Safe Multi-agent Learning via Trapping Regions

Aleksander Czechowski, Frans A. Oliehoek
Delft University of Technology

Abstract
One of the main challenges of multi-agent learning lies in establishing convergence of the algorithms, as, in general, a collection of individual, self-serving agents is not guaranteed to converge with their joint policy, when learning concurrently. This is in stark contrast to most single-agent environments, and sets a prohibitive barrier for deployment in practical applications, as it induces uncertainty in long term behavior of the system. In this work, we apply the concept of trapping regions, known from qualitative theory of dynamical systems, to create safety sets in the joint strategy space for decentralized learning. We propose a binary partitioning algorithm for verification that candidate sets form trapping regions in systems with known learning dynamics, and a heuristic sampling algorithm for scenarios where learning dynamics are not known. We demonstrate the applications to a regularized version of Dirac Generative Adversarial Network, a four-intersection traffic control scenario run in a state of the art open-source microscopic traffic simulator SUMO, and a mathematical model of economic competition.

1 Introduction
In the recent years, enormous progress has been made for single agent planning and learning algorithms, with agents matching or exceeding human performance in various tasks and games [Mnih et al., 2013; Silver et al., 2016]. The vast success of single agent learning can be partially explained by robustness and strong convergence properties of the underlying algorithms in their basic form, such as Q-learning [Watkins and Dayan, 1992] or policy gradients [Sutton et al., 1999]. Despite wide interest, the same cannot be however said for multi-agent learning. Even most basic models, e.g. replicator learning for normal form games, exhibit nonconvergence, cyclic or even chaotic behavior [Sato et al., 2002]. Even worse, it has been shown that in decoupled learning systems, there can be no learning rule that guarantees convergence to a Nash equilibrium [Hart and Mas-Colell, 2003]. These nonconvergent examples have also been found in more practical learning problems, such as Generative Adversarial Networks [Mescheder et al., 2018]. Due to the above limitations, many successful multi-agent learning methods resorted to using a centralized component, such as centralized critic algorithms usually rely on specific assumptions on the reward structure for convergence, for instance fictitious play or exploitability descent in zero-sum games [Brown, 1951; Lockhart et al., 2019].

Despite this undisputed progress in designing convergent multi-agent algorithms, it can be argued that in practical, real-world multi-agent scenarios there will be plenty of situations, where convergence cannot be enforced. One can easily envisage a situation, e.g. in automated traffic control and driving scenarios, or in automated trading, where multiple entities (be it traffic lights, vehicles, or brokers) follow own learning protocols for individual reward maximization. Such learning rules, even though designed to be convergent in static, single-agent environments, would invariably interfere with one another in a multi-agent setting, sometimes resulting in cyclic, or divergent outcomes. The lack of convergence guarantees in such general settings forms a major obstacle for introduction of online learning systems in practical applications, as it introduces a lot of uncertainty over what will be the state of the system, if learning is left unsupervised. Can we nevertheless still establish a type of safety certificates, that would allow us to conclude that simultaneous learning will not spin out of control?

In this paper, we suggest a novel approach to address issue. We start from the realization, that convergence is often not absolutely necessary for reliability. From systems designers perspective, it is often enough to know that learning has rough stability guarantees – that is, that agents will not leave a predetermined region of the strategy space during learning.

Due to space restrictions, the proofs of our theorems and lemmas have been postponed to the Supplementary Material, and can be accessed in the extended version of the paper [Czechowski and Oliehoek, 2023]. There, we also provide an additional example, where we find trapping regions in a model of economic competition, which ensures that none of the competing companies will reduce their production to zero.
For a conceptual application, let us consider a traffic light control network as in Figure 3, where individual traffic light controllers learn the best balance of green time between the phases to minimize the waiting time for approaching vehicles. It is not absolutely necessary that by learning each intersection reaches a final, static setting, but would be essential that at all times it gives minimal green time of at least few seconds to each phase, to make sure that no vehicle gets stuck indefinitely on a red traffic light, and also serve the lingering pedestrian flows.

