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 percentage points 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 Rönnqvist, 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 intersec-

An extended version including technical appendices is available at https://arxiv.org/abs/2205.00877.

tion 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 assumptions); 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 percentage points greater than the optimal solution, while being 41 percentage points 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]. Due to space limitations, all proofs of theorems are provided in the extended version of the paper.

2.1 Coordinated Management of Multiple Intersections

We assume finite time steps $\mathcal{T} = \{0, \dots, T\}$, intersections $\mathcal{I} = \{1, \dots, I\}$ and cars $\mathcal{K} = \{1, \dots, K\}$. The road network is represented by a directed graph $\mathcal{G} = \langle \mathcal{I}, \mathcal{E} \rangle$, 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.[1] $\mathcal{E}_i^{out} = \{(i,j) \in \mathcal{E}\}$ and $\mathcal{E}_i^{in} = \{(j,i) \in \mathcal{E}\}$ are the set of all outgoing and incoming edges of intersection i, respectively. Then, the set of edges connected to i is $\mathcal{E}_i = \mathcal{E}_i^{out} \cup \mathcal{E}_i^{in}$.

The goal of CMMI is to minimize total travel time, or equivalently, total waiting time. It involves computing an allocation, i.e. a time slot for each car to pass each intersection on its route. Let $\mathcal{A}_{(i,k)} \subseteq \mathcal{T} \cup \{\emptyset\}$ denote a set of possible allocations for car k at intersection i, where $\emptyset$ means the car is not yet allocated any time slot. Since we focus on the externality between intersections, we do not model the detail of the behavior inside the intersections. Then, allocation $a_{(i,k)} \in \mathcal{A}_{(i,k)}$ means that car k leaves its incoming edge to intersection i and immediately enters the outgoing edge at $a_{(i,k)}$. Each car $k \in \mathcal{K}$ has a fixed route $r_k = (i_1^k, \dots, i_\Omega^k)$ and a departure time T_k^o, 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 be allocated as $r_k^- = r_k \setminus \{i_\Omega^k\}$. 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 $\mathcal{A} = \prod_{(i,k) \in \mathcal{IK}} \mathcal{A}_{(i,k)}$. We also denote the previous intersection of $j = i_l^k$ by $\gamma(j,k) = i_{l-1}^k$, where $l > 1$. The waiting time of k at i is given by $w_{(i,k)} = a_{(i,k)} - (a_{(j,k)} + L_{(j,i)})$, where $a_{(i,k)} \in \mathcal{A}_{(i,k)}$, $a_{(j,k)} \in \mathcal{A}_{(j,k)}$ and $j = \gamma(i,k)$. If either $a_{(i,k)}$ or $a_{(j,k)}$ is $\emptyset$, $w_{(i,k)} = 0$. Then, the objective of CMMI is to minimize the total waiting time or total delay $D(a) = \sum_{k \in \mathcal{K}} D_k(a)$, where $D_k(a) = \sum_{i \in r_k} w_{(i,k)}$.

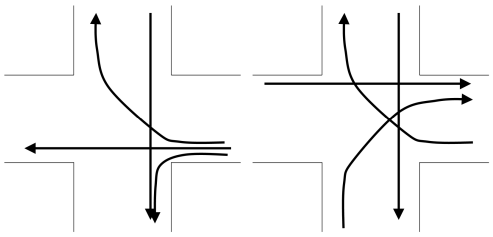

Figure 1: Examples of crossing path set $x_i \in \mathcal{X}_i$ in a typical 4-way intersection. In either case, every pair of the paths in the set is conflicting each other.

A car can collide with other cars when their paths are crossing in an intersection. A path in an intersection i is a pair of an incoming edge (j,i) and an outgoing edge (i,l). We denote the set of all paths passing the intersection by $\mathcal{P}_i = \{((j,i),(i,l)) | (j,i),(i,l) \in \mathcal{E}_i\}$. A set of paths crossing each other can be denoted by a subset $x_i \subseteq \mathcal{P}_i$ (Figure 1). Then, we denote the set of all crossing path sets in i by $\mathcal{X}_i \subseteq 2^{\mathcal{P}_i}$. Let $p_{(i,k)} \in \mathcal{P}_i$ denote the path of k in i. Then, cars k and k' collide at intersection i when their paths are crossing ($p_{(i,k)}, p_{(i,k')} \in x_i$ and $x_i \in \mathcal{X}_i$) and they are allocated to the same time slot ($a_{(i,k)} = a_{(i,k')}$). Also, we denote by $(\perp, (i_1^k, i_2^k))$ a special path that indicates car k departs from i_1^k to i_2^k, and this path crosses any path that also moves to (i_1^k, i_2^k) in the end.

