steps $T = \{0, \ldots, T\}$, intersections $\mathcal{I} = \{1, \ldots, I\}$ and cars $\mathcal{K} = \{1, \ldots, K\}$. The road network is represented by a directed graph $\mathcal{G} = (\mathcal{I}, \mathcal{E})$, where edges $\mathcal{E} \subseteq \mathcal{I}^2$ denote road segments. The length of an edge $e \in \mathcal{E}$ is given by $L_e \in \mathbb{N}_+$. We assume constant and common car speed, each car has a fixed route, and $L_e$ denotes the time to traverse $e$. Also, the edges are not FIFO (First In First Out), but cars can change their order while traversing the edges, if this is beneficial. The path of intersection $i$ is a special path that indicates car $k$ leaves its incoming edge to intersection $i$ immediately and enters the outgoing edge at $a_{i,k}$. Each car $k \in \mathcal{K}$ has a fixed route $r_k = (i_{k1}^e, \ldots, i_{kT}^e)$ and a departure time $T_k^e$, and we assume that they are common knowledge. Since there is no allocation at the destination, we denote the intersections that car $k$ needs to travel by $r_k = r_k \setminus \{i_{kT}^e\}$. Then, the set of variables to be allocated is denoted by $\mathcal{IK} = \{(i,k) | k \in \mathcal{K}, i \in r_k\}$ and the solution space is denoted by $\mathcal{A} = \Pi_{(i,k) \in \mathcal{IK}} \mathcal{A}_{i,k}$. We also denote the previous intersection of $j = i_{k}^{l}$ by $\gamma(j,k) = i_{k}^{l-1}$, where $l > 1$. The waiting time of $k$ at $i$ is given by $w(i,k) = a(i,k) - (a(j,k) + L(i,j))$, where $a(i,k) \in \mathcal{A}_{i,k}, a(j,k) \in \mathcal{A}_{i,k}$ and $j = \gamma(i,k)$. If either $a(i,k)$ or $a(j,k)$ is

\begin{align*}
\min_{a \in \mathcal{A}_{\text{feas}}} D(a)
\end{align*}

where $\mathcal{A}_{\text{feas}} \subseteq \mathcal{A}$ is the set of feasible solutions defined by the following constraints:

\begin{align*}
&h_{wb}^{(i,k)} : w(i,k) \in [0, T_{UB}], \forall (i,k) \in \mathcal{IK}, \\
&h_{\text{cap}}^{(i,e)} : K_e^l \leq K_{UB}, \forall t \in T, \forall e \in \mathcal{E}, \\
&h_{\text{dep}}^k : a(i,k) \geq T_k^e, \forall k \in \mathcal{K}, \\
&h_{\text{arr}}^k : a(i,k) + L(i,j) \leq T, l = \gamma(i,k), \forall k \in \mathcal{K}, \\
&h_{\text{col}}^i : X_i = 0, \forall i \in \mathcal{I},
\end{align*}

which consists of several hard constraints $h$. Specifically, constraint (2) bounds the waiting time where $T_{UB}$ is an upper bound. (3) is the constraint of the edge capacity, where $K_{\text{cap}} = \{|(i,k) \in \mathcal{IK} | a(i,k) \leq t\} - \{|(j,k) \in \mathcal{IK} | a(j,k) \leq t\}$ denotes the number of cars on edge $(i,j)$ at time $t$, and $K_{UB}$ is a capacity. Constraints (4) and (5) are the departure and arrival time, respectively. Constraint (6) guarantees no collisions, where $X_i = \{|(i,k) \in \mathcal{IK} | k \neq k', \forall x_i \in \mathcal{X}_i, (p(i,k), p(k',k) \neq x_i, a(i,k) = a(i,k')\}$ is the number of collisions at $i$. We also provide a MILP formulation of CMMI for linear solvers in Appendix B. Figure 2 shows an
example of CMMI, with $I = 7$ and $K = 4$. For example, the route of car 2 is $r_2 = (7, 5, 2, 3)$ and its path at intersection 5 is $p_{5,2} = ((7, 5), (5, 2))$.

CMMI formulates the essential difficulty of the application of intersection management, as we show in the following.

**Theorem 1** Finding an optimal feasible solution of CMMI that minimizes $D(a)$ is NP-hard, even when all the solutions in $A$ satisfy constraints that bound the waiting time, $h_{wb}, h_{cap}$ and $h_{arr}$.

This is proved by polynomial-time reduction from an NP-hard job shop scheduling problem to CMMI (in Appendix H).

### 2.2 Preliminaries of DCOP and DCOP games

We now formulate a distributed version of CMMI based on DCOP games [Chapman et al., 2011]. A DCOP game is a game $(\mathcal{N}, \mathcal{A}, u)$ corresponding to DCOP $(\mathcal{N}, \mathcal{A}, \mathcal{C})$, where $\mathcal{N}$ is a set of agents, $\mathcal{A}_i$ is the action space of agent $j$, $\mathcal{C} = \{c_1, c_2, \ldots\}$ is a set of constraints. A constraint $c = (N_c, f_c)$ is in the set $\mathcal{C}$. The index function is $\chi$.

Note that the constraints in DCOP and DCOP games are not hard but soft, i.e. using penalties. Hence, additional techniques are required when we apply DCOP / DCOP games to a problem with hard constraints that must be satisfied. A DCOP solution is denoted by joint action $a = (a_1, \ldots, a_n) \in A$ and denote a projection of the solution for constraint $c$ as $a^c = (a_j | j \in N_c)$. We sometimes denote $f_c(a) = f_c(a^c)$ and also denote $a_{-j} \in A_{-j} = \prod_{j \notin N} A_j$, $u = (u_1, \ldots, u_n)$ is a list of utility functions of the agents. Each agent $j \in N$ is involved in a set of constraints $C_j = \{c_j | j \in N_c\}$, and has a utility function $u_j(a) = -\sum_{c \in C_j} \gamma(u_j(a))$. The social welfare is defined as $J(a) = \sum_{j \in N} u_j(a)$. It is known that every DCOP game is a potential game with potential function $J(a)$ [Chapman et al., 2011], and has the following desirable property. A best response update (BRU) of an agent is an update of its action assuming the others stay constant, so as to maximize its utility. It is known that a finite sequence of unilateral BRUs, where one agent $j$ is selected and executes a BRU for each iteration, always converges to a pure Nash equilibrium $a$ [Monderer and Shapley, 1996], which satisfies,

$$u_j(a_{j}, a_{-j}) \geq u_j(a'_{j}, a_{-j}).$$

### 2.3 CMMI as a DCOP game

Now we transform the centralized CMMI to the CMMI game, which is a DCOP game. Since the performance and computational / communication complexity of DCOP games depend on the definition of the agents and the size of their neighborhood, we compare three possible definitions of agents: intersection agents ($\mathcal{N} = I$), car agents ($\mathcal{N} = K$) and atomic agents ($\mathcal{N} = I_K$). The action space of atomic agent $(i, k)$ is the same $A_{i,k}$ defined in Section 2.1. The action space of intersection agent $i$ and car agent $k$ are denoted as $A_i = \prod_{j \in r_i} A_{i,j}$ and $A_k = \prod_{j \in r_k} A_{i,k}$, respectively. Note that $A_{i,k}$ is considerably smaller than $A_i$ or $A_k$.