We propose a method of a priori verifying these constraints, by establishing trapping regions; regions of strategy space, which learning trajectories will never escape. The idea behind this concept is simple: a candidate set for a trapping region is formed by the constraints imposed by practical, problem-dependent safety considerations. By verifying whether such set is forward-invariant for the joint learning operator, we obtain a yes–or–no answer on whether it is safe to allow multi-agent learning (possibly in a decentralized manner), without breaking these constraints. This method can be seen an alternative solution concept in systems, where Nash equilibria are difficult or impossible to reach by learning dynamics. Trapping regions are intended to be used as a safety prerequisite. For instance a road authority could pre-approve the algorithms of automated road users, by checking whether their joint policy forms trapping regions – before they are deployed in real life.

This paper is organized as follows. Section 2 introduces the setting and necessary preliminaries. In Section 3 we present the definition of a trapping region, prove several useful theorems and lemmas that are useful for their verification for Lipschitz-continuous learning, and present two algorithms, for verifying whether given hyperrectangular sets forms a trapping region, only from the knowledge of learning operator on the set boundary. The first algorithm, is based on binary partitioning, and is applicable when learning dynamics are known analytically and Lipschitz, and we would like to have a mathematically rigorous guarantee. The second one is a heuristic algorithm, applicable in scenarios where learning dynamics can only be sampled, its dependability is directly correlated to the number of boundary samples taken.

Finally, in Section 4 we introduce two examples, that illustrate the applications of trapping regions. The first of our first of our examples is a toy problem, a simple GAN-like learning scenario, where gradient learning starting from almost all points never converges, whereas trapping regions are abundant and easy to find. In our second example we deal with a practical traffic control problem, where four intersection in a traffic network adjust their strategies to dispatch traffic in an optimal manner. For this problem, we construct and verify a trapping region, which ensures all traffic directions will be given enough time, when traffic controllers are left to learn unsupervised.

1.1 Related Work
Trapping regions are well known and standard tool in qualitative theory of dynamical systems, e.g. [Meiss, 2007; Bonatti, 2006], but to the best of our knowledge have not been directly applied in learning and control scenarios. The majority of work on safety guarantees in control theory focuses on so-called constrained optimization [Altman, 1999]. In the context of safe reinforcement learning, the focus has been on designing algorithms that satisfy particular safety constraints, c.f. [Garcia and Fernández, 2015] and references therein. In the multi-agent case, research has been directed towards methods where an orchestrator [ElSayed-Aly et al., 2021] or agents individually [Lu et al., 2021] are adapting their behavior to respect the constraints; this has also been the underlying philosophy in the method of barrier functions [Wills and Heath, 2004; Yang et al., 2020]. There are also strong connections to methods of formal verification methods, in particular ones based on reachability analysis [Ruan et al., 2018; Wang et al., 2021].

Relation to Lyapunov Control. Our method shares most similarities with the ones based on Lyapunov functions, such as Neural Lyapunov Control [Chang et al., 2019], see also [Berkenkamp et al., 2017]. There are however several key differences that we would like to highlight here. Most importantly, Lyapunov-based methods are only applicable to (locally) convergent scenarios, as the existence of a Lyapunov function implies the existence of a locally attracting equilibrium of the system. On the other hand, trapping regions are very well suited to deal with problems, where learning never converges to a stationary solution. Even if the learning trajectory does not converge, the bounds provided by the trapping region will ensure that they never diverge into unsafe regions of the joint strategy space. This has strong practical consequences: a non-convergent learning process without safety guarantees would have to be indefinitely supervised, whereas trapping regions ensure that it only explores safe parts of the strategy space, and does not require supervision. Dealing with non-stationarity is particularly important for multi-agent systems, as it was shown e.g. in [Kleinberg et al., 2011] outcomes of a non-convergent cyclic learning process can lead to higher social welfare than these of a stationary Nash; and even worse, often stationarity cannot be ensured, as some learners can be outside of our control (e.g. in adversarial scenarios). In Section 4.1, we provide a low-dimensional example where none of the learning trajectories converge to the equilibrium point, but trapping regions are easy to find, c.f. Figure 2.

