Distributed Influence-Augmented Local Simulators for Parallel MARL in Large Networked Systems

Miguel Suau
Delft University of Technology
m.suaudecastro@tudelft.nl

Jinke He
Delft University of Technology
j.he-4@tudelft.nl

Mustafa Mert Çelikok
Aalto University
mustafa.celikok@aalto.fi

Matthijs T. J. Spaan
Delft University of Technology
m.t.j.spaan@tudelft.nl

Frans A. Oliehoek
Delft University of Technology
f.a.oliehoek@tudelft.nl

Abstract

Due to its high sample complexity, simulation is, as of today, critical for the successful application of reinforcement learning. Many real-world problems, however, exhibit overly complex dynamics, which makes their full-scale simulation computationally slow. In this paper, we show how to factorize large networked systems of many agents into multiple local regions such that we can build separate simulators that run independently and in parallel. To monitor the influence that the different local regions exert on one another, each of these simulators is equipped with a learned model that is periodically trained on real trajectories. Our empirical results reveal that distributing the simulation among different processes not only makes it possible to train large multi-agent systems in just a few hours but also helps mitigate the negative effects of simultaneous learning.[1]

1 Introduction

Imagine we have to train a team of agents in controlling the traffic lights of a very large city, so large that we simply cannot control all traffic lights using a single policy. The first step would be to split the problem into multiple sub-regions. A natural division would be to assign one traffic light to each agent. Then, since the agents act locally, we would limit their observations to contain only local information. This partial observability could affect their optimal policies but would also make each individual decision-making problem more manageable (McCallum, 1995; Dearden and Boutilier, 1997). Moreover, we may also want to reward agents only for what occurs in their local neighborhood such that we reduce the variance of the returns (Spooner et al., 2021) and facilitate credit assignment (Castellini et al., 2020). Finally, we could train all agents together on a big traffic simulator that reproduces the global dynamics. However, if the city is truly large, it could take weeks or even months to optimize their policies. That is assuming training actually converges.

One may argue that, since the agents' observations and rewards are local, we could as well train them on separate simulators that model only the local transition dynamics (i.e. cars moving within each of the sub-regions; van der Pol and Oliehoek 2016). This approach might work if the agents' local transitions are isolated from the rest of the system (Becker et al., 2003), but would probably break when the local regions are coupled. This is because the local simulators would fail to account for the fact that the agents' local regions belong in a larger system and depend on one another. A solution is to model the influence the global system exerts on each of the local regions. Fortunately, this does not necessarily imply modelling the entire system, or else we would just use the global simulator. In

[1] Source code is available at https://github.com/INFLUENCEorg/DIALS.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

many scenarios, such as in the traffic problem, even though the local regions may be affected by many external variables (e.g.traffic densities in other parts of the city), they are only directly influenced by a small subset of them (e.g. road segments that connect the intersections with the rest of the city). This subset of variables are known as the influence sources. The theoretical framework of Influence-Based Abstraction (Oliehoek et al., 2021) shows that by monitoring the posterior distribution of the influence sources given the action local state history (ALSH) one can simulate realistic trajectories that match those produced by the global simulator. The resulting simulator, known as the influence-augmented local simulator (IALS), has been proven effective in single agent scenarios when combined with planning (He et al., 2020) and reinforcement learning (RL) algorithms (Suau et al., 2022b).

In this paper we extend the IBA framework to multi-agent domains. We show how to factorize large networked systems, such as the previous traffic example, into multiple sub-regions so that we can replace the GS by a distributed network of IALSs that can run independently and in parallel. There is one important caveat to this. The IBA framework assumes there is only a single agent learning at a time. This assumption is needed to make the influence distributions stationary. This implies that in our case, since we want the agents to learn simultaneously, previously computed influence distributions would no longer be valid after the agents update their policies. The naive solution would be to recompute new influence distributions every time any of the agents updates its policy. However, we argue that this is not only impractical, since recomputing the distributions is not without costs, but also undesirable. The theoretical results in Section 4.1 demonstrate that multiple (similar) joint policies may induce the same influence distributions, and that even when they vary a little, they can still elicit the same optimal policies. Further, our insights in Section 4.3 hint that what seems to be a problem at first, may in fact be an advantage, since in many situations, maintaining the previous influence distributions implies that the local transitions, although biased, remain stationary.

Contributions The main contributions of this paper are: (1) adapting IBA to multi-agent reinforcement learning (MARL),[2] and demonstrating that simultaneous learning is possible without incurring major computational costs, (2) showing that by distributing the simulation among different processes we can parallelize training and scale up to systems with many agents, (3) revealing that the non-stationarity issues inherent to MARL are partly mitigated as a result of this training scheme.

2 Related Work

A few prior works have investigated the computational benefits of factorizing large systems into independent local regions (Nair et al., 2005; Varakantham et al., 2007; Kumar et al., 2011; Witwicki and Durfee, 2011). Unfortunately, since local regions are often coupled to one another, such factorizations are not always appropriate. Nonetheless, in many cases, the interactions between regions occur through a limited number of variables. Using this property, the theoretical work by Oliehoek et al. (2021) on influence-based abstraction (IBA) describes how to build influence-augmented local simulators (IALS) of local-POMDPs, which model only the variables in the environment that are directly relevant to the agent while monitoring the response of the rest of the system with the influence predictor. The problem is that the exact computation of the conditional influence distribution is intractable and we can only try to estimate it from data. Congeduti et al. (2021) provide theoretical bounds on the value loss when planning with approximate influence predictors. The work by He et al. (2020) has empirically demonstrated the advantage of this approach to improve the efficiency of online planning in two discrete toy problems. Suau et al. (2022b) scale the method to high-dimensional problems by integrating the IBA framework with single-agent RL showing that the IALS can train policies much faster than the GS. In this paper, we extend the IBA solution to MARL and explain how to build a network of independent IALS such that we can train agents in parallel.

One of the consequences of training agents on independent simulators is that the non-stationarity issues arising from having the agents learn simultaneously are partly mitigated. There is a sizeable body of literature that concentrates on this issue (Hernandez-Leal et al., 2017), we include a review of these works in Appendix B for completeness. However, we note that the main purpose of this paper is to scale MARL up to systems with many agents. Hence, we are not concerned here with comparing our method with those that exclusively target non-stationarity, especially given that, for scalability reasons, these cannot be applied to the high-dimensional problems we consider here.

[2] Although the original IBA formulation (Oliehoek et al., 2012) is already framed as multi-agent, it assumes agents learn one at a time while the other agents' policies are fixed.

3 Preliminaries

The type of problems we describe in the introduction can be formulated as factored partially observable stochastic games (Hansen et al., 2004), which are defined as follows.

Definition 1 (fPOSG)**.** A factored partially observable stochastic game (fPOSG) is a tuple $\langle N, S, A, T, \{R_i\}, \Omega, \{O_i\}\rangle$ where $N = \{1, ..., n\}$ is the set of n agents, S is the set of j state variables $S = \{S^1, ..., S^k\}$, such that every state $s^t \in \times_{j=1}^{k} S^j$ is a k-dimensional vector $s^t = \langle s^{1,t}, ..., s^{k,t}\rangle$, $A = \times_{i \in N} A_i$ is the set of joint actions $a^t = \langle a_1^t, ..., a_n^t\rangle$, with A_i being the set of actions for agent i, T is the transition function, with $T(s^{t+1}|s^t, a^t)$, $R_i(s^t, a_i^t)$ is the immediate reward for agent i, $\Omega = \times_{i \in N} \Omega_i$ is the set of joint observations $o_i^t = \langle o_1^t, ..., o_n^t\rangle$, with Ω_i being the set of observations for agent i, and O_i is the observation function for agent i, $O_i(o_i^t|s^t)$.

Solving the fPOSG implies finding the policy π_i for each agent i that maximizes the expected return G^t; as defined in Sutton and Barto (1998). However, agents receive only partial observations o_i of the true state s, which are not necessarily Markovian. Therefore, optimal policies are in general history-dependent in a POSG. Hence, we define the agents' policies $\pi_i(a_i^t|h_i^t)$ as mappings from action-observation histories (AOH), $h_i^t = \langle o_i^1, a_i^1, ..., a_i^{t-1}, o_i^t\rangle$, to probability distributions over actions, such that agent i's optimal policy π_i^* is the one that for every AOH h_i^t selects the action with the highest Q-value $Q^{\pi_i^*}(h_i^t, a_i^t) = \mathbb{E}\left[G^t \mid h_i^t, a_i^t, \pi_i^*\right]$.

Given the structural assumptions we made in the introduction about the agents' only being able to observe and be rewarded for what occurs in their local neighborhood, we can narrow down the problem formulation and work with a specific class of fPOSGs called local-form fPOSGs (Oliehoek et al., 2021), which better encompass the problems we consider here.

Definition 2 (Local-form fPOSG)**.** A Local-form fPOSG is a fPOSG where O_i and R_i depend only on a subset of m state variables $X_i = \{X_i^1, ..., X_i^m\} \subseteq S$, with $m \leq k$ (number of state variables), and agent i's local states $x_i \in \times_{j=1}^{m} X_i^j$ being vectors $x_i^t = \langle x_i^{1,t}, ..., x_i^{k,t}\rangle$, such that $O_i(o_i^t|s^t) = \dot{O}_i(o_i^t|x_i^t)$ and $R_i(s^t, a_i^t) = \dot{R}_i(x_i^t, a_i^t)$, where $\dot{O}_i$ and $\dot{R}_i$ are the local observation and reward functions for agent i.

3.1 Influence-Based Abstraction

We now describe the IBA framework (Oliehoek et al., 2021) which intends to simplify the local-form fPOSG formulation by exploiting its structural properties. The framework assumes that there is a single agent i learning at a time while all other agents' polices π_{-i} are fixed. Hence, from the perspective of agent i the problem reduces to a POMDP (Kaelbling et al., 1996) where states are pairs $\langle s^t, h_{-i}^t\rangle$, with h_{-i}^t being the AOHs of all agents but agent i (Nair et al., 2003). Finding a locally optimal solution to the Local-form fPOSG can be approached by 'alternating maximization' or 'coordinate ascent'. That is, sequentially iterating over all agents, possibly multiple times, and solving their respective POMDPs (Nair et al., 2003; Oliehoek and Amato, 2016).