Given these definitions, the CMMI problem is formulated as follows:

$$\min_{a \in \mathcal{A}_{\text{feas}}} D(a) \tag{1}$$

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

$$\mathcal{A}_{\text{feas}} = \{a \in \mathcal{A} |$$

$$h_{\text{wb}}^{(i,k)} : w_{(i,k)} \in [0, T_{UB}],\ \forall (i,k) \in \mathcal{IK}, \tag{2}$$

$$h_{\text{lcap}}^{(t,e)} : K_e^t \leq K_{UB},\ \forall t \in \mathcal{T}, \forall e \in \mathcal{E}, \tag{3}$$

$$h_{\text{dep}}^{k} : a_{(i_1^k,k)} \geq T_k^o,\ \forall k \in \mathcal{K}, \tag{4}$$

$$h_{\text{arr}}^{k} : a_{(l,k)} + L_{(l,i_\Omega^k)} \leq T, l = \gamma(i_\Omega^k, k), \forall k \in \mathcal{K}, \tag{5}$$

$$h_{\text{col}}^{i} : X_i = 0,\ \forall i \in \mathcal{I}\}, \tag{6}$$

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 $K_{(i,j)}^t = |\{(i,k) \in \mathcal{IK} | a_{(i,k)} \leq t\}| - |\{(j,k) \in \mathcal{IK} | a_{(j,k)} \leq t\}|$ 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 $X_i = |\{((i,k),(i,k') \in \mathcal{IK}) | k \neq$

[1] This could be implemented by overtaking or filtering into appropriate lanes. If this is not possible, it is easy to enforce FIFO by introducing additional DCOP constraints.

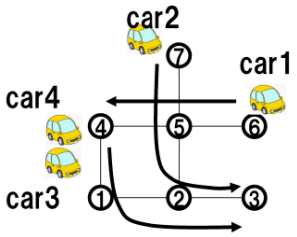

Figure 2: Example of CMMI where a cascade of braking can happen due to the externality between multiple intersections: $T_1^o = T_2^o = 0, T_3^o = 1, T_4^o = 2$ and $L_e = 5$ for all e. For simplicity, all paths at each intersection are assumed to be crossing (i.e., only one car on each intersection at a time is allowed).

$k', \exists x_i \in \mathcal{X}_i, p_{(i,k)}, p_{(i,k')} \in x_i, a_{(i,k)} = a_{(i,k')}\}|$ is the number of collisions at i. We also provide a MILP formulation of CMMI for linear solvers in the extended version. 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 $\mathcal{A}$ satisfy constraints that bound the waiting time, $h_{\text{wb}}, h_{\text{lcap}}$ and h_{arr}.

This is proved by polynomial-time reduction from an NP-hard job shop scheduling problem to CMMI.

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 $\langle \mathcal{N}, \mathcal{A}, u \rangle$ corresponding to DCOP $\langle \mathcal{N}, \mathcal{A}, \mathcal{C} \rangle$, where $\mathcal{N}$ is a set of agents, $\mathcal{A}_j$ is the action space of agent j, $\mathcal{A} = \prod_{j \in \mathcal{N}} \mathcal{A}_j$ is a joint action space of the agents and $\mathcal{C} = \{c_1, c_2, \ldots\}$ is a set of constraints. A constraint $c = (\mathcal{N}_c, f_c) \in \mathcal{C}$ is defined using $f_c : \prod_{j \in \mathcal{N}_c} \mathcal{A}_j \to \mathbb{R}_+$, which is a cost function of the actions by a group of agents, or *neighborhood*, $\mathcal{N}_c \subseteq \mathcal{N}$.

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 $a = (a_1, \ldots, a_n) \in \mathcal{A}$ and we denote a projection of the solution for constraint c as $a^c = (a_j | j \in \mathcal{N}_c)$. We sometimes denote $f_c(a) = f_c(a^c)$ and also denote $a_{-j} \in \mathcal{A}_{-j} = \prod_{l \neq j \in \mathcal{N}} \mathcal{A}_l$. $u = (u_1, \ldots, u_n)$ is a list of utility functions of the agents. Each agent $j \in \mathcal{N}$ is involved in a set of constraints $\mathcal{C}_j = \{c | j \in \mathcal{N}_c\}$, and has a utility function $u_j(a) = -\sum_{c \in \mathcal{C}_j} f_c(a^c)$. The social welfare is defined as $J(a) = \sum_{j \in \mathcal{N}} u_i(a)$. 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, $\forall j \in \mathcal{N}, \forall a'_j \in \mathcal{A}_j$, and $\forall a_{-j} \in \mathcal{A}_{-j}$:

$$u_j(a_j, a_{-j}) \geq u_j(a'_j, a_{-j}). \tag{7}$$

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 ($\mathcal{N} = \mathcal{I}$), car agents ($\mathcal{N} = \mathcal{K}$) and atomic agents ($\mathcal{N} = \mathcal{IK}$). The action space of atomic agent (i, k) is the same $\mathcal{A}_{(i,k)}$ defined in Section 2.1. The action space of intersection agent i and car agent k are defined as $\mathcal{A}_i = \prod_{k | i \in r_k^-} \mathcal{A}_{(i,k)}$ and $\mathcal{A}_k = \prod_{i \in r_k^-} \mathcal{A}_{(i,k)}$, respectively. Note that $\mathcal{A}_{(i,k)}$ is considerably smaller than $\mathcal{A}_i$ or $\mathcal{A}_k$.

We denote the index function as χ, where $\chi(True) = 1, \chi(False) = 0$. Also, let P_{fwd}, P_{col}, P_{none}, P_2 and P_1 denote the weight constants to prioritise 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 $h_{\text{wb}}^{(i,k)}$ corresponds to soft constraint $f_{\text{wb}}^{(i,k)}(a) = \chi(\neg h_{\text{wb}}^{(i,k)}) * P_1$, with neighbor $\mathcal{N}_{\text{wb}}^{(i,k)} = \{(\gamma(i,k), k), (i,k)\}$ that includes agents involved in $f_{\text{wb}}^{(i,k)}(a)$. 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 $\mathcal{N}_c$ 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 the extended version.

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 $u_j(a) = -\sum_{c \in \mathcal{C}_j} f_c(a) = f_{\text{delay}}(a)$. In case of $\mathcal{N} = \mathcal{K}$, $f_{\text{delay}}(a) = \sum_{i \in r_k} w_{(i,k)}$. Then, $J(a) = \sum_{k \in \mathcal{K}} u_j(a) = -\sum_{k \in \mathcal{K}} \sum_{i \in r_k} w_{(i,k)} = -D(a)$. □

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 $\mathcal{N}_{\text{delay}}$ contains two agents, $(\gamma(i,k), k)$ and (i,k), and their waiting times are double counted when computing the utlities 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 $\mathcal{C} \setminus \{c_{\text{delay}}\}$ is zero. Also, $\mathcal{A}_{\text{feas}}$ is the same for all types of agents because if the total

Algorithm 1 Best response update with downstream reset

```
1: procedure a =BRUDR(a, bestU)
2:     while a violates (7) do
3:         Δ̂ = 0, â = ∅
4:         for j ∈ N do
5:             for a'_j ∈ A_j do
6:                 a'' = DownReset(j, a, (a'_j, a_{-j}))
7:                 Δu_j = u_j(a'') − u_j(a)
8:                 if Δu_j > 0 then
9:                     if not bestU then
10:                        a = a''
11:                        break
12:                    else if Δu_j > Δ̂ then
13:                        Δ̂ = Δu_j, â = a''
14:        if bestU and â ≠ None then
15:            a = â
16:    Return a
```

penalty is zero for a type of the agents, it remains zero if we change the way of aggregation.

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 the extended version.

3.1 Polynomial Time Feasible Algorithm

To address the challenges above, we propose a novel algorithm for CMMI 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:

$$P_{\text{fwd}} \gg P_{\text{col}} \gg P_{\text{none}} \gg P_2 \gg P_1 \gg T. \tag{8}$$

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_{wb}. We initialize the joint action as being unallocated, $a = \emptyset$, which means $a_{(i,k)} = \emptyset, \forall (i,k) \in \mathcal{I} \times \mathcal{K}$.

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_{wb} 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 $\mathcal{A}$ satisfy $h_{\text{wb}}, h_{\text{lcap}}$ and h_{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, $h_{\text{wb}}, h_{\text{lcap}}$ and h_{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 $\mathcal{A}$ satisfy $h_{\text{wb}}, h_{\text{lcap}}$ and h_{arr}.

Proof. This follows from Lemma 2. Since all agents update

at most once, the number of the iterations in BRUDR is less than $T * I * K$. The size of the action space is polynomial in $I * T$ multiplied by the complexity of calculating f_c. As in the definition of constraints, the complexity of calculating $f_c, \forall c \in \mathcal{C}$ is also polynomial. □

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 $h_{\text{wb}}, h_{\text{lcap}}$ and h_{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 the extended version.