The second difference comes in computational complexity. Neural Lyapunov Control requires evaluation of learning directions over a whole domain, while we only require it on a boundary of a domain, effectively reducing the dimension of the verification problem by one (c.f. Lemma 1).

2 Preliminaries
We consider decentralized learning schemes for groups of $n$ agents that can be represented compactly by discrete adaptive dynamics of the form:

$$
x_{i+1}^1 := x_i^1 + \gamma F_1(x_{i}^1, \ldots, x_{i}^n),
$$

$$
\vdots
$$

$$
x_{i+1}^n := x_i^n + \gamma F_n(x_{i}^1, \ldots, x_{i}^n),
$$

where $x_i \in X_i \subset \mathbb{R}^{k_i}$ represents a point in the strategy space of a given agent $i$ (e.g. weights in a neural network or ratios
We remark that an equilibrium for the learning system (3) is characterized by the property that learning trajectories that begin inside of it. This is a standard argument, more commonly known in the continuous case (e.g. [Meiss, 2007]), and is formalized in Lemma 1. The following theorem, a folklore in dynamical systems community, highlights the advantage of establishing trapping regions; a trapping region not only guarantees that the learning curves starting inside can never escape it, but also that there is a learning equilibrium (possibly a Nash equilibrium) inside of it.

**Theorem 1.** Let $T$ be a trapping region. Then

1. Any learning trajectory (3) that starts in $T$ never leaves $T$.
2. If $T$ is convex, then there exists a learning equilibrium $x^* \in \text{int } T$.

### 3 Algorithmic Verification: Explicit Learning Dynamics

In practice, verification of condition (4) can be troublesome, as the volume of the trapping region usually requires a prohibitively high amount of samples. For small learning rates and continuous learning dynamics, it is however enough to verify this assumption on the boundary, as any trajectory that could leave the region would have to pass through the boundary area. This is a standard argument, more commonly known in the continuous case (e.g. [Meiss, 2007]), and is formalized for our discrete setting in Lemma 1.

**Lemma 1.** Given a compact set $T$, if $\gamma > 0$ is sufficiently small, and for all $x \in \partial T$ we have

$$x + \gamma F(x) \in \text{int } T,$$

then $T$ is a trapping region.

Lemma 1 can be used to derive exact inequalities needed to be satisfied by the learning operators, which are sufficient to establish a trapping region. In what follows, we denote by $x^i = [x^{i1}, \ldots, x^{ik_i}]^T$ and $F_i = [F_{i1}, \ldots, F_{ik_i}]^T$ the components of strategies and learning operators for each agent $i \in 1, \ldots, n$. In the examples we will sometimes omit the second subscript, when the strategy space of each agent is one-dimensional.

**Definition 2.** Let $T$ be a set of the form of a product of intervals

$$T := [x_{-1}^{n1}, x_{+1}^{n1}] \times \cdots \times [x_{-nk_n}^{nk_n}, x_{+nk_n}^{nk_n}] \subset \mathbb{R}^N.$$
For $i \in 1, \ldots, n$, $j \in 1, \ldots, k_i$, we denote by $T_{ij}^l$ the set of all points $x \in T$, such that $\pi_{ij} - \text{the projection onto i-th agents j-th component satisfies } \pi_{ij}x = x_{ij}$. We call this set the $(ij)$th left face of $T$. Similarly, we denote by $T_{ij}^r$ the set of all points $x \in T$, such that $\pi_{ij}x = x_{ij}^r$, and call it the $(ij)$th right face of $T$.