Looking at the definition of Local-form fPOSG, one can argue that, when solving for agent i, sampling actions from the policies of all the other agents $\pi_{-i}(a_{-i}^t|h_{-i}^t)$ and simulating the transitions $T(s^{t+1}|s^t, a^t)$ of the full set of state variables is unnecessary, and while doing so is possible in small problems, it might become computationally intractable in large domains with many agents. Instead, we can define a new transition function $\bar{T}_i$ that models only agent i's local state variables x_i, $\bar{T}_i(x_{i,t+1}|x_i^t, a_i^t)$. The problem is that x_i^{t+1} may still depend on the other agents actions a_{-i}^t and the non-local state variables $S \setminus X_i$, which means that $\bar{T}_i$ is not well defined. Fortunately, in many problems, only a fraction of the non-local state variables will *directly influence* agent i's local region.

The diagram on the left of Figure 1 is a Dynamic Bayesian Network (DBN) (Pearl, 1988; Boutilier et al., 1999) describing a particular instance of the transition dynamics for a generic agent i in a local-form fPOSG. Agent i's local region, corresponds to the variables that lie within the red box, $x_i \in X_i = \{X_i^1, X_i^2\}$. The diagram also shows the non-local variables, known as influence sources $u_i \in U_i \subseteq S \setminus X_i$, that influence the local region directly. The three dots on the top indicate that there can be, potentially many, other non-local variables in S affecting the local variables X_i. These are denoted by $y_i \in Y_i \subseteq S \setminus X_i \cup U_i$. The diagram also shows that agent j can affect agent i's local region through its actions $a_j \in A_j$. However, both Y_i and A_j can only influence X_i via U_i. Hence, given u_i^t, x_i^{t+1} is conditionally independent of y_i^t and a_j^t, therefore $P(x_i^{t+1}|x_i^t, u_i^t, y_i^t, a_j^t) = P(x_i^{t+1}|x_i^t, u_i^t)$.

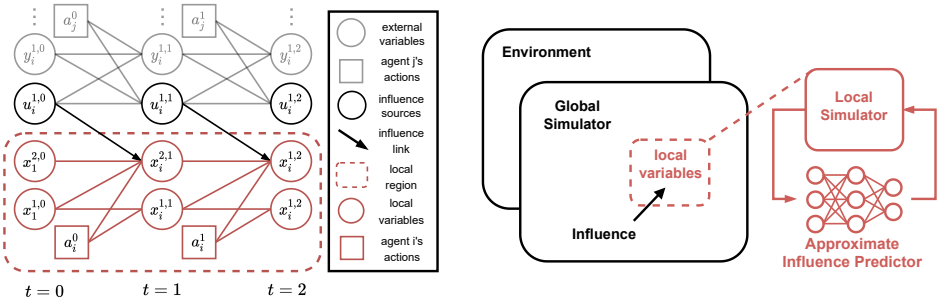


Figure 1: **Left:** A Dynamic Bayesian Network showing agent i's transition dynamics in a local-form fPOSG prototype. **Right:** A conceptual diagram of the IALS.

The above implies that by inferring the value of the influence source u_i^t, we can monitor the influence of the other agents and the non-local state variables and thus compute the local state transitions. This can be done by keeping track of the action-local-state history (ALSH) $l_i^t = \langle x_i^1, a_i^1 ..., a_i^{t-1}, x_i^t \rangle$.

Definition 3 (IALM)**.** An influence-augmented local Model (IALM) for agent i is a tuple $\langle X_i, U_i, A_i, \dot{T}_i, \dot{R}_i, \Omega_i, \dot{O}_i, I_i \rangle$, with local states $x_i \in \times_{j \in |X_i|} X_i^j$, influence sources $u_i \in \times_{j \in |U_i|} U_i^j$, local transition function $\dot{T}_i(x_i^{t+1} | x_i^t, u_i^t, a_i^t)$, local observation function $\dot{O}_i(o_i^{t+1} | x_i^{t+1})$, local reward function $\dot{R}_i(x_i^t, a_i^t)$, and influence distribution $I_i(u_i^t | l_i^t)$.

Using the IALM we can compute agent i's local transitions as

$$P(x_i^{t+1} | l_i^t, a_i^t) = \sum_{u^t} \dot{T}_i(x_i^{t+1} | x_i^t, u_i^t, a_i^t) I_i(u_i^t | l_i^t). \tag{1}$$

Note that, as opposed to the Local-form fPOSG, the transition function $\dot{T}_i$ in the IALM is defined purely in terms of the local state variables and the influence sources. Moreover, since u_t *d-separates* (Bishop, 2006) x_t from y_t, we only need to maintain a belief over u_i^t, $I_i(u_i^t | l_i^t)$, rather than over the full set of of state variables s^t and other agents' histories h_{-i}^t, $P(s^t, h_{-i}^t | l_i^t)$. All in all, this translates into a much more compact, yet *exact* representation of the problem (Oliehoek et al., 2021), which should be computationally much lighter than the original Local-form fPOSG.

3.2 Influence-Augmented Local Simulators

Here we briefly describe how the IALM formulation can be used in practice to build IALS (Suau et al., 2022b), which consist of a *local simulator* and an *approximate influence predictor*.

Local simulator (LS): The LS is an abstracted version of the environment that only models a small portion of it. As opposed to a global simulator (GS), which should closely reproduce the dynamics of every state variable, the LS focuses on characterizing the transitions of those variables X_i that agent i directly interacts with, $\dot{T}_i(x_i^{t+1} | x_i^t, u_i^t, a_i^t)$.

Approximate influence predictor (AIP): The AIP monitors the interactions between agent i's local region X_i, the external variables Y_i, and the other agents' actions A_{-i}, by estimating $I_i(u_i^t | l_i^t)$. Since, due to combinatorial explosion, computing the exact probability $I_i(u_i^t | l_i^t)$ is generally intractable (Oliehoek et al., 2021), a neural network is used instead to approximate the influence distribution. Thus, we write $\hat{I}_{\theta_i}$ to denote agent i's AIP, where θ_i are the network parameters. The AIP $\hat{I}_{\theta_i}$ is trained on a dataset D_i of N samples of the form (l_i^t, u_i^t) collected from the GS. Since role of the AIP is to estimate the conditional probability of the influence sources u_i^t given the past ALSH, we can formulate the task as a classification problem and optimize the network using the expected cross-entropy loss (Bishop, 2006).

4 Distributed Influence-Augmented Local Simulators

As mentioned in the previous section, local-form fPOSGs are solved iteratively in the IBA framework. This means that only a single agent can update its policy at a time. Here, we relax this assumption and discuss the advantages and disadvantages of simultaneous learning. Proofs for all the theoretical results in this section can be found in Appendix A.

4.1 Enabling Parallelization

The main reason to disallow simultaneous learning is that changes in the other agents' policies can affect agent i's influence distribution $I_i(u_i^t|l_i^t)$, which may become non-stationary. This renders previously computed influences useless because they no longer capture the true response of the global system.

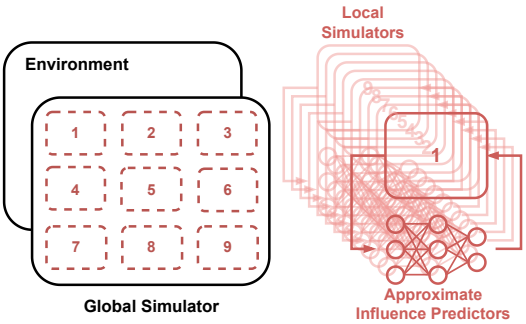


Figure 2: A conceptual diagram of the DIALS

This restriction, however, prevents IBA from unlocking its full potential. The fact that each agent's IALS is independent of the oth'ers means that the computations can be distributed among different processes that can run in parallel. Hence, putting aside the non-stationarity issue, and assuming no overhead costs in spawning an increasing number of processes, the total runtime of the method would stay constant if the dimensionality of the global system grew, either because the number of non-local variables or the number of agents increased. This is in contrast to having agents learn simultaneously in the same GS, in which case larger environments imply longer runtimes. Moreover, since each IALS simulates only a portion of the environment the total amount of memory space needed would be split among the different processors. Hence, we could run the simulation on multiple machines with small memory rather than one big machine with very large memory.

In principle, one could prevent the AIPs from becoming stale by simply updating all $\{\hat{I}_{\theta_i}(u_i^t|l_i^t)\}_{i\in N}$ every time any of the other agents changes its policy. However, this creates a difficult moving target problem, and makes the whole method very inefficient since, especially in deep RL, policies are updated very frequently. Fortunately, as we argue in the following, in many cases, paying the extra cost of retraining the AIPs is neither necessary nor desirable.

4.1.1 Multiple joint policies may induce the same influence distribution

In the following, we show that multiple joint policies may often map onto the same influence distribution $I_i(u_i^t|l_i^t) \in \Psi_i$ for agent $i \in N$.

Lemma 1. *Let $\Pi = \times_{i\in N}\Pi_i$ be the product space of joint policies with Π_i being the set of polices for agent i. Moreover, let $\Psi = \times_{i\in N}\Psi_i$ be the product space of joint influences, with Ψ_i being the set of influence distributions for agent i. Every joint policy $\pi \in \Pi$ induces exactly one influence distribution $I_i(u_i^t|l_i^t) \in \Psi_i$ for every agent $i \in N$.*

Proposition 1. *The space of joint policies $\Pi = \times_{i\in N}\Pi_i$ is necessarily greater than or equal to the space of joint influences $\Psi = \times_{i\in N}\Psi_i$, $|\Pi| \geq |\Psi|$. Moreover, there exist local-form fPOSGs for which the inequality is strict.*

The advantages of this result were shown empirically by Witwicki and Durfee (2010), who demonstrated that planning times can be reduced by searching the space of joint influences rather than the space of joint policies, which is often much larger. In fact, in the extreme case of local transition independence (Becker et al., 2003),[3] we have that for all joint policies π there is a single $\{I_i\}_{i\in N}$

Corollary 1. *Let agent i's influence sources u_i^t be independent of the other agents' actions a_{-i}. Then, for any joint policy $\pi \in \Pi$, there is a unique influence distribution $I_i^* \in \Psi_i$ for every agent $i \in N$ and $|\Pi| \gg |\Psi| = 1$.*

The result above implies that, in this particular case, we would only need to train the AIPs once at the beginning. Although we do not expect the situation in Corollary 1 to be the norm, we do believe that in many scenarios, such as in the two environments we explore here, we would not need to retrain the AIPs very often because similar joint policies will influence the local regions in very similar, if not in the same, ways. Furthermore, the next result shows that even when this is not the case, an outdated I_i computed from an old joint policy might still produce the same optimal policy for agent i.