We denote the index function as $\chi$, where $\chi(\text{True}) = 1$, $\chi(\text{False}) = 0$. Also, let $P_{\text{wb}}, P_{\text{col}}, P_{\text{some}}$, $P_2$ and $P_3$ denote the weight constants to prioritize the constraints. These penalties are incurred if the constraints are violated. Then CMMI (1)-(6) is converted to constraints of a DCOP game. For example, in the case of atomic agents, hard constraint $h_{wb}^{i,k}$ corresponds to soft constraint $f_{wb}^{i,k}(a) = \chi(h_{wb}^{i,k}) * P_1$, with neighbor $N_{wb}^{i,k} = \{(\gamma(i, k), k), (i, k)\}$ that includes agents involved in $f_{wb}^{i,k}(a)$. The objective $D(a)$ is also decomposed into a constraint for each agent $f_{\text{delay}}^{i,k}(a) = w_{i,k}$. As for the other types of agents, the set $\mathcal{C}$ is same and $f_c$ and $N_c$ are the aggregation of composing atomic agents. For example, the soft constraints of all intersections on the route are aggregated, in case of a car agent. All differences in $u_j$ and $J$ come from the aggregation, which makes the computational complexity large. Due to the space limit, the full formulation of the three agent models are provided in Appendix C.

The minimization of $D(a)$ coincides with maximizing $J(a)$ in case of car agents as follows:

**Lemma 1** In the case of car agents, $D(a) = -J(a)$ if $a \in \mathcal{A}_{\text{feas}}$.

**Proof.** If $a$ is feasible, the utility of car agent $j$ is $u_j(a) = -\sum_{c \in C_{j}} f_c(a) = -f_{\text{delay}}(a)$. In case of $N = K$, $f_{\text{delay}}(a) = \sum_{i \in r_k} w_{i,k}$. Then, $J(a) = \sum_{k \in K} u_j(a) = -\sum_{k \in K} \sum_{i \in r_k} w_{i,k} = -D(a)$. □

In case of intersection agents and atomic agents, $J(a)$ is not exactly equal to $-D(a)$ but a weighted sum of $w_{i,k}$. This is because $N_{\text{delay}}$ contains two agents, $(\gamma(i, k), k)$ and $(i, k)$, and their waiting times are double counted when computing the utilities of those agents, in case of atomic agents for example. However, we can still use $-J(a)$ as an approximation of $D(a)$.

Note that $\mathcal{A}_{\text{feas}}$ is the set of feasible solutions where the total penalty of all soft constraints $C \setminus C_{\text{delay}}$ is zero. Also, $\mathcal{A}_{\text{feas}}$ is the same for all types of agents because if the total penalty is zero for a type of the agents, it remains zero if we change the way of aggregation.
Algorithm 1 Best response update with downstream reset

1: procedure $a = \text{BRUDR}(a, bestU)$
2: while $a$ violates (7) do
3:     $\Delta = 0, \hat{a} = 0$
4:     for $j \in \mathcal{N}$ do
5:         for $a_j' \in A_j$ do
6:             $a'' = \text{DOWNRESET}(j, a_i, (a_j', a_{-j}))$
7:             $\Delta u_j = u_j(a'') - u_j(a)$
8:             if $\Delta u_j > 0$ then
9:                 if not $bestU$ then
10:                     $a = a''$
11:                 break
12:             else if $\Delta u_j > \Delta$ then
13:                 $\Delta = \Delta u_j, \hat{a} = a''$
14:             if $bestU$ and $\hat{a} \neq \text{None}$ then
15:                 $a = \hat{a}$
16:         Return $a$

Since CMMI is NP-hard, we do not compute the optimal solution. Instead, a pure Nash equilibrium can be computed using the BRU dynamics. However, the feasibility of pure Nash equilibria is not guaranteed by the trivial application of BRU, because they are local optima of $J(a)$. Even worse, it is known that finding a pure Nash equilibrium in a potential game is PLS-complete [Fabrikant et al., 2004]. We address this issue in Section 3.

3 Algorithm and Theoretical Analysis

To address the difficulty of CMMI discussed above, we propose a distributed solution in the sense that agents do not have to know all the routes of other agents, and they only require the information of available time slots for each intersection at the moment. First, we show that our algorithm computes a feasible solution in polynomial time, and the communication complexity is not very intensive either. We then show a negative result that the solution can be arbitrarily inefficient compared to an optimal solution. Last, we describe how to improve the performance of our algorithm in practice. Due to the space limit, most of the sub-routines and details of the practical implementation are found in Appendices E and D.

3.1 Polynomial time feasible algorithm for CMMI

To address the challenges above, we propose a novel algorithm based on BRU. Recall that BRU does not guarantee feasible solutions due to the soft constraints. Since we want to minimize $D(a)$ among feasible solutions, we assume the magnitude of the penalty constants as follows:

$$P_{\text{wad}} \gg P_{\text{col}} \gg P_{\text{none}} \gg P_2 \gg P_1 \gg T.$$ (8)

Since BRU tries to find a local maximum of $J(a)$, (8) gives priority to obtaining a feasible solution before minimizing $D(a)$. In effect, we apply the “big M” method to DCOPs, using it to convert hard constraints to large penalty functions, or soft constraints, in the DCOP objective function. Our algorithm, BRU with downstream reset (BRUDR) is shown in Algorithm 1. Briefly, every time the algorithm allocates a time slot to a car at an intersection, it clears all allocations of the car at downstream intersections to prevent the violation of $h_{\text{wr}}$. We initialize the joint action as being unallocated, $a = \emptyset$, which means $a_{i,k} = \emptyset, \forall (i,k) \in I \times K$.

We consider two variants of the algorithm, one which always updates the agent with the largest improvement in utility (indicated by $bestU = True$) and one which updates any agent ($bestU = False$). The algorithm terminates when the joint action converges to a pure Nash equilibrium (line 2). Otherwise, it computes the improvement in utility of the update for each action of each agent (line 4-7). Without loss of generality, the agent loop is ordered by agent ID (line 4). Also, the action loop is in ascending order of the value $a_j'$ (line 5). The types of agents make a difference here because the different size of neighborhood causes a different aggregation of the costs. In particular, only car agents have a neighborhood over all intersections on their own routes. After that, different from the normal BRU, the algorithm resets the allocation of the downstream intersections of the current agent $j$ to $\emptyset$ using the function DOWNRESET (line 6). Without this process, a car can violate $h_{\text{wr}}$ by creating an infeasible schedule between different intersections, especially in the case of atomic agents with a non-empty initial allocation. If $bestU = False$, the joint action is updated when there is an improvement (line 10). If $bestU = True$, the algorithm finds an update with the largest improvement (line 13), and applies the update (line 15). Finally, the algorithm returns a Nash equilibrium (line 16). With the resetting, BRUDR works as a greedy algorithm that allocates time slots from the agents in the upstream, until all agents are allocated. BRUDR guarantees the feasibility of the solution as follows.