Our next Lemma follows directly from Lemma 1 applied to a trapping region of form of a product of intervals.

**Lemma 2.** Given a set $T \in \mathbb{R}^N$ which is a product of intervals, assume that the following isolation inequalities are satisfied:

\[
F_{ij}(x) > 0, \quad \forall x \in T_{ij}^l,
\]

\[
F_{ij}(x) < 0, \quad \forall x \in T_{ij}^r.
\]

Then, the set $T$ is a trapping region for $\gamma > 0$ sufficiently small.

For Lipschitz-continuous learning dynamics, and trapping regions of form of a product of intervals, explicit bound on the range of $\gamma$ can be given.

**Theorem 2.** Let $T$ be as in Lemma 2 and $F$ be Lipschitz-continuous with Lipschitz constant over $T$ bounded from above by $L$. The upper bound on step size $\gamma$ for which $T$ forms a trapping region in the learning system (1) can be given explicitly by

\[
\gamma < \frac{\min_{p \in \{l,r\}} \min_{ij} \min_{x \in T_{ij}^p} |F_{ij}(x)|}{L \max_{x \in T} ||F(x)||_{\max}}.
\]

This theorem provides a method to verify trapping regions for Lipschitz-continuous learning dynamics.

**Remark 1.** For sufficiently regular boundaries $\partial T$, conditions (7) can be generalized to situations, where $T$ is not a product of intervals. Namely, for $T$ to be a trapping region it is enough that

\[
\langle F(x), n_{\partial T}(x) \rangle < 0, \quad \forall x \in \partial T,
\]

where $n_{\partial T}(x)$ is the normal vector to $\partial T$, pointing in direction outwards of $T$, c.f. [Meiss, 2007].

The visualization of assumption (7) from Lemma 2 is presented in Figure 1; the intuition behind it is that the learning operators $F_{ij}$ have to point inwards, into the trapping region, so their values have to be positive on left faces and negative on right faces. When the adaptive dynamics are not given explicitly (e.g. they depend on a reward from environment simulator), one may need to resort to verifying condition (7) approximately, by evaluating the learning dynamics $F_{ij}$ on a finite subset of points, which provide good enough coverage of faces of $T$. If some analytical knowledge on learning dynamics is available, we can verify (7) rigorously (with sufficient numerical precision). For instance, assume that we know the upper bound for the Lipschitz constant of $F$ over $T$, given by $L$. Our verification is based on the following observation. We will check whether

\[
\pm F_{ij}(x) > 0, \quad x \in S,
\]

where $S$ denotes either of the faces $T_{ij}^l$, $T_{ij}^r$, respectively, or their hyperrectangular subsets. Then it is enough to verify that either

\[
\mp F_{ij}(C(S)) + L \text{diam}(S)/2 < 0,
\]

where $C(S)$ is the baricenter (i.e. the centroid / intersection of diagonals) of $S$. Alternatively, we can show that

\[
\pm F_{ij}(C(S)) \leq 0,
\]

which will prove that the candidate $T$ is not a trapping region.

If $S$ is the whole face of $T$ (i.e. $T_{ij}^l$ or $T_{ij}^r$ for some $i, j$), then the verification of inequality (11) can fail, even despite that $T$ is a trapping region. Therefore, we propose to adopt *binary space partitioning* mechanism [Fuchs et al., 1980] to iteratively subdivide faces of $T$ into smaller hyperrectangles, until inequalities fail, or all hyperrectangles have been verified. For details, we refer to the pseudocode in Algorithm 1. The function SPLIT in Algorithm 1 splits a hyperrectangle $S$ into two disjoint non-empty hyperrectangles $S_1, S_2$, such that $S = S_1 \cup S_2$ in half, along the longest dimension of the hyperrectangle.