[3] As opposed to IBA, Becker et al. (2003) assume agents are tied by a shared global reward.

4.1.2 Multiple influence distributions may induce the same optimal policy

We use the simulation lemma (Kearns and Singh, 2002) to prove that if two influence distributions are similar enough they will induce the same optimal policy.

Lemma 2. *Let M_i^1 and M_i^2 be two IALMS differing only on their influence distributions $I_i^1(u_i^t|l_i^t)$ and $I_i^2(u_i^t|l_i^t)$. Let $Q_{M_i^1}^{\pi_i}$ and $Q_{M_i^2}^{\pi_i}$ be the value functions induced by M_i^1 and M_i^2 for the same π_i. If I_i^1 and I_i^2 satisfy*

$$\sum_{l_i^t, u_i^t} P(l_i^t|h_i^t) \left| I_i^1(u_i^t|l_i^t) - I_i^2(u_i^t|l_i^t) \right| \leq \xi, \text{ then } \left| Q_{M_i^1}^{\pi_i}(h_i^t, a_i^t) - Q_{M_i^2}^{\pi_i}(h_i^t, a_i^t) \right| \leq \bar{R} \frac{(H-t)(H-t+1)}{2} \xi \tag{2}$$

for all π_i, h_i^t, and a_i^t, where H is the horizon and $\bar{R} = ||R||_\infty$

Intuitively, Lemma 2 shows that the difference in value between M_i^1 an M_i^2 is upper-bounded by the maximum difference between I_i^1 and I_i^2 times a constant. Actually, if the *action-gap* (Farahmand, 2011) (i.e. value difference between the best and the second best action) in one of the IALMs is larger than twice the difference between the $Q_{M_1}^{\pi_i}$ and $Q_{M_2}^{\pi_i}$ the IALMs share the same optimal policy.

Theorem 1. *Let M_i^1 and M_i^2 be two IALMS differing only on their influence distributions $I_i^1(u_i^t|l_i^t)$ and $I_i^2(u_i^t|l_i^t)$. M_i^1 and M_i^2 induce the same optimal policy π^* if, for some Δ,*

$$Q_{M_i^1}^{\pi_i^*}(h_i^t, \bar{a}_i^t) - Q_{M_i^1}^{\pi_i^*}(h_i^t, \hat{a}_i^t) > 2\Delta \quad \forall h_i^t, \hat{a}_i^t \neq \bar{a}_i^t \text{ with } \left| Q_{M_i^1}^{\pi_i}(h_i^t, a_i^t) - Q_{M_i^2}^{\pi_i}(h_i^t, a_i^t) \right| \leq \Delta \quad \forall h_i^t, a_i^t, \pi_i, \tag{3}$$

where $\bar{a}_i^t = \arg\max_{a_i^t} Q_{M_i^1}^{\pi_i^*}(h_i^t, a_i^t)$

Combining Lemma 2 and Theorem 1, we see that, because the difference in value between M_i^1 an M_i^2 depends on ξ (Lemma 2), the closer the distributions I_i^1 and I_i^2 are, the more likely it is that M_i^1 and M_i^2 share the same optimal policy. Note that we have no control over the action gap as it is domain-dependent. In some domains, the gap might be large and we can be more relaxed about not retraining the AIPs. In some others, the gap might be small and we may need to retrain the AIPs more frequently.

4.2 Algorithm

After the analysis above, we are now ready to present our method, which we call Distributed Influence-Augmented Local Simulators (DIALS). Algorithm 1 describes how we can train multi-agent systems with DIALS. As mentioned earlier, the key advantage in using DIALS is that simulations can be distributed among different processes, and thus training can be fully parallelized. This enables MARL to scale to very large systems with many learning agents. Moreover, following from our theoretical results, AIP training, which can also be done in parallel, is performed only every certain number of timesteps. The hyperparameter F in Algorithm 1 controls the AIPs' training frequency. The effect of F on the learning performance is empirically investigated in Section 5.

Algorithm 1 MARL with DIALS

```
Initialize policies {π_i}_{i∈N} and AIPs {Î_{θ_i}}_{i∈N}
repeat
    Collect datasets {D_i}_{i∈N} from GS                    ▷ See Algorithm 2 in Appendix C
    in parallel, for i ∈ N do
        Train AIP Î_{θ_i} on dataset D_i                    ▷ See Section 3.2
    end for
    in parallel, for i ∈ N do
        for F steps do                                      ▷ F is the AIPs' training frequency
            Simulate trajectories with IALS ⟨Ṫ_i, Ṙ_i, Ȯ_i, Î_{θ_i}⟩   ▷ See Algorithm 3 in Appendix C
            Train policy π_i                                ▷ Using any standard RL method
        end for
    end for
until end of training
```

4.3 Mitigating the negative effects of simultaneous learning

The results in the previous section showed that the AIPs may not need to be retrained every single time policies are updated, as the influence distributions may often stay the same or vary only a little. Yet, we now argue that, even when changes in the joint policy do affect the influence distributions $I_i(u_i^t|l_i^t)$ significantly, it may be advantageous not to retrain the AIPs.

First, we have already mentioned that when all agents learn simultaneously in the same simulator the transition dynamics often look non-stationary from the perspective of each individual agent. This may result in sudden performance drops caused by oscillations in the value targets (Claus and Boutilier, 1998). In contrast, when using independent IALS to train our agents, the transition dynamics remain stationary unless we update the AIPs. Hence, by not updating the AIPs too frequently, we get a biased but otherwise more consistent learning signal that the agents can rely on to improve their policies.

Second, we posit that the poor empirical convergence of many off-the-shelf Deep RL methods (Hernandez-Leal et al., 2017; Yu et al., 2021) is also because stochastic gradient descent updates often result in policies that perform worse than the previous ones. Thus, when learning together, agents may try to adapt to other agents' poor performing polices. These policies, however, are likely to be temporary as they are just a result of the inherent stochasticity of the learning process. Similarly, in many environments, agents shall take exploratory actions before they can improve their policies, which may also negatively impact cooperation if they learn simultaneously (Zhang et al., 2009, 2010). In our case, we can again benefit from the fact that the AIPs need to be purposely retrained, and do so only when the policies of the other agents have improved sufficiently.

Even though, further theoretical analysis would be needed to be more conclusive about the benefits of using independent simulators, the observations above give reasons to believe that what we initially described as a problem may in fact be an advantage. This view is also supported by our experiments.

5 Experiments

The goal of the experiments is to: (1) test whether we can reduce training times by replacing GS with DIALS, (2) investigate how the method scales to large environments with many learning agents, (3) evaluate the convergence benefits of using separate simulators to train agents rather than a single GS, and (4) study the effect of the AIPs' training frequency F on the agents' learning performance.

5.1 Experimental setup

Agents are trained independently with PPO (Schulman et al., 2017)[4] on (1) the global simulator (GS), (2) distributed influence-augmented local simulators (DIALS) with AIPs trained periodically on datasets collected from the GS using the most recent joint policy, (3) DIALS with untrained AIPs (untrained-DIALS).

To measure the agent's performance, training is interleaved with periodic evaluations on the GS. The results are averaged over 10 random seeds on all except on the largest scenarios (10×10) for which, due to computational limitations we could only run 5 seeds. We report the mean return of all learning agents. We also compare the simulators in terms of total runtime. For DIALS this includes: the agents' training time, the AIPs' training time, and the time for data collection.

5.2 Environments

Traffic control The first domain we consider is a multi-agent variant of the traffic control benchmark proposed by Vinitsky et al. (2018). In this scenario agents are requested to manage the lights of a big traffic network. Each agent controls a single traffic light and can only observe cars when they are inside the intersection's local neighborhood. Their goal is to maximize the average speed of cars within their respective intersections. To demonstrate the scalability of the method we evaluate DIALS on four different variants of the traffic network with 4, 25 , 49, and even 100 intersections (agents). A screenshot of the traffic network with 25 intersections is shown in Appendix F. The GS and LS are built using Flow (MIT License) (Wu et al., 2017) and SUMO (Eclipse Public License Version 2)

[4]The vanilla PPO algorithm with decentralized value functions (independent PPO; IPPO) has been shown to perform exceptionally well on several multi-agent environments (de Witt et al., 2020; Yu et al., 2021).

(Lopez et al., 2018). The GS simulates the entire traffic network while each LS models only the local neighborhood of each intersection $i \in N$ (Figure 10 in Appendix F). We use the same LS for every agent-intersection but since, depending on where they are located, they are influenced differently by the rest of the traffic network, we have separate AIPs, $\{I_{\theta_i}\}_{i \in N}$, for each them. These are feedforward neural networks with the same architecture but different weights θ_i, trained periodically with frequency F on datasets $\{D_i\}_{i \in N}$ collected from the GS. The influence sources u_t^i are binary variables indicating whether or not a car will be entering from each the four incoming lanes.

Warehouse Commissioning The second domain we consider is a warehouse commissioning task (Suau et al., 2022b). A team of robots (blue) need to fetch the items (yellow) that appear with probability 0.02 on the shelves (dashed black lines) of the warehouse (see Figure 9 in Appendix F). Each robot has been designated a 5×5 square region and can only collect the items that appear on the shelves at the edges. The regions overlap so that each of the 4 item shelves in a robot's region is shared with one of its 4 neighbors. The robots receive a reward between $[0, 1]$ when collecting an item. The exact value depends on how old the item is compared to the other items in their region. This is to encourage the robots to collect the oldest items first. The robots receive as observations a bitmap encoding their own location and a set of 12 binary variables that indicate whether or not a given item needs to be collected. The robots, however, cannot see the location of the other robots even though all of them are directly or indirectly influencing each other through their actions. We built four variants of the warehouse with 4, 25, 49, and 100 robots (agents). A screenshot of the warehouse with 25 robots is shown in Appendix F. The GS simulates the entire warehouse while the LS models only a 5×5 square region (Figure 10 in Appendix F). We use the same LS for every robot (agent) but since, depending on where they are located they are influenced differently by the rest of the robots, we have separate AIPs, $\{\hat{I}_{\theta_i}\}_{i \in N}$, for each of them. These are GRUs (Cho et al., 2014) with the same architecture but different weights θ_i, which we train periodically with frequency F on datasets $\{D_i\}_{i \in N}$ collected from the GS. Robot i's influence sources u_i^t encode the location of the four neighbor robots. If its AIP $\hat{I}_{\theta_i}$ predicts that any of the neighbor robots is at one of the 12 cells within its region, and there is an active item on that cell, that item is removed and robot i can no longer collect it.