Lemma 2 In cases of the car agents and atomic agents, BRUDR converges to a feasible pure Nash equilibrium, with each agent updating the action at most once, if all the solutions in $A$ satisfy $h_{\text{wr}}, h_{\text{cap}}$ and $h_{\text{arr}}$.

Briefly, this is because agents can always find the earliest allocation that satisfies the soft constraints, because agents can delay their trips as much as they need assuming that the 3 bounding constraints, $h_{\text{wr}}, h_{\text{cap}}$ and $h_{\text{arr}}$ are always satisfied. In practice, this assumption holds when the traffic is not so dense. In case of intersection agents, however, the resetting mechanism can unallocate the other cars on their downstream route, making the feasible updates impossible.

Note that Lemma 2 means that the efficiency of BRUDR depends only on the allocation order of the agents, because all agents choose their allocation only once. As for the computational complexity of BRUDR, it suffers from an exponential size of the search space for the cases of intersection agents and car agents. To address this, we restrict the action space. For each intersection on the route, the car checks the feasibility of each time slot, and allocates the earliest one that satisfies all the soft constraints. Now we can solve CMMI in polynomial time as follows.

Theorem 2 With the restricted action space, BRUDR converges to a feasible pure Nash equilibrium in polynomial time if all solutions in $A$ satisfy $h_{\text{wr}}, h_{\text{cap}}$ and $h_{\text{arr}}$.

Proof. This follows from Lemma 2. Since all agents update at most once, the number of the iterations in BRUDR is less
Theorem 2 shows that our algorithm relaxes the problem of finding a Nash equilibrium to an easier complexity class than PLS-complete, by assuming the three constraints $h_{a,b}$, $h_{\text{cap}}$ and $h_{\text{arr}}$ are satisfied. In practice, the algorithm can be accelerated even further by replacing the best response update of a single player with the stochastic response updates of multiple players. For each iteration, all players simultaneously try to choose their best response to the joint actions in the former iteration, but succeed only with some probability $p$. This is also known as the distributed stochastic algorithm (DSA) and it almost surely converges to a Nash equilibrium in potential games and DCOP games [Chapman et al., 2011].

Note that it is easy to apply our approach to dynamic settings, because of the greedy nature of the distributed algorithms. The agents can simply run BRUDR again when a new vehicle appears or some vehicles change their routes. Since it is the same as if the vehicles existed with the new routes from the beginning (but which had not yet been updated), the same proof for Theorem 2 would hold as well. Also, the communication complexity is not very intensive in our approach. See summary in Appendix F.

Despite of the scalability above, the efficiency of the solution can be arbitrarily worse than the optimal one. We discuss the detail in Appendix G, but Figure 2 provides an illustrative example. In the optimal solution, $D(a_{so}) = 1$ where all cars do not have to brake except for car 1. However, BRUDR can output a Nash equilibrium where all cars except for car 1 brake, resulting in $D(a_{nc}) = 3$. Since we can add more cars after car 4, $D(a_{nc})$ can be arbitrarily worse than $D(a_{so})$.

3.2 Personal Best Actions

Although the inefficiency of CMMI cannot be bounded, in practice, it is still possible to coordinate the intersections and improve efficiency. We propose a technique that can be applied to a wide range of multiagent coordination problems. We can improve the efficiency by just replacing the initial input of BRUDR with the Personal Best Actions (PBA).

$$a_j = \arg \max_{a'_j \in A_j} u_j(a'_j, \emptyset_{-j}), \forall j \in N,$$

(9)

where $\emptyset_{-j}$ means $a_l = \emptyset, \forall l \neq j$. In CMMI, this means that agents choose the earliest time slots without worrying about collisions.

PBA can improve the efficiency compared to the default BRUDR (with initial solution $a = \emptyset$) on average across all instances of CMMI. We show this empirically in the later experimental section, but here we briefly explain why PBA performs better than the default BRUDR. Let $\mathcal{F}$ be the finite set of all the instances of CMMI. Then we split $\mathcal{F}$ into the two disjoint subsets as $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$. The set $\mathcal{F}_1$ consists of special cases where a single update of an agent from PBA can reach the best Nash equilibrium. All of the other instances fall into $\mathcal{F}_2$. In case of $\mathcal{F}_1$, BRUDR starting from PBA always outputs the best Nash equilibrium. Note that the same thing never happens with the default BRUDR, because it cannot terminate in one step from $a = \emptyset$ when $N \geq 2$. Then, BRUDR with PBA always outperforms the default BRUDR for $\mathcal{F}_1$. For $\mathcal{F}_2$, both BRUDR with PBA and the default BRUDR cannot guarantee the optimality, and show the same performance on average. Then, BRUDR with PBA always outperforms the default BRUDR across all instances $\mathcal{F}$ on average. Since the idea of PBA is simple, we can apply it to other resource allocation problems.

4 Experimental Results

Given that we do not have any efficiency bounds, we evaluate BRUDR using simulations with the data set of vehicle trajectories in Athens that were collected as part of the pNEUMA field experiment [Barmpounakis and Geroliminis, 2020]. We analyse a zone of Athens including 24 intersections and extract the vehicle trajectories that pass those intersections to compute the routes of the vehicles. Based on this, we create a traffic demand model as a Poisson distribution for each route, with the mean parameter estimated from the route data. We generate cars for each time slot randomly according to the Poisson distribution, until the total number of cars reaches the parameter $K$. We simulate different levels of car density, by changing the parameter $K$. We denote the default BRUDR and BRUDR with PBA as $\text{default}$ and $\text{pba}$, respectively. Since our interest is in the problem instances where cars conflict with each other, we exclude cases without conflicts, i.e., where $D(a_{so}) = 0$. We also exclude the easy cases where both $\text{pba}$ and $\text{default}$ reach $a_{so}$.

In what follows we first compare the performance of BRUDR and the traditional first-come-first-served AIM [Dresner and Stone, 2008] with different settings. We then evaluate the scalability of the algorithm to larger settings. All the error bars in the figures show 95% confidence intervals.

For reproducibility of the experiments, the details of the setting are in Appendix I. The code is also available in the supplementary material.

4.1 Small Settings

We compare the performance of BRUDR with an optimal solution by changing the definitions of agents $(K, \mathcal{I}$ and $\mathcal{I} \times K)$, with different parameters $K \in \{20, 30\}$, and different initial allocations (PBA and $a = \emptyset$). Since finding an optimal solution is NP-hard, we only focus on the small size problems. Note that the case of the atomic agents with the initial actions $a = \emptyset$ corresponds to the traditional AIM that uses a FCFS allocation policy [Dresner and Stone, 2008]. We run 300 simulations for each setting.