**Theorem 3.** If $T$ is a trapping region, Algorithm 1 is guaranteed to terminate in finite steps. Without loss of generality, assume that $T$ is a unit hypercube. Then, the computational complexity of the algorithm is $O \left( \log(L/2m^*) \sum_{i=1}^k \sum_{j=1}^n k_i \right)$, where

\[
m^* = \min_{ij} \min_{x \in T_{ij}^l} |F_{ij}(x)|.
\]

Conversely, if Algorithm 1 terminates and returns true, $T$ is a trapping region for learning rates as in Theorem 2.

### 3.3 Algorithmic Verification: Sampled Learning Dynamics

In some situations, the exact learning dynamics are not available – e.g. they depend on a reward, which can only be obtained from a real world or an experiment. Then, one has to resort to heuristic verification of trapping regions, by sampling points from the faces of the interval set. We provide the pseudocode for this situation in Algorithm 2, and apply it in practice in traffic management example in Section 4.2.
Let $\mathbf{T}$ be the set of all sampled points, $D$ be the size of mesh generated by the sample, i.e.

$$D = \sup_{i,j,x \in \mathbf{T}_{i,j}} \min_{x'' \in S^*} ||x - x''||. \quad (14)$$

Also let

$$m^* = \min_{i,j,x \in S^*} ||F_{ij}(x^*)|| \quad (15)$$

quantify how close we were to fail verifying isolation over $S^*$. If Algorithm 2 returns true and $F$ is Lipschitz-continuous with Lipschitz constant $L < m^*/D$, then $\mathbf{T}$ is a trapping region for learning rates as in Theorem 2.

4 Examples

In this Section we will provide examples of application of Algorithm 1 to two systems with known dynamics – a Generative Adversarial Network in Subsection 4.1 and of application of Algorithm 2 to a traffic learning system with dynamics provided by the system simulator in Subsection 4.2. Additional example in a model of economic competition is provided in the Supplementary Material.

4.1 Generative Adversarial Learning

Our first example is a system, which exemplifies the issue of non-convergence of multi-agent learning, but where trapping regions can be readily constructed. Since the learning is non-convergent, methods based on Lyapunov functions, and regions of attraction of equilibria would not be applicable to this scenario. We consider a parameterized family of learning systems, where the parameter controls the coupling between learner rewards – from completely decoupled, to strongly coupled. More concretely, agent one is in control of a continuous variable $\psi \in \mathbb{R}$, and agent two controls $\theta \in \mathbb{R}$. The rewards of each agent are the negative of the loss functions for each, and these are given by

$$L_1(\psi, \theta) = \psi^4 + \epsilon \psi \theta, \quad (16)$$

and

$$L_2(\psi, \theta) = \theta^4 - \epsilon \psi \theta, \quad (17)$$

for some positive, small $\epsilon$.

Both agents use gradient descent on their respective loss functions, with a fixed step $\gamma$, which leads to following update rules

$$\psi_{t+1} := \psi_t - \gamma (4 \psi_t^3 + \epsilon \theta_t),$$

$$\theta_{t+1} := \theta_t - \gamma (4 \theta_t^3 - \epsilon \psi_t), \quad (18)$$

which, for short, we shall denote by $(\psi_{t+1}, \theta_{t+1}) =: G(\psi_t, \theta_t)$. 

---

Algorithm 1: Rigorous trapping region verification via binary space partitioning.

**Inputs:** Learning dynamics $F$, $\mathbf{T} = [x_{-1}^{11}, x_{+1}^{11}] \times \cdots \times [x_{-nk}^{nk}, x_{+nk}^{nk}]$ – a candidate for the trapping region, $L$ – upper bound for Lipschitz constant of $F$ over $\mathbf{T}$.