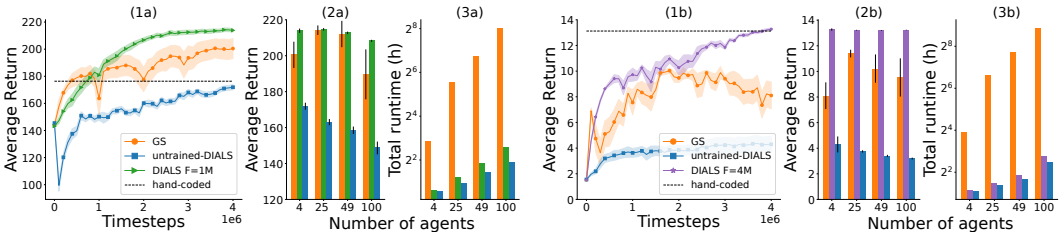
5.3 Results

GS vs. DIALS The two plots on the left of Figures 3a and 3b show the average return as a function of the number of timesteps obtained with GS, DIALS, and untrained-DIALS on the 4-agent traffic and warehouse environments. Shaded areas indicate the standard error of the mean. Agents are trained for 4M timesteps on all three simulators. The results reveal that, while agents trained on DIALS seem to converge steadily towards similar high-performing policies, agents trained with the GS often get stuck in local minima, hence the poor mean episodic reward and large standard error obtained with the GS relative to that of the DIALS. In contrast, the low performance of agents trained with the untrained-DIALS indicates that estimating the influences correctly is important for learning good policies. It is worth noting that the gap between the GS and the DIALS is larger in Figure 3b than in Figure 3a. We posit that this is because, in the warehouse domain, agents are more strongly coupled. For comparison, the dashed-black lines in the plots on the left of Figures 3a and 3b show the performance of hand-coded policies. For the traffic domain we used fixed traffic light controllers that were extensively optimized by Wu et al. (2017). For the warehouse domain, we hand-coded policies that follow the shortest path toward the oldest item in the agent's region. The learning curves for the other scenarios are provided in Appendix D together with further discussion on these results.[5][6]

Scalability The benefits of parallelization are more apparent when moving to larger environments. The two bar plots in Figures 3a and 3b depict the final average return and the total run time of training 4, 25, 49 and 100 agents on the two tasks for 4M timesteps. The plots show that DIALS scales far better than the GS to larger problem sizes, while also yielding better performing policies. For example, training **100 agents** on the traffic network takes less than **6 hours** with the DIALS, whereas training them with the GS would take more than **10 days**. This is a **speedup factor of 40**. In fact,

[5] A video showing the GS of the traffic network and one of the IALS is provided at https://youtu.be/DgVE6OIQQz8.

[6] A video showing how agents trained with DIALS perform on the 100-agent variant of the traffic scenario is provided at https://youtu.be/G9EthZ-G3vo.



(a) Traffic (b) Warehouse

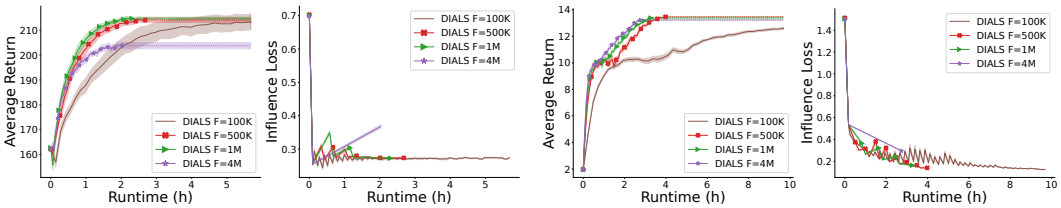
Figure 3: **(1a) and (1b)** Learning curves with the three simulators on the 4-intersection traffic and 4-robot warehouse environments. **(2a) and (2b):** Final average return of agents trained with the three simulators for 4M timesteps. **(3a) and (3b):** Total runtime of training with the three simulators for 4M timesteps. The y-axis is in $\log_2$ scale.

since the maximum execution time allowed by our computer cluster is 1 week, the results reported for the GS in the scenarios of size 10×10 do not correspond to 4M timesteps but the equivalent of 1 week of training. We would also like to point out that, disregarding the overhead costs associated with multiprocessing, the DIALS runtime should remain constant independently of the problem size. However, to update the AIPs, new samples are collected form the GS, which does increase the runtime. This explains the gap between DIALS and untrained-DIALS. That said, the number of samples needed to update the AIPs (80K for traffic and 10K for warehouse) is significantly lower than the samples needed to train the agents (4M), which is why the runtime difference between GS and DIALS is so large. A table with a breakdown of the runtimes is given in Appendix G.

AIPs' training frequency Our first results have already demonstrated that isolating the agents in separate simulators and not updating the AIPs too frequently can be beneficial for convergence. We now further investigate this phenomenon by evaluating the agents' learning performance for different values of the hyperparameter F. The two plots on the left of Figures 4a and 4b show the learning curves for agents trained on DIALS where F is set to 100K, 500K, 1M, and 4M timesteps. In the traffic domain, the gap between the green and the purple curve (Figure 4a) suggests that it is important to retrain the AIPs at least every 1M timesteps, such that agents become aware of changes in the other agents' policies. In contrast, in the warehouse domain (Figure 4b), we see that training the AIPs only once at the beginning (DIALS $F = 4$M) seems sufficient. In fact, updating the AIPs too frequently (DIALS $F = 100$K) is detrimental to the agents' performance. This is consistent with our hypothesis in Section 4.3. The plots on the right show the average cross-entropy (CE) loss of the AIPs evaluated on trajectories sampled from the GS. As explained in Section 4 since all agents learn simultaneously, the influence distributions $\{I(u_i^t|l_i^t)\}_{i \in N}$ are non-stationary. For this reason, we see that the CE loss changes as the policies of the other agents are updated. We can also see how the CE loss decreases when the AIPs are retrained, which happens more or less frequently depending on the hyperparameter F. Note that the CE not only measures the distance between the two probability distributions but also the absolute entropy. In the warehouse domain, the neighbor robots' locations become more predictable (lower entropy) as their policies improve. This explains why in the first plot from the right the CE loss decreases even though the AIPs are not updated. Also in the same plot, even though by the end of training DIALS $F = 4$M is highly inaccurate, as evidenced by the gap between the purple and the other curves, it is still good enough to train policies that match the performance of those trained with DIALS $F = 500$K and $F = 1$M. This is in line with our results in Section 4. The same plots for the rest of scenarios are provided in Appendix D.

6 Scope and Limitations

DIALS targets networked environments with well-defined local regions where the interactions between different regions occur through a limited number of variables. There is plenty of examples of domains that have this particular structure including traffic, heating and water systems, logistics, telecommunications, etc. Knowledge of the influence sources U and how these affect the local regions is required for building the DIALS. In most cases however, (as in the two environments we explored here) some domain knowledge suffices to be able to tell what the influence sources are.



(a) Traffic 25 agents

(b) Warehouse 25 agents

Figure 4: **Left (a) and (b):** Learning curves with DIALS for different values of F on the 25-agent versions of the two environments. **Right (a) and (b):** Influence CE loss as a function of runtime averaged over the 25 AIPs.

Moreover, having or being able to build high-fidelity local simulators of these local regions is also a requirement. Fortunately, there exist plenty of simulators of real systems that can readily be used such as, SUMO (Lopez et al., 2018), Robosuite (Zhu et al., 2020), BRAX (Freeman et al., 2021). There is also a lot of commercial software for building custom-made simulators such as Mujoco (Todorov et al., 2012), or Unity (Juliani et al., 2018). Also, note that most (if not all) of the work that has applied RL to real-world problems relies on simulation to train the policies offline (Bellemare et al., 2020; Degrave et al., 2022). Hence, we believe that DIALS can have strong impact on many real world applications.

Finally, although the experiments reveal that DIALS can considerably accelerate training times, it is also memory demanding. As shown in Appendix H (Table 3), the total memory usage with DIALS increases exponentially with the number of simulators/processes. There is thus a trade-off between fast computation and total memory needed. Note however, that the memory is split among the different processes. Hence, rather than using a big machine with large memory, DIALS can run on several smaller ones with less memory.

7 Conclusion

This paper has offered a practical solution that allows training large networked systems with many agents in just a few hours. We showed how to factorize these systems into multiple sub-regions such that we could build distributed influence-augmented local simulators (DIALS).

The key advantage of DIALS is that simulations can be distributed among different processes, and thus training can be fully parallelized. To account for the interactions between the different sub-regions, the simulators are equipped with approximate influence predictors (AIPs), which are trained periodically on real trajectories sampled from a global simulator (GS). We demonstrated that, although with DIALS agents learn simultaneously, training the AIPs very frequently is neither necessary nor desirable. Our results reveal that DIALS not only enables MARL to scale up but also mitigates the non-stationarity issues of simultaneous learning.

Future work could analyse this phenomenon from a theoretical perspective, study how to adapt DIALS to more strongly coupled domains where frequent training of the AIPs is important, or design a method to directly estimate how the changes in the agents' policies affect the influence distributions. This is so that the AIPs can readily be updated without having to run the GS to generate new samples.

Acknowledgements

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 758824 —INFLUENCE). Mustafa Mert Çelikok is partially funded by KAUTE Foundation - The Finnish Science Foundation for Economics and Technology.

References

Becker, R., Zilberstein, S., Lesser, V., and Goldman, C. V. (2003). Transition-independent decentralized Markov decision processes. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems*, pages 41–48.

Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., and Wang, Z. (2020). Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94.

Castellini, J., Devlin, S., Oliehoek, F. A., and Savani, R. (2020). Difference rewards policy gradients. *arXiv preprint arXiv:2012.11258*.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*.

Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proc. of the National Conference on Artificial Intelligence*, pages 746–752.