Figure 3 shows the ratio of total delay compared to the optimum one. Since solutions can be infeasible in case of intersection agents, it includes only the result of feasible solutions. As discussed in Section 3.2, the delay of $\text{pba}$ becomes smaller than the one of $\text{default}$ and close to the optimum, in case of the car agents. Note that there is no clear improvement in case of intersection agents. This is because the improvement requires updates at multiple intersections and cannot be done.
by a single update at an intersection, so \( F_1 \) is almost empty. Hence, this result shows that the case of car agents enables coordination of multiple intersections and exceeds the performance of the single intersection management (about 41% points reduction on average, while it is 2.6% points greater than the delay of optimal solutions when \( K = 20 \)). In case of car agents, \( pba \) achieves the social optimum even in the pathological case of Example 1, maintaining the platoon (car 3 and 4). However, the ratio becomes smaller when \( K \) increases, in all settings. Since larger \( K \) causes more conflicts among cars, this indicates that even the optimum traffic deteriorates by congestion close to Nash equilibria, and there is little room remained to improve. However, cars can have negative externalities that cause cascades of braking as in Figure 2, even in large problems. We analyze this aspect further in Appendix J.

Since the objective of CMMI is minimization of total delay \( D(a) \), it can cause unfair allocations. We show an interesting finding that the fairness depends on the agent models. Figure 4 shows the Gini index, which is a degree of unfairness among cars. PBA with the intersection agents is fairer than PBA with the other agents with 95% significance, because all cars that pass an intersection join a neighborhood and share the common utility. This indicates there may be a tradeoff between efficiency and fairness, since intersection agents are more fair but are worse in social welfare, while car agents have the highest social welfare but lower fairness. This tradeoff can also be seen in Example 1, and as is often observed MAS in allocation problems in general.

### 4.2 Scalability

Next, we evaluate the scalability of BRUDR in case of car agents. We change the number of cars \( K \in \{10, 20, 30, 40, 50, 70, 90, 110, 130\} \), and run 100 simulations for each case. Since most of the runtime is spent on computing \( c_{\text{cap}} \), we compare the cases with and without the constraint. Figure 5 shows the runtime of BRUDR. Though it shows that our algorithm is polynomial time in \( K \), the runtime can exceed \( 10^4 \) seconds if the computation of \( c_{\text{cap}} \) is included. However, if we can assume the traffic is not so dense and can omit the computation of \( c_{\text{cap}} \), our algorithm is scalable to more than 100 cars. Also, PBA contributes to reducing the runtime because all agents are already allocated in the initial solution. Note that the current implementation uses a single CPU just for theoretical verification. However, our polynomial-time algorithm can be easily executed in a distributed fashion and therefore scale, technically.

### 5 Conclusions

We model the coordination problem of multiple intersections with the theory of DCOP. The goal is the theoretical analysis of the problem, and to find an efficient feasible allocation in polynomial time. To this end, we present a polynomial-time algorithm that guarantees to find a feasible solution and we present a novel modification to improve its efficiency in practice. In addition, through numerical simulation using real-world data, we show that our algorithm outperforms traditional single intersection management approaches. Since our problem is a special case of the job shop scheduling problem, our work may also be applicable to a wide variety of scheduling problems. Future directions include an algorithm that bounds the inefficiency with communications among agents, as well as extending the problem to include vehicle routing rather than assuming fixed routes.
6 Ethics statement

Our work has positive ethical impacts on society. This is because our algorithm can benefit society and the environment by reducing total travel time and pollution. A possible negative impact is that people might feel unfair because some vehicles have to wait more than the others. We could address this by looking into envy-free algorithm in future work.

References

[Barmpounakis and Geroliminis, 2020] Emmanouil Barmpounakis and Nikolas Geroliminis. On the new era of urban traffic monitoring with massive drone data: The pneuma large-scale field experiment. Transportation Research Part C: Emerging Technologies, 111:50–71, 2020.
[Bredström and Rönnqvist, 2008] David Bredström and Mikael Rönnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. European Journal of Operational Research, 191(1):19–31, 2008.
[Chapman et al., 2011] Archie C Chapman, Alex Rogers, and Nicholas R Jennings. Benchmarking hybrid algorithms for distributed constraint optimization games. Autonomous Agents and Multi-Agent Systems, 22(3):385–414, 2011.
[Dresner and Stone, 2008] Kurt Dresner and Peter Stone. A multiagent approach to autonomous intersection management. Journal of artificial intelligence research, 31:591–656, 2008.
[Farinelli et al., 2004] Alex Farinelli, Christos Papadimitriou, and Kunal Talwar. The complexity of pure nash equilibria. In Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, pages 604–612, 2004.
[Farinelli et al., 2008] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas R Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2, pages 639–646, 2008.
[Fioretto et al., 2018] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. Distributed constraint optimization problems and applications: A survey. Journal of Artificial Intelligence Research, 61:623–698, 2018.
[Gonzalez, 1982] Teofilo Gonzalez. Unit execution time shop problems. Mathematics of Operations Research, 7(1):57–66, 1982.
[Jin et al., 2012] Qiu Jin, Guoyuan Wu, Kanok Boriboonsomsin, and Matthew Barth. Multi-agent intersection management for connected vehicles using an optimal scheduling approach. In 2012 International Conference on Connected Vehicles and Expo (ICCVE), pages 185–190. IEEE, 2012.
[Li et al., 2020] Bai Li, Youmin Zhang, Ning Jia, and Xiaoyan Peng. Autonomous intersection management over continuous space: A microscopic and precise solution via computational optimal control. IFAC-PapersOnLine, 53(2):17071–17076, 2020.

[Monderer and Shapley, 1996] Dov Monderer and Lloyd S Shapley. Potential games. Games and Economic Behavior, 14(1):124–143, 1996.
[Namazi et al., 2019] Elnaz Namazi, Jingyue Li, and Chaoru Lu. Intelligent intersection management systems considering autonomous vehicles: A systematic literature review. IEEE Access, 7:91946–91965, 2019.
[Nemhauser et al., 1978] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. Mathematical Programming, 14(1):265–294, 1978.
[Pearce et al., 2005] Jonathan P Pearce, Rajiv T Maheswaran, and Milind Tambe. Local algorithms for distributed constraint optimization in dynamic, anytime environments. In Ambient Agents Workshop, AAMAS. Citeseer, 2005.
[Vetta, 2002] Adrian Vetta. Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings., pages 416–425. IEEE, 2002.
[Vu et al., 2020] Huan Vu, Samir Aknine, Sarvapali Ramchurn, and Alessandro Farinelli. Decentralised multi-intersection congestion control for connected autonomous vehicles. In Multi-Agent Systems and Agreement Technologies, pages 37–51. Springer, 2020.
[Wang et al., 2020] Yunpeng Wang, Pinlong Cai, and Guangquan Lu. Cooperative autonomous traffic organization method for connected automated vehicles in multi-intersection road networks. Transportation Research Part C: Emerging Technologies, 111:458–476, 2020.
[Wu et al., 2012] Jia Wu, Abdeljalil Abbas-Turki, and Abdellah El Moudni. Cooperative driving: an ant colony system for autonomous intersection management. Applied Intelligence, 37(2):207–222, 2012.
A Summary of notation

| Time | Intersection | Car | Edge (road segment) | Length of edge | Allocation | Route of car |
|------|--------------|-----|--------------------|----------------|------------|-------------|
|      | t ∈ T       | i ∈ I | k ∈ K              | e ∈ E          |            |             |
|      | a(i,k) ∈ A(i,k) ⊆ T ∪ {∅} | r_k = (i_t^k, ..., i_t^k) | r_k = r_k \ {i_t^k} |            |             |
|      | T_W | γ(i,k) = i_t^k | w(i,k) | D(a) = ∑ k∈K ∑ i∈r_k w(i,k) |            |             |
|      |                |            | P_i | x_i ∈ X_i | c = (N_c, f_c) ∈ C |             |
|      |              |            |             | ti |             |             |