**Returns:** Is $\mathbf{T}$ a trapping region?

**Start:**

1: for agent $i$ in 1:n in parallel do
2: for coordinate $j$ in 1:k in parallel do
3: for direction in {left,right} in parallel do
4: if direction is left then
5: $\text{SETS_TO_CHECK} = \{T_{ij}^l\}$, $\delta = -1$
6: else
7: $\text{SETS_TO_CHECK} = \{T_{ij}^r\}$, $\delta = 1$
8: while $\text{SETS_TO_CHECK} \neq \emptyset$ do
9: $S = \text{SETS_TO_CHECK}.\text{POP}()$
10: $C(S) = \text{baricenter}(S)$
11: if $\delta F_{ij}(C(S)) \geq 0$ then
12: return false // no isolation
13: else if $\delta F_{ij}(C(S)) + L\text{diam}(S)/2 \geq 0$ then
14: // need subdivision to check isolation
15: $S_1, S_2 = \text{SPLIT}(S)$ // binary partitioning
16: $\text{SETS_TO_CHECK}.\text{PUSH}(S_1, S_2)$
17: return true

---

Algorithm 2: Non-rigorous trapping region verification via sampling.

**Inputs:** Learning dynamics $F$, $F$ – learning dynamics, can be only sampled (e.g. from simulator), $\mathbf{T} = [x_{-1}^{11}, x_{+1}^{11}] \times \cdots \times [x_{-nk}^{nk}, x_{+nk}^{nk}]$ – a candidate for the trapping region, $M$ – sample size per face.

**Returns:** Is $\mathbf{T}$ a trapping region?

**Start:**

1: for agent $i$ in 1:n in parallel do
2: for coordinate $j$ in 1:k in parallel do
3: for direction in {left,right} in parallel do
4: if direction is left then
5: $\text{SET} = T_{ij}^l$, $\delta = -1$
6: else
7: $\text{SET} = T_{ij}^r$, $\delta = 1$
8: // a uniformly spaced sample of $M$ points
9: $S = \text{SAMPLE_POINTS(SET, } M)$
10: for $x \in S$ in parallel do
11: if $\delta F_{ij}(x) \geq 0$ then
12: return false // no isolation
13: return true

---
Although a system with such prescribed loss functions is nothing more than a toy example, it serves to accentuate the problems of non-convergence. Similar learning systems have been thoroughly analyzed in literature; this system in fact has the same update rules as the famously non-convergent Dirac-GAN example in [Mescheder et al., 2018] with the Wasserstein loss function, where both the generator and the discriminator apply an $L^1$ regularization term weighted by factor inversely proportional to $\epsilon$.

The dynamics of (18) are surprisingly complicated for such low dimensional system. The system possesses a single Nash equilibrium $(\psi, \theta) = (0,0)$ (also the only learning equilibrium), regardless of the value of $\epsilon$. For joint optimization, the equilibrium is always locally unstable (regardless of how small the system coupling parameter $\epsilon$ is), and the learning trajectories starting from its near proximity diverge from it until they enter a cyclic regime. For initial conditions of larger norm, they converge towards the cyclic attractor, and never reach the equilibrium; in fact none of the other trajectories do. The divergence from the Nash equilibrium is formalized via the following proposition below (with $\| \cdot \|$ denoting $L^2$ norm):

**Proposition 2.** For any $\gamma > 0$ and any $\epsilon > 0$ there exist a value $R_0 > 0$, such that for any $(\psi_0, \theta_0)$ with $0 < \| (\psi_0, \theta_0) \| < R_0$ we have $\| G(\psi_0, \theta_0) \| > \| (\psi_0, \theta_0) \|$. As a consequence, $(0,0)$ is a repelling equilibrium.