Congeduti, E., Mey, A., and Oliehoek, F. A. (2021). Loss bounds for approximate influence-based abstraction. In *Proceedings of the Twentieth International Conference on Autonomous Agents and MultiAgent Systems*.

de Witt, C. S., Gupta, T., Makoviichuk, D., Makoviychuk, V., Torr, P. H., Sun, M., and Whiteson, S. (2020). Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*.

Dearden, R. and Boutilier, C. (1997). Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89(1-2):219–283.

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de Las Casas, D., et al. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419.

Farahmand, A.-m. (2011). Action-gap phenomenon in reinforcement learning. *Advances in Neural Information Processing Systems*, 24.

Foerster, J., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., and Mordatch, I. (2018a). Learning with opponent-learning awareness. In *Proceedings of the Seventeenth International Conference on Autonomous Agents and MultiAgent Systems*.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018b). Counterfactual multi-agent policy gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. (2021). Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*.

Hansen, E. A., Bernstein, D. S., and Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proc. of the National Conference on Artificial Intelligence*, pages 709–715.

He, J., Suau, M., and Oliehoek, F. (2020). Influence-augmented online planning for complex environments. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4392–4402. Curran Associates, Inc.

Hernandez-Leal, P., Kaisers, M., Baarslag, T., and de Cote, E. M. (2017). A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., et al. (2018). Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

Kaelbling, L. P., Littman, M., and Moore, A. (1996). Reinforcement learning: A survey. *Journal of AI Research*, 4:237–285.

Kearns, M. and Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2):209–232.

Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in neural information processing systems*, 12.

Kumar, A., Zilberstein, S., and Toussaint, M. (2011). Scalable multiagent planning using probabilistic inference. In *Proc. of the International Joint Conference on Artificial Intelligence*, pages 2140–2146.

Laurent, G. J., Matignon, L., Fort-Piat, L., et al. (2011). The world of independent learners is not markovian. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 15(1):55–64.

Li, W., Wang, X., Jin, B., Sheng, J., and Zha, H. (2021). Dealing with non-stationarity in marl via trust-region decomposition. In *International Conference on Learning Representations*.

Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., and Wießner, E. (2018). Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*.

Lyu, X., Xiao, Y., Daley, B., and Amato, C. (2021). Contrasting centralized and decentralized critics in multi-agent reinforcement learning. *Proceedings of the Twentieth International Conference on Autonomous Agents and MultiAgent Systems*.

McCallum, A. K. (1995). *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester.

Nair, R., Tambe, M., Yokoo, M., Pynadath, D. V., and Marsella, S. (2003). Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proc. of the International Joint Conference on Artificial Intelligence*, pages 705–711.

Nair, R., Varakantham, P., Tambe, M., and Yokoo, M. (2005). Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proc. of the National Conference on Artificial Intelligence*, pages 133–139.

van der Pol, E. and Oliehoek, F. A. (2016). Coordinated deep reinforcement learners for traffic light control. NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems.

Oliehoek, F., Witwicki, S., and Kaelbling, L. (2021). A sufficient statistic for influence in structured multiagent environments. *Journal of Artificial Intelligence Research*, 70:789–870.

Oliehoek, F. A. and Amato, C. (2016). *A Concise Introduction to Decentralized POMDPs*. Springer Briefs in Intelligent Systems. Springer.

Oliehoek, F. A., Witwicki, S. J., and Kaelbling, L. P. (2012). Influence-based abstraction for multiagent systems. In *AAAI12*.

Pearl, J. (1988). *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Rabinowitz, N., Perbet, F., Song, F., Zhang, C., Eslami, S. A., and Botvinick, M. (2018). Machine theory of mind. In *International conference on machine learning*, pages 4218–4227. PMLR.

Raileanu, R., Denton, E., Szlam, A., and Fergus, R. (2018). Modeling others using oneself in multi-agent reinforcement learning. In *International conference on machine learning*, pages 4257–4266. PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Spooner, T., Vadori, N., and Ganesh, S. (2021). Factored policy gradients: Leveraging structure for efficient learning in MOMDPs. *Advances in Neural Information Processing Systems*, 34.

Suau, M., He, J., Congeduti, E., Starre, R. A., Czechowski, A., and Oliehoek, F. A. (2022a). Influence-aware memory architectures for deep reinforcement learning in pomdps. *Neural Computing and Applications*, pages 1–17.

Suau, M., He, J., Spaan, M. T., and Oliehoek, F. (2022b). Influence-augmented local simulators: A scalable solution for fast deep rl in large networked systems. In *International Conference on Machine Learning*, pages 20604–20624. PMLR.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

Varakantham, P., Marecki, J., Yabu, Y., Tambe, M., and Yokoo, M. (2007). Letting loose a SPIDER on a network of POMDPs: Generating quality guaranteed policies. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems*.

Vinitsky, E., Kreidieh, A., Le Flem, L., Kheterpal, N., Jang, K., Wu, C., Wu, F., Liaw, R., Liang, E., and Bayen, A. M. (2018). Benchmarks for reinforcement learning in mixed-autonomy traffic. In *Conference on robot learning*, pages 399–409. PMLR.

Witwicki, S. J. and Durfee, E. H. (2010). Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In *Proc. of the International Conference on Automated Planning and Scheduling*, pages 185–192.

Witwicki, S. J. and Durfee, E. H. (2011). Towards a unifying characterization for quantifying weak coupling in Dec-POMDPs. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems*, pages 29–36, Taipei, Taiwan.

Wu, C., Kreidieh, A., Parvate, K., Vinitsky, E., and Bayen, A. M. (2017). Flow: A modular learning framework for autonomy in traffic. *arXiv preprint arXiv:1710.05465*.

Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., and Wu, Y. (2021). The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*.

Zhang, C., Abdallah, S., and Lesser, V. (2009). Integrating organizational control into multi-agent learning. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 757–764.

Zhang, C., Lesser, V. R., and Abdallah, S. (2010). Self-organization for coordinating decentralized reinforcement learning. In *AAMAS*, volume 10, pages 739–746.

Zhu, Y., Wong, J., Mandlekar, A., and Martín-Martín, R. (2020). robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] All proofs are included in Appendix A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code and instructions on how to reproduce all the experimental results are included in the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5.1 for description of the experimental setup and Appendix I for a full list of the hyperparameter configurations used for the experiments along with an explanation on how these were chosen.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] The shaded areas in the learning curves and the black vertical lines on the bar plots in Section 5.3 and Appendix D show the standard error of the mean.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The computer requirements and specifications along with a table showing a breakdown of the total runtimes for all experiments are provided in Appendix G.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proofs

Lemma 1. *Let $\Pi = \times_{i \in N} \Pi_i$ be the product space of joint policies with Π_i being the set of polices for agent i. Moreover, let $\Psi = \times_{i \in N} \Psi_i$ be the product space of joint influences, with Ψ_i being the set of influence distributions for agent i. Every joint policy $\pi \in \Pi$ induces exactly one influence distribution $I_i(u_i^t | l_i^t) \in \Psi_i$ for every agent $i \in N$.*

Proof. We will prove it by contradiction. Let us assume there is a single joint policy π that induces two different influence distributions I_i^1 and I_i^2 on agent i. From the definition of influence (Section 4.1; Oliehoek et al. 2021) we have

$$I_i^1(u_i^t | l_i^t) = \sum_{u_i^{t-1}, y_i^{t-1}, a_{-i}^{t-1}} P^1(u_i^t | x_i^{t-1}, u_i^{t-1}, y_i^{t-1}, a^{t-1}) P^1(u_i^{t-1}, y_i^{t-1}, a_{-i}^{t-1} | l_i^t) \tag{4}$$

and

$$I_i^2(u_i^t | l_i^t) = \sum_{u_i^{t-1}, y_i^{t-1}, a_{-i}^{t-1}} P^2(u_i^t | x_i^{t-1}, u_i^{t-1}, y_i^{t-1}, a^{t-1}) P^2(u_i^{t-1}, y_i^{t-1}, a_{-i}^{t-1} | l_i^t). \tag{5}$$

First we see that, because $\langle x_i^{t-1}, u_i^{t-1}, y_i^{t-1} \rangle$ fully determines the Markov state s^{t-1}, the first term in the summation can be computed from the environment's transition function, and thus

$$\begin{aligned} P^1(u_i^t | x_i^{t-1}, u_i^{t-1}, y_i^{t-1}, a^{t-1}) =& P^2(u_i^t | x_i^{t-1}, u_i^{t-1}, y_i^{t-1}, a^{t-1}) \\ =& \sum_{s^t} \mathbb{1}(u^t, s^t) T(s^t | s^{t-1}, a^{t-1}), \end{aligned} \tag{6}$$

where $\mathbb{1}(u^t, s^t)$ is an indicator function that determines if the state s^t is feasible in the context of u^t.

Further, we know that

$$P^1(u_i^{t-1}, y_i^{t-1}, a_{-i}^{t-1} | l_i^t) = \sum_{h_{-i}^{t-1}} \pi_{-i}(a_{-i}^{t-1} | h_{-i}^{t-1}) P^1(u_i^{t-1} y_i^{t-1}, h_{-i}^{t-1} | l_i^t) \tag{7}$$

$$P^2(u_i^{t-1}, y_i^{t-1}, a_{-i}^{t-1} | l^t) = \sum_{h_{-i}^{t-1}} \pi_{-i}(a_{-i}^{t-1} | h_{-i}^{t-1}) P^2(u_i^{t-1} y_i^{t-1}, h_{-i}^{t-1} | l_i^t) \tag{8}$$

where $P^1(u_i^{t-1} y_i^{t-1}, h_{-i}^{t-1} | l^t)$ and $P^2(u_i^{t-1} y_i^{t-1}, h_{-i}^{t-1} | l^t)$ can be computed recursively as

$$\begin{aligned} &P^1(u_i^{t-1}, y_i^{t-1}, h_{-i}^{t-1} | l_i^t) = \\ &\qquad \sum_{h_{-i}^{t-2}, o_{-i}^{t-1}} O(o_{-i}^{t-1} | x_i^{t-1}, u_i^{t-1}, y_i^{t-1}) \pi_{-i}(a_{-i}^{t-2} | h_{-i}^{t-2}) P^1(u_i^{t-1}, y_i^{t-1}, h_{-i}^{t-2} | l_i^t), \end{aligned} \tag{9}$$

and

$$\begin{aligned} &P^2(u_i^{t-1}, y_i^{t-1}, h_{-i}^{t-1} | l_i^t) = \\ &\qquad \sum_{h_{-i}^{t-2}, o_{-i}^{t-1}} O(o_{-i}^{t-1} | x_i^{t-1}, u_i^{t-1}, y_i^{t-1}) \pi_{-i}(a_{-i}^{t-2} | h_{-i}^{t-2}) P^2(u_i^{t-1}, y_i^{t-1}, h_{-i}^{t-2} | l_i^t), \end{aligned} \tag{10}$$

with $h_{-i}^{t-1} = \langle h_{-i,t-2}, a_{-i,t-2}, o_{-i}^{t-1} \rangle$. Then, if we further unroll equations (9) and (10) up to timestep 0, we see that all probability distributions in both cases are equivalent and we reach a contradiction. Hence,

$$I_i^1(u_i^t | l_i^t) = I_i^2(u_i^t | l_i^t) \tag{11}$$

□

Proposition 1. *The space of joint policies $\Pi = \times_{i \in N} \Pi_i$ is necessarily greater than or equal to the space of joint influences $\Psi = \times_{i \in N} \Psi_i$, $|\Pi| \geq |\Psi|$. Moreover, there exist local-form fPOSGs for which the inequality is strict.*