B MILP formulation of CMMI

Variables of MILP y consists of \( y_{t,k,e} \in \{0,1\}, \forall t \in T, \forall k \in K, \forall e \in E_k \), where \( E_k = \{(i,j) | i \in r_k^- \} \). The value of \( y_{t,k,e} = 1 \) if and only if \( a(i,k) = t \). The wait time of car \( k \) to enter \( e = (i,j) \) is:

\[
c(y,k,e) = \sum_{t_{out} \in T} t_{out} \cdot y_{t_{out},k,e} - \sum_{t_{in} \in T} t_{in} \cdot y_{t_{in},k,γ(i,k),i} - L(γ(i,k),i).
\]

Given this, the MILP formulation is as follows:

\[
\min_y \sum_{k \in K} \sum_{e \in E_k} c(y,k,e)
\]

subject to:

(11) \( c(y,k,e) ≥ 0, \forall k \in K, \forall e \in E_k \),

(12) \( c(y,k,e) ≤ T_{UB}, \forall k \in K, \forall e \in E_k \),

(13) \( \sum_{k \in E_k} y_{t,k,e} ≤ K_{UB} \forall t \in T, \forall e \in E \),

(14) \( t_{in} = 1, \forall k \in K \),

(15) \( y_{t,k,e} = 0, \forall k \in K, \forall t > T - L(i_t^k \ldots i_t^k) \),

(16) \( \sum_{(e,i,k) \in K} e_{in} \in E_k \),

(17) \( x_{t,k,e} = 1, \forall k \in K, \forall e \in E_k \),

(18) \( x_{t,k,e} ≤ 1, \forall t \in T, \forall e \in E \),

(19) \( x_{t,k,e} ≤ 1, \forall t \in T, \forall k \in K \).

where \( K_e = \{ k \in K | e \in E_k \} \). Constraints (11)-(16) correspond to (2)-(6) in Section 2. Constraints (17)-(19) are newly added to guarantee that only one vehicle is allocated at a time at an edge.

C Constraints for three agent models

We define a set of cars that pass edge \((i,j)\) as:

\[
K_{(i,j)} = \{ k \in K | i \in r_k^-, j \in r_k^+ \}.
\]

Also, we define the following hard constraints that apply to to all three models. First, car \( k \) can be allocated at \( i \) only when it is allocated at the upstream intersection: \( γ(i,k) \)

\[
h_{fwd}^{i,k}(a) = a(i,k) = 0 ∨ a(γ(i,k),k) ≠ 0, \forall (i,k) \in IK
\]

(20)

Second, car \( k \) is allocated at \( i \):

\[
h_{alloc}^{i,k}(a) = a(i,k) ≠ 0, \forall (i,k) \in IK
\]

(21)

C.1 Constraints for atomic agents

For the case of atomic agents, the set of constraints \( C \) is formulated as follows:

(22)

\[
c_{delay}^{i,k} : N_{delay}^{i,k} = \{(γ(i,k), k), (i, k)\},
\]

(23)

\[
f_{delay}^{i,k}(a) = u(i,k), \forall (i,k) \in IK,
\]

(24)

\[
c_{wb}^{i,k} : N_{wb}^{i,k} = \{(γ(i,k), k), (i, k)\},
\]

(25)

\[
f_{wb}^{i,k}(a) = χ(-h_{wb}^{i,k}) ∗ P_1, \forall (i,k) \in IK,
\]

(26)

\[
c_{cap}^{i,k} : N_{cap}^{i,k} = \{(i,k), (j,k)| k \in N, i = γ(j,k)\},
\]

(27)

\[
f_{cap}^{i,k}(a) = χ(-h_{cap}^{i,k}) ∗ P_1, \forall t \in T, \forall e \in E,
\]

(28)

\[
c_{dep}^{i,k} : N_{dep}^{i,k} = \{(i_t^k \ldots i_t^k)\},
\]

(29)

\[
f_{dep}^{i,k}(a) = χ(-h_{dep}^{i,k}) ∗ P_2, \forall k \in K,
\]

\[
c_{arr}^{i,k} : N_{arr}^{i,k} = \{(γ(i_k^k), k), k\},
\]

\[
f_{arr}^{i,k}(a) = χ(-h_{arr}^{i,k}) ∗ P_1, \forall k \in K,
\]

\[
c_{col}^{i,k} : N_{col}^{i,k} = \{(i, k) \in IK\},
\]

\[
f_{col}^{i,k}(a) = χ(-h_{col}^{i,k}) ∗ P_1, \forall t \in T, \forall e \in E,
\]

The correspondence between the hard constraints \( h \) (2)-(6) and the soft constraints \( f \) (23)-(27) is straightforward. Also, (20) and (21) correspond to (28) and (29). The neighborhood \( N_e \) of each constraint includes agents that are involved in the soft constraint. (22) corresponds to the objective \( D(a) \).
C.2 Constraints for car agents

\[ c^k_{\text{delay}} : N^k_{\text{delay}} = \{ k \}, \]
\[ f^k_{\text{delay}}(a) = \sum_{i \in r_k} w_{i,k}, \forall k \in K, \quad (30) \]
\[ c^k_{\text{wb}} : N^k_{\text{wb}} = \{ k \}, \]
\[ f^k_{\text{wb}}(a) = (1 - \prod_{i \in r_k} \chi(h_{\text{wb}}^{i,k}))P_1, \forall k \in K, \quad (31) \]
\[ c^{(i,e)}_{\text{lcap}} : N^{(i,e)}_{\text{lcap}} = \mathcal{K}_{(i,j)}, \]
\[ f^{(i,e)}_{\text{lcap}}(a), \forall t \in T, \forall e = (i,j) \in \mathcal{E}, \quad (32) \]
\[ c^k_{\text{dep}} : N^k_{\text{dep}} = \{ k \}, \]
\[ f^k_{\text{dep}}(a), \forall k \in K, \quad (33) \]
\[ c^k_{\text{arr}} : N^k_{\text{arr}} = \{ k \}, \]
\[ f^k_{\text{arr}}(a), \forall k \in K, \quad (34) \]
\[ c^i_{\text{col}} : N^i_{\text{col}} = \{ k \in K | i \in r_k \}, \]
\[ f^i_{\text{col}}(a), \forall i \in \mathcal{I}, \quad (35) \]
\[ c^k_{\text{fwd}} : N^k_{\text{fwd}} = \{ k \}, \]
\[ f^k_{\text{fwd}}(a) = (1 - \prod_{i \in r_k} \chi(h_{\text{fwd}}^{i,k}))P_{\text{fwd}}, \forall k \in K, \quad (36) \]
\[ c^k_{\text{alloc}} : N^k_{\text{alloc}} = \{ k \}, \]
\[ f^k_{\text{alloc}}(a) = (1 - \prod_{i \in r_k} \chi(h_{\text{alloc}}^{i,k}))P_{\text{none}}, \forall k \in K, \quad (37) \]