On the other hand, it is easy to find trapping regions. We report that by executing Algorithm 1 we have successfully established existence of various trapping regions for different values of $\epsilon$, and computed $\gamma$ bounds via formula (8) (by making use of quantities obtained in the algorithm):

- $T = [-0.1, 0.1]^2$ and $\epsilon \in \{0.01, 0.02, 0.03, 0.04\}$ (we report failure, i.e. it is not a trapping region for $\epsilon = 0.05$) and the upper bounds on $\gamma$ are given by $\{4.6 \times 10^{-3}, 7.9 \times 10^{-3}, 1.7 \times 10^{-3}, 10^{-17}\}$, respectively;
- $T = [-0.2, 0.2]^2$ and $\epsilon \in \{0.05, 0.1, 0.15\}$ (failure for $\epsilon = 0.2$), and the upper bounds on $\gamma$ are given by $\{1.9 \times 10^{-2}, 4.6 \times 10^{-4}, 2.0 \times 10^{-4}\}$, respectively.

The Lipschitz constants in both examples were found analytically, by maximizing the $L^1$ norm of the total derivative $\| D(\psi, \theta) G \|$ over $(\psi, \theta) \in T$. We remark that the closer we got to the point of failure, the more subdivisions were needed in the partitioning algorithm, however the execution time was near immediate – a few seconds at most on a modern laptop, without leveraging parallelization. To contrast, a brute force optimization without verifying the trapping region would yield endless execution without convergence, and without any guarantees that learning will not diverge.

For this particular system, we can also prove the existence of an $\epsilon$-parameterized family of trapping regions theoretically, by the following proposition:

**Proposition 3.** The square given by $[-\sqrt{\epsilon}, \sqrt{\epsilon}]^2$ is a trapping region for step size $\gamma > 0$ small enough. As a consequence, trajectories never leave $[ -\sqrt{\epsilon}, \sqrt{\epsilon}]^2$, and there is an equilibrium inside $[-\sqrt{\epsilon}, \sqrt{\epsilon}]^2$ (it is in fact the global Nash equilibrium $(0,0)$).

### 4.2 Multi-Agent Traffic Management

Our second example is of a more practical nature. We analyze a rectangular network of four signalized intersections, each situated 200 meters from its two nearest neighbors, as depicted in Figure 3. Each of the intersections controls traffic by alternating between one of two phases – giving green to either the vertical or the horizontal stream of vehicles. The cycle time, i.e. the total time for serving the horizontal and, subsequently, the vertical movement is set to 60 seconds. For each episode of simulation, of length of two hours, each intersection can select a strategy from the continuous set $[0, 60]$ which determines the amount of green seconds to be assigned to the first phase (the offset). The remainder of the cycle is assigned to the second phase. The vehicle streams are generated on all roads in all directions (i.e. east ↔ west, and north ↔ south), and, for simplicity, we excluded left and right turning movements on intersections. The simulation is controlled by
The gradient of own reward by difference quotients:

Equations (1) and (2). Each intersection controller estimates the learners is performed via decoupled gradient descent, as in equations could have been performed for other traffic patterns. Vehicles are spawned every five seconds. Analogous computations could have been performed for other traffic patterns.

For our experiment, the selection of the strategy by the learners is performed via decoupled gradient descent, as in Equations (1) and (2). Each intersection controller estimates the gradient of own reward by difference quotients:

\[ \delta \nabla_x R_i(x) \approx R_i(x, x_{-i}) - R_i(x + \delta, x_{-i}) \]  

for some small \( \delta \) (in our experiments \( \delta = 0.1 \)). The adaptation rate \( \gamma \) is set to \( 10^{-6} \). Such settings were chosen as they would give satisfactory results for learning on one intersection, while keeping other intersections fixed.

As discussed previously, non-convergence is an undesirable learning effect in such traffic management scenario, as one would like to ensure that learning always stays within some predetermined bounds, so minimal green time can be given to vehicle flows and pedestrians within each cycle. As a reasonable prerequisite we assume that each phase should be given at least 20 seconds of green time, which translates to the candidate for a trapping region given by \( T = [20, 40]^4 \).

From the nature of the problem, we expect the reward function to be continuous, but we do not have an analytical formula for it. Therefore, we apply Algorithm 2 and sample the candidate for a trapping region given by \( T \). Learning curves, which begin on the boundary of the trapping region do not escape it, and evolve in the interior of the set.