Proof. From Proposition 1 it follows that the space of joint influences Ψ is at most as large as the space of joint policies Π, $|\Psi| \not> |\Pi|$. Hence, we just need to show that in some cases Π is strictly greater than Ψ, $|\Pi| > |\Psi|$. □

A clear example is that where each agent's local region X_i is independent of the other agents' policies π_{-i} (Becker et al., 2003). That is, the actions of other agents a_{-i} have no effect on agent i's local state transitions. From the definition of IALM (Definition 3) we know that, in our setting, a_{-i} can only affect the local state transitions through u_i. Therefore, for the local transitions to be independent the following should hold

$$P(u_i^t|x_i^{t-1}, u_i^{t-1}, y_i^{t-1}, a^{t-1}) = P(u_i^t|x_i^{t-1}, u_i^{t-1}, y_i^{t-1}, a_i^{t-1}) \tag{12}$$

The equation above reflects that only agent i can affect u_i^t. Thus, in the event of local transition independence, we have that

$$\forall i \in N : \exists! I_i^*(u_i^t|l_i^t) \in \Psi_i : \forall \pi \in \Pi \left(I_i(u_i^t|l_i^t, \pi) = I_i^*(u_i^t|l_i^t)\right) \tag{13}$$

That is, for any joint policy $\pi \in \Pi$ there is a unique influence distribution $I_i^* \in \Psi_i$ for every agent $i \in N$, and thus, in this particular case, $|\Pi| \gg |\Psi| = 1$.

To prove Lemma 2 we will use the following lemma.

Lemma 3. *Let $I_i^1(u_i^t|l_i^t)$ and $I_i^2(u_i^t|l_i^t)$ be two different influence distributions with M_i^1 and M_i^2 being the IALMs induced by each of them respectively. Moreover, let $P^1(h_i^{t+1}|h_i^t, a_i^t)$ and $P^2(h_i^{t+1}|h_i^t, a_i^t)$ denote the resulting local AOH transitions for M_i^1 and M_i^2 respectively. The following inequality holds*

$$\sum_{x_i^{t+1}} \left|P^1(h_i^{t+1}|h_i^t, a_i^t) - P^2(h_i^{t+1}|h_i^t, a_i^t)\right| \leq \sum_{l_i^t, u_i^t} P(l_i^t|h_i^t) \left|I^1(u_i^t|l_i^t) - I^2(u_i^t|l_i^t)\right| \quad \forall h_i^t, a_i^t \tag{14}$$

Proof.

$$\begin{aligned}
&\sum_{h_i^{t+1}} \left|P^1(h_i^{t+1}|h_i^t, a_i^t) - P^2(h_i^{t+1}|h_i^t, a_i^t)\right| \\
=&\sum_{o_i^{t+1}} \Big| \sum_{x_i^{t+1}} O_i(o_i^{t+1}|x_i^{t+1}) \sum_{u_i^t} \dot{T}_i(x_i^{t+1}|x_i^t, u_i^t, a_i^t) \sum_{l_i^t} I^1(u_i^t|l_i^t) P(l_i^t|h_i^t) \\
&- \sum_{x_i^{t+1}} O_i(o_i^{t+1}|x_i^{t+1}) \sum_{u_i^t} \dot{T}_i(x_i^{t+1}|x_i^t, u_i^t, a_i^t) \sum_{l_i^t} I^2(u_i^t|l_i^t) P(l_i^t|h_i^t) \Big| \\
=&\sum_{o_i^{t+1}} \Big| \sum_{x_i^{t+1}} O_i(o_i^{t+1}|x_i^{t+1}) \sum_{u_i^t} \dot{T}_i(x_i^{t+1}|x_i^t, u_i^t, a_i^t) \sum_{l_i^t} P(l_i^t|h_i^t) \left[I^1(u_i^t|l_i^t) - I^2(u_i^t|l_i^t)\right] \Big| \\
=&\Big| \sum_{l_i^t, u_i^t} P(l_i^t|h_i^t) \left[I^1(u_i^t|l_i^t) - I^2(u_i^t|l_i^t)\right] \Big| \\
\leq&\sum_{l_i^t, u_i^t} P(l_i^t|h_i^t) \left|I^1(u_i^t|l_i^t) - I^2(u_i^t|l_i^t)\right|
\end{aligned} \tag{15}$$

□

Lemma 2. *Let M_i^1 and M_i^2 be two IALMS differing only on their influence distributions $I_i^1(u_i^t|l_i^t)$ and $I_i^2(u_i^t|l_i^t)$. Let $Q_{M_i^1}^{\pi_i}$ and $Q_{M_i^2}^{\pi_i}$ be the value functions induced by M_i^1 and M_i^2 for the same π_i. If I_i^1 and I_i^2 satisfy*

$$\sum_{l_i^t, u_i^t} P(l_i^t|h_i^t) \left|I_i^1(u_i^t|l_i^t) - I_i^2(u_i^t|l_i^t)\right| \leq \xi, \text{ then } \left|Q_{M_i^1}^{\pi_i}(h_i^t, a_i^t) - Q_{M_i^2}^{\pi_i}(h_i^t, a_i^t)\right| \leq \bar{R}\frac{(H-t)(H-t+1)}{2}\xi \tag{2}$$

for all π_i, h_i^t, and a_i^t, where H is the horizon and $\bar{R} = ||R||_\infty$

Proof. This is a special case of the simulation lemma (Kearns and Singh, 2002). We have that the set of local states and actions is the same for both IALMs. Moreover, the reward function is also the same $R^1(x_t, a_t) = R^2(x_t, a_t)$.

$$\left|Q^{\pi_i}_{M_i^1}(h_i^t,a_i^t)-Q^{\pi_i}_{M_i^2}(h_i^t,a_i^t)\right| = \Bigg|\sum_{x_i^t} P(x_i^t|h_i^t)R(x_i^t,a_i^t) + \sum_{h_i^{t+1},a_i^{t+1}} P^1(h_i^{t+1}|h_i^t,a_i^t)\pi_i(a_i^{t+1}|h_i^{t+1})Q^{\pi_i}_{M_i^1}(h_i^{t+1},a_i^{t+1}) - \sum_{x_i^t} P(x_i^t|h_i^t)R(x_i^t,a_i^t) - \sum_{h_i^{t+1},a_i^{t+1}} P^2(h_i^{t+1}|h_i^t,a_i^t)\pi_i(a_i^{t+1}|h_i^{t+1})Q^{\pi_i}_{M_i^2}(h_i^{t+1},a_i^{t+1})\Bigg|, \tag{16}$$

where $P^1(h_i^{t+1}|h_i^t,a_i^t)$ and $P^2(h_i^{t+1}|h_i^t,a_i^t)$ are the AOH transitions induced by I^1 and I^2 respectively.

$$\begin{aligned}\left|Q^{\pi_i}_{M_i^1}(h_i^t,a_i^t)-Q^{\pi_i}_{M_i^2}(h_i^t,a_i^t)\right| &= \Bigg|\sum_{h_i^{t+1},a_i^{t+1}} \pi_i(a_i^{t+1}|h_i^{t+1})\Big[\\ &\quad P^1(h_i^{t+1}|h_i^t,a_i^t)Q^{\pi_i}_{M_i^1}(h_i^{t+1},a_i^{t+1}) - P^2(h_i^{t+1}|h_i^t,a_i^t)Q^{\pi_i}_{M_i^2}(h_i^{t+1},a_i^{t+1})\Big]\Bigg| \\ &= \Bigg|\sum_{h_i^{t+1},a_i^{t+1}} \pi_i(a_i^{t+1}|h_i^{t+1})\Big[\\ &\quad P^1(h_i^{t+1}|h_i^t,a_i^t)Q^{\pi_i}_{M_i^1}(h_i^{t+1},a_i^{t+1}) - P^2(h_i^{t+1}|h_i^t,a_i^t)Q^{\pi_i}_{M_i^1}(h_i^{t+1},a_i^{t+1}) \\ &\quad + P^2(h_i^{t+1}|h_i^t,a_i^t)Q^{\pi_i}_{M_i^1}(h_i^{t+1},a_i^{t+1}) - P^2(h_i^{t+1}|h_i^t,a_i^t)Q^{\pi_i}_{M_i^2}(h_i^{t+1},a_i^{t+1})\Big]\Bigg| \\ &\leq \Bigg|\bar{R}(H-t)\sum_{h_i^{t+1}}\left(P^1(h_i^{t+1}|h_i^t,a_i^t)-P^2(h_i^{t+1}|h_i^t,a_i^t)\right) \\ &\quad + \sum_{h_i^{t+1},a_i^{t+1}} \pi_i(a_i^{t+1}|h_i^{t+1})P^2(h_i^{t+1}|h_i^t,a_i^t)\Big[Q^{\pi_i}_{M_i^1}(h_i^{t+1},a_i^{t+1}) - Q^{\pi_i}_{M_i^2}(h_i^{t+1},a_i^{t+1})\Big]\Bigg|\end{aligned} \tag{17}$$

since $Q^{\pi_i}_{M_i^1}(h_i^{t+1}) \leq \bar{R}(H-t)$. Then, from Lemma 3 we know that

$$\sum_{h_i^{t+1}}\left(P^1(h_i^{t+1}|h_i^t,a_i^t)-P^2(h_i^{t+1}|h_i^t,a_i^t)\right) \leq \sum_{l_i^t,u_i^t} P(l_i^t|h_i^t)\left|I^1(u_i^t|l_i^t)-I^2(u_i^t|l_i^t)\right| \leq \xi \quad \forall h_i^t, a_i^t. \tag{18}$$