C.3 Constraints for intersection agents

\[ c^{(i,j)}_{\text{delay}} : N^{(i,j)}_{\text{delay}} = \{ i, j \}, \]
\[ f^{(i,j)}_{\text{delay}}(a) = \sum_{k \in \mathcal{K}_{(i,j)}} w_{(i,j),k}, \forall (i,j) \in \mathcal{E}, \quad (38) \]
\[ c^{(i,j)}_{\text{wb}} : N^{(i,j)}_{\text{wb}} = \{ i, j \}, \]
\[ f^{(i,j)}_{\text{wb}}(a) = (1 - \prod_{k \in \mathcal{K}_{(i,j)}} \chi(h_{\text{wb}}^{(i,j)k}))P_1, \forall (i,j) \in \mathcal{E}, \quad (39) \]
\[ c^{(t,e)}_{\text{lcap}} : N^{(t,e)}_{\text{lcap}} = \{ i, j \}, \]
\[ f^{(t,e)}_{\text{lcap}}(a), \forall t \in T, \forall e = (i,j) \in \mathcal{E}, \quad (40) \]
\[ c^k_{\text{dep}} : N^k_{\text{dep}} = \{ k \}, \]
\[ f^k_{\text{dep}}(a), \forall k \in K, \quad (41) \]
\[ c^k_{\text{arr}} : N^k_{\text{arr}} = \{ (i,O), k \}, \]
\[ f^k_{\text{arr}}(a), \forall k \in K, \quad (42) \]
\[ c^i_{\text{col}} : N^i_{\text{col}} = \{ i \}, \]
\[ f^i_{\text{col}}(a), \forall i \in \mathcal{I}, \quad (43) \]
\[ c^k_{\text{fwd}} : N^k_{\text{fwd}} = \{ (i,j) \}, \]
\[ f^k_{\text{fwd}}(a) = (1 - \prod_{k \in \mathcal{K}_{(i,j)}} \chi(h_{\text{fwd}}^{(i,j)k}))P_{\text{fwd}}, \forall (i,j) \in \mathcal{E}, \quad (44) \]
\[ c^i_{\text{alloc}} : N^i_{\text{alloc}} = \tau_k, \]
\[ f^i_{\text{alloc}}(a) = (1 - \prod_{k \in \mathcal{K}_{(i,j)}} \chi(h_{\text{alloc}}^{(i,j)k}))P_{\text{none}}, \forall i \in \mathcal{I}, \quad (45) \]

D Practical Implementation

For example, the infrastructure and the system can work as follows. When a driver sets the destination into a car navigation system, the car communicates with the intersection on its route and asks for the availability of the slots at that moment. Then the car checks which slots can satisfy the constraints of CMMI, and reserves feasible slots at relevant intersections. This computation can be done in a distributed manner by each car. The intersections work as a database that store the usage of the slots, and mediate the communication between cars by sharing the information of available slots.

The advantage of this distributed design is threefold. First, it is computationally scalable in dynamic setting. When a new vehicle appears in the environment or some vehicles change their routes, only the agents influenced by those changes has to update their own trips, while a centralized approach has to compute the entire allocation. Second, it is preferable from a privacy perspective, because we do not want cars to have much information about the routes of other cars. Third, there is the real-world robustness of having optimisation nodes scattered on edge computing devices across a physical network, rather than a centralised optimisation node that can be a single point of failure. Furthermore, our approach can be applied to the mixed traffic of manual and autonomous vehicles, because it also works for connected (but not necessarily autonomous) vehicles. In that case, the agent can be on e.g. the smartphone and can communicate with the intersections, compute the algorithm, and show riders the timing that they can pass the intersections. Additional infrastructure can also help such as a traffic light that shows the timing to the drivers of manual vehicles.

E Algorithms

Algorithm 2 shows the details in case of atomic agents. The other cases are in Appendix E, but the idea is the same. The loop goes through the route in the right sequence (line 3). Then the reset is done at all intersections after i (line 4-7).

To address the exponential size of the search space, we replace the action space in line 5 of Algorithm 1 with the action space restricted by Algorithm 5. In more detail, for each intersection on the route (line 3), it checks the feasibility of each time slot (line 4), and allocates the earliest one that satisfies all the soft constraints (line 14-21). Algorithm 5 shows the case of the car agents. The case of the intersection agents is in Appendix E.

Algorithm 6 shows the A function for intersection agents.
Algorithm 2 Downstream reset (in case of atomic agents)

1: procedure $a' = \text{DownReset}(j, a, a')$
2: \hspace{1em} (i, k) = j, \text{down} = False$
3: \hspace{1em} for $i' \in r_k$ do
4: \hspace{2em} if $\text{down}$ then
5: \hspace{3em} $a'_{(i', k)} = \emptyset$
6: \hspace{2em} if $i = i'$ then
7: \hspace{3em} $\text{down} = True$
8: \hspace{1em} Return $a'$

Algorithm 3 Downstream reset (in case of car agents)

1: procedure $a' = \text{DownReset}(j, a, a')$
2: \hspace{1em} $k = j$, $\text{down} = False$
3: \hspace{1em} for $i \in r_k$ do
4: \hspace{2em} if $\text{down}$ then
5: \hspace{3em} $a'_{(i, k)} = \emptyset$
6: \hspace{2em} if $a_{(i, k)} \neq a'_{(i, k)}$ then
7: \hspace{3em} $\text{down} = True$
8: \hspace{1em} Return $a'$

Algorithm 4 Downstream reset (in case of intersection agents)

1: procedure $a' = \text{DownReset}(j, a, a')$
2: \hspace{1em} $i = j$, $\text{down} = False$
3: \hspace{1em} for $k \in \{k | i \in r_k \}$ do
4: \hspace{2em} if $a_{(i, k)} = a'_{(i, k)}$ then
5: \hspace{3em} continue
6: \hspace{2em} for $i_k \in r_k$ do
7: \hspace{3em} \hspace{1em} if $\text{down}$ then
8: \hspace{3em} \hspace{2em} $a'_{(i, k)} = \emptyset$
9: \hspace{3em} \hspace{2em} if $i_k = i$ then
10: \hspace{3em} \hspace{3em} $\text{down} = True$
11: \hspace{1em} Return $a'$

Algorithm 5 Fastest feasible allocation (in case of car agents)