For our experiment, the selection of the strategy by the learners is performed via decoupled gradient descent, as in Equations (1) and (2). Each intersection controller estimates the gradient of own reward by difference quotients:

\[ \delta \nabla_x R_i(x) \approx R_i(x, x_{-i}) - R_i(x + \delta, x_{-i}) \]  

for some small \( \delta \) (in our experiments \( \delta = 0.1 \)). The adaptation rate \( \gamma \) is set to \( 10^{-6} \). Such settings were chosen as they would give satisfactory results for learning on one intersection, while keeping other intersections fixed.

As discussed previously, non-convergence is an undesirable learning effect in such traffic management scenario, as one would like to ensure that learning always stays within some predetermined bounds, so minimal green time can be given to vehicle flows and pedestrians within each cycle. As a reasonable prerequisite we assume that each phase should be given at least 20 seconds of green time, which translates to the candidate for a trapping region given by \( T = [20, 40]^4 \).

From the nature of the problem, we expect the reward function to be continuous, but we do not have an analytical formula for it. Therefore, we apply Algorithm 2 and sample the candidate for a trapping region given by \( T \). Learning curves, which begin on the boundary of the trapping region do not escape it, and evolve in the interior of the set.

For our experiment, the selection of the strategy by the learners is performed via decoupled gradient descent, as in Equations (1) and (2). Each intersection controller estimates the gradient of own reward by difference quotients:

\[ \delta \nabla_x R_i(x) \approx R_i(x, x_{-i}) - R_i(x + \delta, x_{-i}) \]  

for some small \( \delta \) (in our experiments \( \delta = 0.1 \)). The adaptation rate \( \gamma \) is set to \( 10^{-6} \). Such settings were chosen as they would give satisfactory results for learning on one intersection, while keeping other intersections fixed.

As discussed previously, non-convergence is an undesirable learning effect in such traffic management scenario, as one would like to ensure that learning always stays within some predetermined bounds, so minimal green time can be given to vehicle flows and pedestrians within each cycle. As a reasonable prerequisite we assume that each phase should be given at least 20 seconds of green time, which translates to the candidate for a trapping region given by \( T = [20, 40]^4 \).

From the nature of the problem, we expect the reward function to be continuous, but we do not have an analytical formula for it. Therefore, we apply Algorithm 2 and sample the candidate for a trapping region given by \( T \). Learning curves, which begin on the boundary of the trapping region do not escape it, and evolve in the interior of the set.

For our experiment, the selection of the strategy by the learners is performed via decoupled gradient descent, as in Equations (1) and (2). Each intersection controller estimates the gradient of own reward by difference quotients:

\[ \delta \nabla_x R_i(x) \approx R_i(x, x_{-i}) - R_i(x + \delta, x_{-i}) \]  

for some small \( \delta \) (in our experiments \( \delta = 0.1 \)). The adaptation rate \( \gamma \) is set to \( 10^{-6} \). Such settings were chosen as they would give satisfactory results for learning on one intersection, while keeping other intersections fixed.

As discussed previously, non-convergence is an undesirable learning effect in such traffic management scenario, as one would like to ensure that learning always stays within some predetermined bounds, so minimal green time can be given to vehicle flows and pedestrians within each cycle. As a reasonable prerequisite we assume that each phase should be given at least 20 seconds of green time, which translates to the candidate for a trapping region given by \( T = [20, 40]^4 \).

From the nature of the problem, we expect the reward function to be continuous, but we do not have an analytical formula for it. Therefore, we apply Algorithm 2 and sample the candidate for a trapping region given by \( T \). Learning curves, which begin on the boundary of the trapping region do not escape it, and evolve in the interior of the set.
References


[Wang et al., 2021] Xiaoyan Wang, Jun Peng, Shuqiu Li, and Bing Li. Formal reachability analysis for multi-agent