Hence,

$$\left|Q^{\pi_i}_{M_i^1}(h_i^t,a_i^t)-Q^{\pi_i}_{M_i^2}(h_i^t,a_i^t)\right| \leq \sum_{k=t}^{H}\bar{R}(H-k)\xi = \bar{R}\frac{(H-t)(H-t+1)}{2}\xi. \tag{19}$$

□

Theorem 1. *Let M_i^1 and M_i^2 be two IALMS differing only on their influence distributions $I_i^1(u_i^t|l_i^t)$ and $I_i^2(u_i^t|l_i^t)$. M_i^1 and M_i^2 induce the same optimal policy π^* if, for some Δ,*

$$Q^{\pi_i^*}_{M_i^1}(h_i^t,\bar{a}_i^t) - Q^{\pi_i^*}_{M_i^1}(h_i^t,\hat{a}_i^t) > 2\Delta \quad \forall h_i^t, \hat{a}_i^t \neq \bar{a}_i^t \text{ with } \left|Q^{\pi_i}_{M_i^1}(h_i^t,a_i^t)-Q^{\pi_i}_{M_i^2}(h_i^t,a_i^t)\right| \leq \Delta \quad \forall h_i^t, a_i^t, \pi_i, \tag{3}$$

where $\bar{a}_i^t = \arg\max_{a_i^t} Q^{\pi_i^*}_{M_i^1}(h_i^t,a_i^t)$

Proof. We will prove it by contradiction. Let us assume there is a policy π^* that is optimal for M_i^1 but not for M_i^2. This implies that, for some h_i^t, there is at least one action $\hat{a}_i^t \neq \bar{a}_i^t$ for which

$$Q^{\pi^*}_{M_i^2}(h_i^t,\bar{a}_i^t) < Q^{\pi^*}_{M_i^2}(h_i^t,\hat{a}_i^t) \tag{20}$$

Then, because the maximum gap between $Q_{M_i^1}$ and $Q_{M_i^2}$ is Δ,

$$Q^{\pi^*}_{M_i^1}(h_i^t, \bar{a}_i^t) - \Delta \leq Q^{\pi^*}_{M_i^2}(h_i^t, \bar{a}_i^t) < Q^{\pi^*}_{M_i^2}(h_i^t, \hat{a}_i^t) \leq Q^{\pi^*}_{M_i^1}(h_i^t, \hat{a}_i^t) + \Delta. \tag{21}$$

Therefore, we have

$$Q^{\pi^*}_{M_i^1}(h_i^t, \bar{a}_i^t) - Q^{\pi^*}_{M_i^1}(h_i^t, \hat{a}_i^t) < 2\Delta, \tag{22}$$

which contradicts the statement

$$Q^{\pi^*}_{M_i^1}(h_i^t, \bar{a}_i^t) - Q^{\pi^*}_{M_i^1}(h_i^t, a_i^t) > 2\Delta \quad \forall h_i^t, a_i^t \tag{23}$$

□

B Further Related Work

There is a sizeable body of literature that concentrates on the non-stationarity issues arising from having multiple agents learning simultaneously in the same environment (Laurent et al., 2011; Hernandez-Leal et al., 2017). Although often times the problem can be simply ignored with virtually no consequences for the agents' performance (Tan, 1993), in general, disregarding changes in the other agents' policies, and assuming individual Q-values to be stationary, can have a catastrophic effect on convergence (Claus and Boutilier, 1998).

The problem of non-stationarity becomes even more severe in the Dec-POMDP setting (Oliehoek and Amato, 2016) since policy changes may not be immediately evident from each agent's AOH. To compensate for this Raileanu et al. (2018) and Rabinowitz et al. (2018) explicitly train models that predict the other agents' goals and behaviors. In contrast, Foerster et al. (2018a) add an extra term to the learning objective that is meant to predict the other agents' parameter updates. This approach is empirically shown to encourage cooperation in general-sum games. In order to better approximate the value function, several works have studied the use of additional information during training to inform each individual agent of changes in the other agents' policies, leading to the ubiquitous centralized-training decentralized-execution (CTDE) paradigm. The works by Lowe et al. (2017) and Foerster et al. (2018b) exploit this by training a single centralized critic that takes as input the true state and joint action of all the agents. This critic is then used to update the policies of all agents following the actor-critic policy gradient update (Konda and Tsitsiklis, 1999). Even though the use of additional information to augment the critic may help reduce bias in the value estimates, the idea lacks any theoretical guarantees and has been shown to produce the same policy gradient in expectation as those produced by multiple independent critics (Lyu et al., 2021). Moreover, according to Lyu et al., naively augmenting the critic with all other agents' actions and observations can heavily increase the variance of the policy gradients. Both results, however, assume that the critics have converged to the true on-policy value estimates. The authors do admit that, in practice, critics are often used even when they have not yet converged. In such situations, centralized critics might provide more stable policy updates since they are better equipped to follow the true non-stationary Q values. Following a similar perspective, the concurrent work by Spooner et al. (2021) tries to reduce variance by using a per-agent baseline function that removes from the policy gradient the contributions to the joint value estimates of those agents that are conditionally independent, thus effectively providing the agent with more stable updates. The works by de Witt et al. (2020) and Yu et al. (2021) show that the vanilla PPO algorithm (Schulman et al., 2017) works already quite well on several multi-agent tasks. Yu et al. attribute the positive empirical results to the clipping parameter ϵ, which prevents individual policies from changing drastically, and in turn reduces the problem of non-stationarity. Li et al. (2021) further analyze this idea and propose a method to estimate the joint policy divergence, which is then used as a constraint in the optimization objective.

C Algorithms

The two algorithms below describe how to generate the datasets $\{D_i\}_{i \in N}$ with the GS (Algorithm 2) and how to simulate trajectories with each of the IALS (Algorithm 3).

Algorithm 2 Collect datasets $\{D_i\}_{i \in N}$ with GS

Input: T, $\{\dot{O}_i\}_{i \in N}$, $\pi^0 = \{\pi_i^0\}_{i \in N}$ ▷ Global simulator, observation functions, and joint policy
for $n \in \langle 0, ..., N/T \rangle$ **do**
  $s^0 \leftarrow$ reset ▷ Reset initial state
  $\{x_i^0\}_{i \in N} \leftarrow s^0$ ▷ Extract local states from global state
  $\{l_i^0 \leftarrow x_i^0\}_{i \in N}$ ▷ Initialize each agent's ALSH with initial local state
  $\{o_i^0 \sim O_i(\cdot \mid x^0)\}_{i \in N}$ ▷ Sample each agent's observation from O_i
  $\{h_i^0 \leftarrow o_i^0\}_{i \in N}$ ▷ Initialize each agent's AOH with initial observation
  for $t \in \langle 0, ..., T \rangle$ **do**
    $\{u_i^0\}_{i \in N} \leftarrow s^0$ ▷ Extract each agent's influence sources from global state
    $\{D_i \leftarrow (l_i^t, u_i^t)\}_{i \in N}$ ▷ Append ALSH-influence-source pair to the datasets
    $\{a_i^t \sim \pi(\cdot \mid h_i^t)\}_{i \in N}$ ▷ Sample each agent's action from π_i
    $s^{t+1} \sim T(\cdot \mid s^t, a^t = \{a_i^t\}_{i \in N})$ ▷ Sample next state from GS
    $\{x_i^{t+1}\}_{i \in N} \leftarrow s^{t+1}$ ▷ Extract local states from global state
    $\{l_i^{t+1} \leftarrow \langle a_i^t, x_i^{t+1} \rangle\}_{i \in N}$ ▷ Append action-local-state pairs to each agent's ALSH
    $\{o_i^{t+1} \sim \dot{O}_i(\cdot \mid x^{t+1})\}_{i \in N}$ ▷ Sample each agent's observation from $\dot{O}_i$
    $\{h_i^{t+1} \leftarrow \langle a_i^t, o_i^{t+1} \rangle\}_{i \in N}$ ▷ Append actions-observation pairs to each agent's AOH
  end for
end for

Algorithm 3 Simulate agent i's trajectory with IALS

Input: $\dot{T}_i, \dot{R}_i, \dot{O}_i, \pi_i, \hat{I}_{\theta_i}$ ▷ local simulator, local reward and observation functions, policy, AIP
$x_i^0 \leftarrow$ reset ▷ Reset initial state
$o_i^0 \sim \dot{O}_i(\cdot | x_i^0)$ ▷ Sample observation from $\dot{O}_i$
$h_i^0 \leftarrow o_i^0$ ▷ Initialize AOH with initial observation
for $t \in \langle 0, ..., T \rangle$ **do**
  $a_i^t \sim \pi(\cdot \mid h_i^t)$ ▷ Sample action
  $\dot{R}_i(x_i^t, a_i^t)$ ▷ Compute reward
  $u_i^t \sim \hat{I}_{\theta_i}(\cdot \mid l_i^t)$ ▷ Sample influence sources from AIP
  $x_i^{t+1} \sim \dot{T}(\cdot \mid x_i^t, a_i^t, u_i^t)$ ▷ Sample next local state from LS
  $l_i^{t+1} \leftarrow \langle a_i^t, x_i^{t+1} \rangle$ ▷ Append action-local-state pair to ALSH
  $o_i^{t+1} \sim \dot{O}_i(\cdot \mid x_i^{t+1})$ ▷ Sample observation from O
  $h_i^{t+1} \leftarrow \langle a_i^t, o_i^{t+1} \rangle$ ▷ Append action-observation pair to AOH
end for