1: procedure $a'_j = A_j(a)$
2: \hspace{1em} $a'_j = a_j$
3: \hspace{1em} for $i \in r_j$ do
4: \hspace{2em} for $t \in T$ do
5: \hspace{3em} $a'_{(i,j)} = t$
6: \hspace{3em} $\text{feasible} = True$
7: \hspace{3em} for $i_l \in C_j \setminus \{v_{	ext{delay}}\}$ do
8: \hspace{4em} if $f_i(a') > 0$ then
9: \hspace{5em} $\text{feasible} = False$
10: \hspace{5em} break
11: \hspace{3em} if $\text{feasible}$ then
12: \hspace{4em} break
13: \hspace{1em} Return $a'$

F Communication complexity

The communication complexity depends on what we define an agent to be. For example, atomic agents only have to communicate with an upcoming intersection independently, while a car agent has to communicate with all intersections on its route. Similarly, intersection agents have to communicate with all cars that can pass the intersection at the same time. However, with the system design above, we do not need communication among cars, but a car only has to communicate with intersections on its route. Also, according to Lemma 2, each agent has to communicate at most once in case of atomic and car agents. The size of the message between a car and an intersection to check the available slots is at most $T^w(\text{max degree of the road network})$, and then is polynomial in the problem size. When $\text{bestU} = True$ in BRUDR, the same message exchange as in MGM (Maximum Gain Message [Pearce et al., 2005]) is needed, but it just requires one more message to share the expected improvement in utility.

G Negative result on unbounded PoA

Even though BRUDR finds a pure Nash equilibrium, it is not always the maximizer of the social welfare $J(a)$, even in case of the car agents, where the social optimum coincides with the optimal solution of $D(a)$. In this section, we discuss the possibility to bound the PoA of CMMI. First, we define the PoA as follows.

$$PoA = \max_{a_{so} \in \mathcal{A}_{so}} \frac{|J(a_{so})| + \epsilon}{|J(a_{so})| + \epsilon}$$

(46)

where $\mathcal{A}_{so}$ is the set of all feasible Nash equilibria and $a_{so}$ is a social optimum. The lower value of PoA is desirable to guarantee the lower bound of the efficiency, and $PoA = 1$ is the best. We add a constant $\epsilon > 0$ because $J(a_{so})$ can be zero. Although there are techniques to bound the PoA based on the submodularity of the objective function [Nemhauser et al., 1978; Vetta, 2002], unfortunately they cannot be applied to CMMI. This is because our objective function $J(a)$ does not satisfy the “non-decreasing” property, which is a necessary condition of using these techniques. (Allocating time slots can violate soft constraints, which decreases $J(a)$). Even worse, we have the following negative result.

**Theorem 3** The PoA of CMMI is unbounded.

**Proof.** The following example shows a case where PoA can be unbounded.

**Example 1** The road network and routes of cars are shown in Figure 2. There are 4 cars $K = \{1, 2, 3, 4\}$ and 7 intersections $I = \{1, \ldots, 7\}$. The cars have different departure time as $T_1 = T_2 = 0, T_3 = 1, T_4 = 2$. All links have the same length $L_e = 5$, and $T_{UB} = 1$.

Let’s assume car agents for now. Since car 1 and car 2 depart at $t = 0$, one of them has to brake at intersection $i = 5$ at $t = 5$. If car 1 brakes, all other cars can go through without braking, then the solution is a social optimum $a_{so}$, which is also a Nash equilibrium. In this case, the total delay is $D(a_{so}) = 1$ and the social welfare is $J(a_{so}) = -D(a_{so}) = -1$. However, if car 2 brakes at $i = 5$, car 2 and 3 arrive at $i = 2$ at the same time $t = 11$. In this case, car 3 has to brake because $T_{UB} = 1$. Also, car 4 has to brake to avoid car 3. The solution is also a Nash equilibrium, because only car 1 can improve the efficiency, but
Algorithm 6 Fastest feasible allocation (in case of intersection agents)

1: procedure $a'_j = A_j(a)$
2:   $a'_j = a_j$
3:   for $k \in \{k \in K|j \in r_k^i\}$ do
4:     for $t \in T$ do
5:       $t_{min} = T_k^o$
6:       if $j \neq i_k^t$ then
7:         $i = \gamma(j, k)$
8:       if $a(i, k) = \emptyset$ then
9:         $a(i, k) = \emptyset$
10:   else
11:     $t_{min} = a(i, k) + L(i, j)$
12:   if $t < t_{min}$ then
13:     continue
14:   $a'(j, k) = t$
15:   feasible = True
16:   for $c_i \in C_j \setminus \{c_{delay}\}$ do
17:     if $f_i(a') > 0$ then
18:       feasible = False
19:       break
20:   if feasible then
21:       break
22: Return $a'$

has no incentive to do so. The total delay of the solution $a_{ne}$
is $D(a_{ne}) = 3$ and the social welfare is $J(a_{ne}) = -3$. Thus $PoA = |J(a_{ne})|/J(a) + \epsilon = (3 + \epsilon)/(1 + \epsilon)$. Note that we can make instances that have more cars departing from $i = 4$ after car 4. In general, $PoA = (N - 1 + \epsilon)/(1 + \epsilon)$. Then, $\lim_{N \to \infty} PoA = \infty$. Note that the Nash equilibria above are also Nash equilibria in case of other agent types. Then similarly, PoA is also unbounded in those cases.

As we can see in Example 1, ensuring smooth passage of a platoon of cars (car 3 and 4) is a key to traffic efficiency. However, Theorem 1 and 3 indicate that it is difficult to create a platoon going through multiple intersections without braking. In the following section, we propose a technique that can address this issue.

H Proofs

H.1 Proof of Theorem 1

Proof. We show the detail of the reduction of job shop scheduling (JSP) to CMMI. It is known that the following JSP is NP-hard [Gonzalez, 1982]. There are $n \geq 1$ independent jobs $J = \{J_1, \ldots, J_n\}$ to be processed by $m$ machines $M = \{M_1, \ldots, M_m\}$. Each job $J_i$ consists of $l_i$ tasks, and $j$th task of $J_i$ is denoted by $\tau(i,j)$. The mapping between tasks and machines is given by $q(i,j)$, which means that task $\tau(i,j)$ is processed by $M_{q(i,j)}$. The $j + 1$th task cannot be executed before the $j$th task terminates. The processing time of task $\tau(i,j)$ is $t(i,j) = 1$. A schedule $S$ consists of a sequence of assignments of jobs to machines in such a way that:

- no job meets with more than one machine at a time
- no machine works with more than one job at a time, and
- $J_i$ meets machine $M_j$ for exactly $t(i,j)$ time units.

No task is interrupted during the process. Let $f_i(S)$ represent the finishing time for job $J_i$ in schedule $S$. The goal of the problem is to find schedule $S$ that minimises the mean finishing time (MFT), $\sum_{J \in \mathcal{S}} f_i(S)/n$.