Despite of the scalability above, the efficiency of the solution can be arbitrarily worse than the optimal one. We discuss the detail in the extended version, 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 $D(a_{ne}) = 3$. Since we can add more cars after car 4, $D(a_{ne})$ 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).

$$a_j = \underset{a'_j \in \mathcal{A}_j}{\operatorname{argmax}}\, u_j(a'_j, \emptyset_{-j}), \forall j \in \mathcal{N}, \tag{9}$$

where $\emptyset_{-j}$ means $a_l = \emptyset, \forall l \in -j$. 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 $\mathcal{F} = \mathcal{F}_1 \sqcup \mathcal{F}_2$. 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] [2]. 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 $default$ and 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 pba and $default$ 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.

4.1 Small Settings

We compare the performance of BRUDR with an optimal solution by changing the definitions of agents ($\mathcal{K}$, $\mathcal{I}$ and $\mathcal{I} \times \mathcal{K}$), 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 pba becomes smaller than the one of $default$ 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 $\mathcal{F}_1$ is

[2] We thank EPFL Urban Transport Systems Laboratory for the data source: pNEUMA – open-traffic.epfl.ch

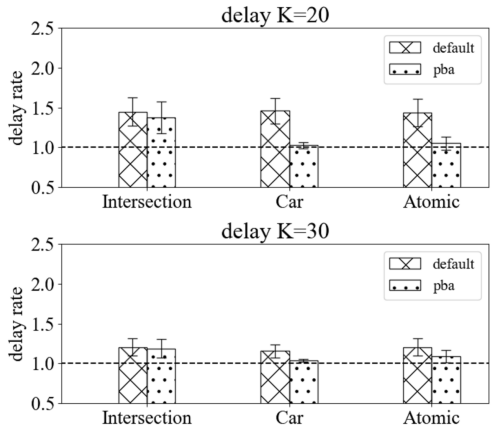

Figure 3: The ratio of total delay compared to the optimum one $D(a)/D(a_{so})$. Comparisons between $default$ and pba with different types of agents and density of cars K/I.

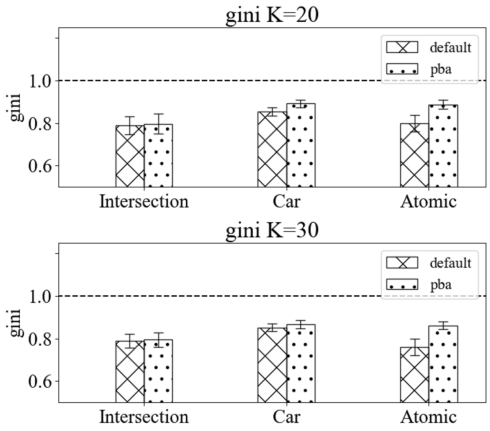

Figure 4: Gini index $GI(a) = \sum_{k\in\mathcal{K}}\sum_{k'\in\mathcal{K}}|D_k(a) - D_{k'}(a)|/2K|D(a)|$. Comparisons between $default$ and pba with different types of agents and density of cars K/I.

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 percentage points reduction on average, while it is 2.6 percentage 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 Figure 2, 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 the extended version.

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 Figure 2, 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_{lcap}, 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_{lcap} is included. However, if we can assume the traffic is not so

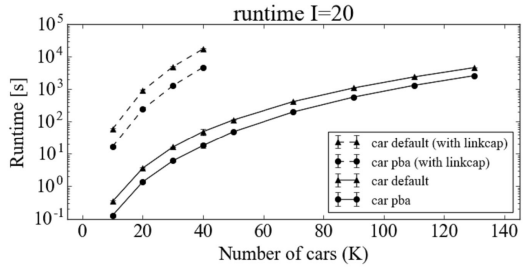

Figure 5: Runtime with different number of cars K. Comparisons between $default$ and pba, with and without c_{lcap} computation.

dense and can omit the computation of c_{lcap}, 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.

Ethical 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 treated unfairly 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 optimisation 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.

[Fabrikant *et al.*, 2004] Alex Fabrikant, 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.

[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.

[Pearce *et al.*, 2005] Jonathan P Pearce, Rajiv T Maheswaran, and Milind Tambe. Local algorithms for distributed constraint optimization in dynamic, anytime environments. In *AmbiAgents Workshop, AAMAS*. Citeseer, 2005.

[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.