Now we show that any of this JSP can be reduced to a special case of CMMI where:

- The set of cars $\mathcal{K}$ is the set of jobs $\mathcal{J}$.
- The set of intersections $\mathcal{I}$ is the set of machines $\mathcal{M}$.
- The road network contains edges that cover all the consecutive two tasks.
- $T$ and $T_{UB}$ and $K_{UB}$ are large enough so that $h_{wb}, h_{lcap}$ and $h_{arr}$ are not violated.
- The objective function of CMMI $D(a)$ is MFT multiplied by constant $K$.

H.2 Proof of Lemma 2

Proof. First, we assume car agents, or atomic agents. Because of the order of the penalty constants as in (8), BRUDR tries to satisfy the constraints one by one with the priority. First, the largest $P_{fwd}$ forces updates from upstream agents, and the first term of $c_{fwd}$ is never violated by the downstream reset. Next, agents can always find the earliest allocation that satisfy $c_{col}$ and $c_{exc}$, because agents can delay their trips as much as they need to avoid collisions without the 3 bounding constraints, $c_{ubs}, c_{lcap}$ and $c_{arr}$. Then, the other hard constraints, the second term of $c_{fwd}$ and $c_{dep}$ also can be satisfied with the same reason without violating other constraints. Since agents can find the earliest allocation without violating the hard constraints of other agents, and later updating agents do not change the earliest allocations of already allocated agents, BRUDR converges to a pure Nash equilibrium with each agent updating the action at most once.

Copyright International Joint Conferences on Artificial Intelligence (IJCAI), 2022. All rights reserved.
This does not hold in case of intersection agents, and Figure 6 shows an example. The two cars depart at the same time, and conflict at intersection 2. Let’s say BRUDR allocates from the smallest ID. Then, agent of intersection 1 allocates the departure time of car 2 first, then intersection 2 computes the allocation. Because of the largest $P_{\text{fwd}}$, car 1 cannot be allocated yet and then only car 2 is allocated, as $a_{2,1} = \emptyset, a_{2,2} = 5$ for example. Then intersection 3 allocates both cars. After the first updates of all intersections, intersection 2 tries to update again because car 1 is not allocated there yet. Then it tries to update as $a_{2,1} = 5, a_{2,2} = 6$, for example. However, this fails because car 2 also updates at the intersection and has to reset at the downstream intersection 3, which results in the same $P_{\text{none}}$ but increases a unit time of delay. Then the update is not the best response and is not executed, remaining $a_{2,1} = \emptyset$ which is infeasible.

I Experimental settings

- We call numpy.random.seed at the beginning of the main code with different seed for each run of simulation.
- We use CPLEX 12.7.1, Python 3.8, Red Hat Enterprise Linux Server release 6.10 and Intel Xeon CPU E5-2670 (2.60 GHz) with 8 cores, 132 GB memory to run the experiments.
- The source code is in the supplementary material. We do not use any DCOP simulators. All the codes are implemented from scratch.
- We use pNEUMA dataset of zone 4, on the date of 24/10/2018 from 8:30 to 9:00 (Figure 7).
- The real-world data of vehicle trajectory is provided in Latitude-Longitude coordinate system. The code converts the data to East-North-Up coordinate. Then the code scans all data points in a vehicle trajectory and decides that an intersection is on the route if a data point is inside a radius ($r = 5 \text{ m}$) from the center of the intersection. This conversion procedure is also included in the source code.

J Experiment on externality among vehicles

As we saw in Figure 3, the difference between algorithms disappears when the problem size becomes large. However, cars can have negative externalities that cause cascades of braking as in Figure 2, even in large problems. To see this aspect, we compute the negative externality of agent $i$ as follows.

$$ex_i = D(a_{\text{ne}}) - D(a_{\text{ne}}^{-i}),$$

where $a_{\text{ne}}^{-i}$ is a Nash equilibrium when agent $i$ is absent. Note that $ex_i$ is positive and a larger value is worse. Figure 8 shows the worst externality of agents $\max_i ex_i$ with different number of cars $K$ when cars are controlled by \textit{default}. For each $K$, we conducted 200 experiments. Also note that the computational complexity of $ex_i$ is large and this experiment is difficult to scale. It shows that the negative externality of agents becomes larger when the size of the problems becomes larger. Meanwhile, we also see the improvement by \textit{pba} defined as $D(e_{\text{ne}}^{\text{default}}) - D(e_{\text{ne}}^{\text{pba}})$, where $e_{\text{ne}}^{\text{default}}$ and $e_{\text{ne}}^{\text{pba}}$ are Nash equilibria computed by the default BRUDR and BRUDR starting from \textit{pba}, respectively. Figure 9 shows the ratio of the improvement over the worst negative externality. It shows that the improvement by \textit{pba} becomes larger when $K$ increases, and reaches about 8% of the worst externality.

K Reproducibility Checklist

K.1 Algorithm

If the paper introduces a new algorithm, it must include a conceptual outline and/or pseudocode of the algorithm for the paper to be classified as CONVINCING or CREDIBLE. (CONVINCING)

K.2 Theoretical contribution

If the paper makes a theoretical contribution:

1. All assumptions and restrictions are stated clearly and formally (yes)
2. All novel claims are stated formally (e.g., in theorem statements) (yes)
3. Appropriate citations to theoretical tools used are given (yes)
4. Proof sketches or intuitions are given for complex and/or novel results (yes)
5. Proofs of all novel claims are included (yes)

For a paper to be classified as CREDIBLE or better, we expect that at least 1. and 2. can be answered affirmatively, for CONVINCING, all 5 should be answered with YES. (CONVINCING)

K.3 Data sets
If the paper relies on one or more data sets:

1. All novel datasets introduced in this paper are included in a data appendix (NA)
2. All novel datasets introduced in this paper will be made publicly available upon publication of the paper (NA)
3. All datasets drawn from the existing literature (potentially including authors’ own previously published work) are accompanied by appropriate citations (yes)
4. All datasets drawn from the existing literature (potentially including authors’ own previously published work) are publicly available (yes)
5. All datasets that are not publicly available (especially proprietary datasets) are described in detail (NA)

Papers can be qualified as CREDIBLE if at least 3., 4., and 5. can be answered affirmatively, CONVINCING if all points can be answered with YES. (CONVINCING)

K.4 Experiments
If the paper includes computational experiments:

1. All code required for conducting experiments is included in a code appendix (yes)
2. All code required for conducting experiments will be made publicly available upon publication of the paper (yes)
3. Some code required for conducting experiments cannot be made available because of reasons reported in the paper or the appendix (NA)
4. This paper states the number and range of values tried per (hyper-)parameter during development of the paper, along with the criterion used for selecting the final parameter setting (yes)
5. This paper lists all final (hyper-)parameters used for each model/algorithm in the experiments reported in the paper (yes)

6. In the case of run-time critical experiments, the paper clearly describes the computing infrastructure in which they have been obtained (yes, in Appendix I)

For CREDIBLE reproducibility, we expect that sufficient details about the experimental setup are provided, so that the experiments can be repeated provided algorithm and data availability (3., 5., 6.), for CONVINCING reproducibility, we also expect that not only the final results but also the experimental environment in which these results have been obtained is accessible (1., 2., 4.). (CONVINCING)