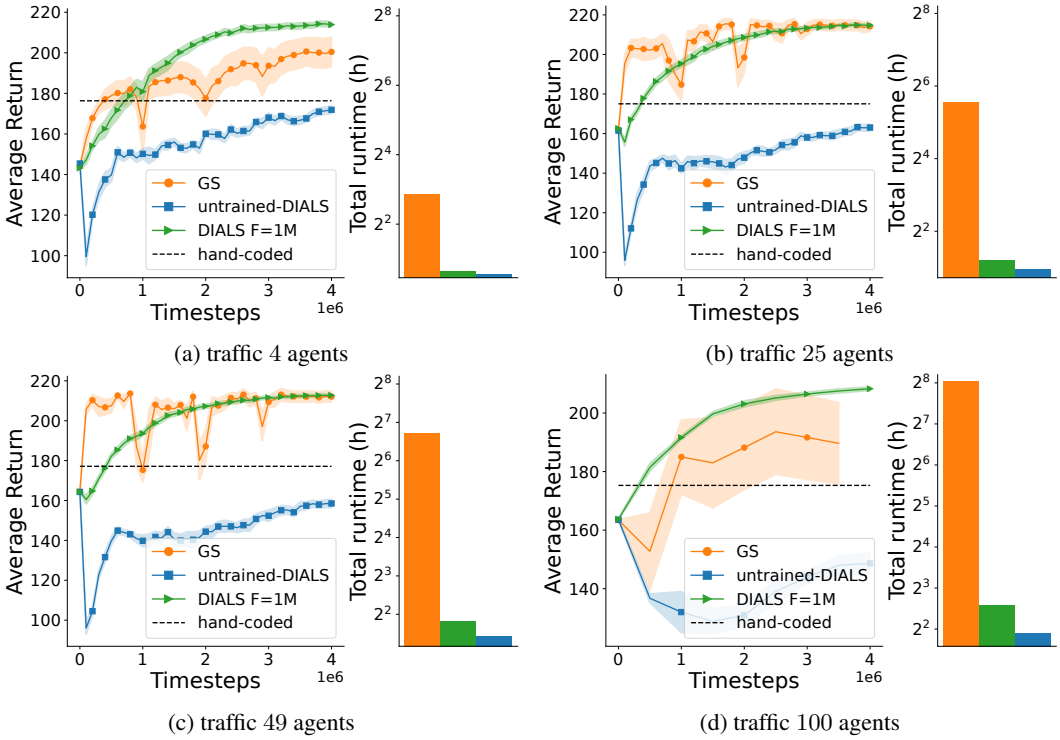
D Results

D.1 DIALS vs GS

The plots in Figures 5 and 6 show the learning curves of agents trained with the GS, DIALS, and untrained-DIALS on the 4 variants of the traffic and warehouse environments (4, 25, 49, and 100 agents). The bar plots show the total runtime of training for 4M timesteps with the three simulators. Shaded areas indicate the standard error of the mean.

The orange curves in Figures 5d and 6d stop at 3.5M and 2M timesteps respectively. This is because the maximum execution time allowed by our computer cluster is 1 week, and training 100 agents with the GS takes longer. A breakdown of the runtimes for the three simulators is provided in Appendix G. Note that the runtime measurements were made on the only machine in our computer cluster with more than 100 CPUs. This is so that it would fit DIALS when training on the 100-agent variants. However, the experiments that required less than 100 CPUs were ran on different machines with different CPUs.

The bar plots indicate that DIALS is computationally more efficient and scales much better than the GS. Note that the y axis is in $\log_2$ scale. Moreover, agents trained with DIALS seem to converge steadily towards similar high-performing policies in both environments, while agents trained with



(a) traffic 4 agents

(b) traffic 25 agents

(c) traffic 49 agents

(d) traffic 100 agents

Figure 5: **Left (a), (b), (c), and (d):** Average return as a function of the number of timesteps with GS, DIALS $F = 1$M, and untrained-DIALS on the traffic environment. **Right (a), (b), (c), and (d):** Total runtime of training for 4M timesteps, y-axis is in $\log_2$ scale.

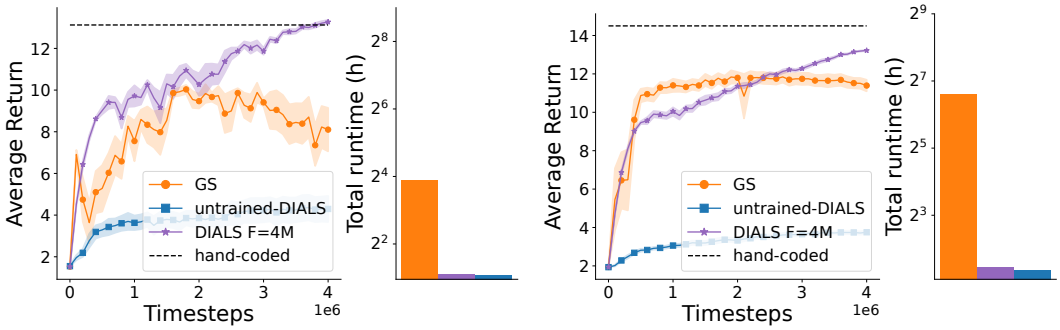
the GS suffer frequent performance drops and often get stuck in local minima. This is evidenced by the oscillations in the orange curves, and the poor mean episodic reward and large standard errors compared to the green (traffic) and purple (warehouse) curves. The plots also reveal that estimating the influence distributions correctly is important, as indicated by the large gap between DIALS and untrained-DIALS in both environments.

It is worth noting that the gap between GS and DIALS is larger in the warehouse (Figure 6) than in the traffic environment (Figure 5). We posit that this is because, in the warehouse environment, agents are more strongly coupled. To see this imagine that, by random chance during training, a robot starts favoring items from one shelf over the three others. The robot's neighbors might exploit this and start collecting items from the unattended shelves. However, as soon as this first robot changes its policy, and starts collecting items more evenly from all four shelves, the neighbor robots will experience a sudden drop in the value of their policies, which can have catastrophic effects on the learning dynamics. With the DIALS, however, agents are trained on separate simulators and only become aware of changes in the joint policy when the AIPs are retrained. This prevents them from constantly co-adapting to one another. This is in line with our discussion in Section 4.3.

D.2 AIPs training frequency

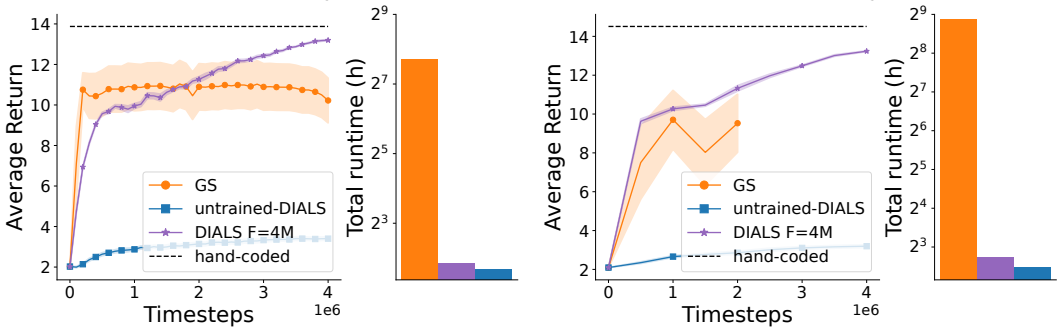
The two plots on the left of Figures 7 and 8 show a comparison of the agents' average return as a function of runtime for different values of the AIPs training frequency parameter F (100K, 500K, 1M, and 4M timesteps). For ease of visualization, since DIALS $F = 500$K, $F = 1$M, and $F = 4$M take shorter to finish than DIALS $F = 100$K, the red, green, and purple curves are extended by dotted horizontal lines. Due to computational limitations, we ran these experiments only on the 4, 25, and 49-agent variants of the two environments. We then chose the best performing values for F ($F = 1$M for traffic and $F = 4$M for warehouse), and used those to ran DIALS on the environments with 100 agents.

In the traffic domain, the gap between the green and the purple curve (Figure 7) suggests that it is important to retrain the AIPs at least every 1M timesteps, such that agents become aware of changes



(a) Warehouse 4 agents

(b) Warehouse 25 agents



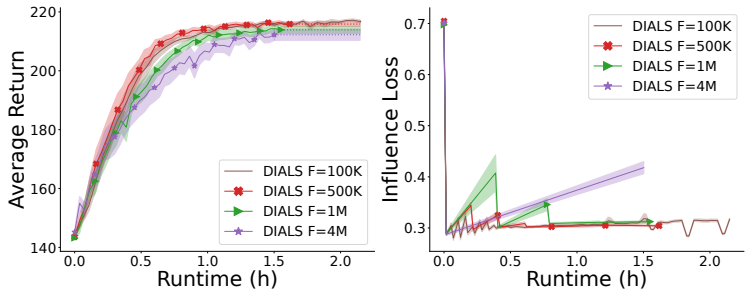
(c) Warehouse 49 agents

(d) Warehouse 100 agents

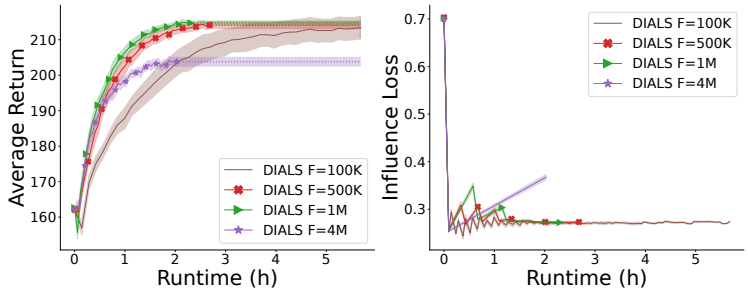
Figure 6: **Left (a), (b), (c), and (d):** Average return as a function of the number of timesteps with GS, DIALS $F = 1$M, and untrained-DIALS on the warehouse environment. **Right (a), (b), (c), and (d):** Total runtime of training for 4M timesteps, y-axis is in $\log_2$ scale.

in the other agents' policies. This is consistent on all the three variants (Figures 7a, 7b, and 7c). In contrast, in the warehouse domain (Figure 8), we see that training the AIPs only once at the beginning (DIALS $F = 4$M) is sufficient (Figures 8a, 8b, and 8c). In fact, as indicated by the gap between the brown and the rest of the curves, updating the AIPs too frequently (DIALS $F = 100$K), aside from increasing the runtimes, seems detrimental to the agents' performance. This is consistent with our hypothesis in Section 4.3: "by not updating the AIPs too frequently, we get a biased but otherwise more consistent learning signal that the agents can rely on to improve their policies."

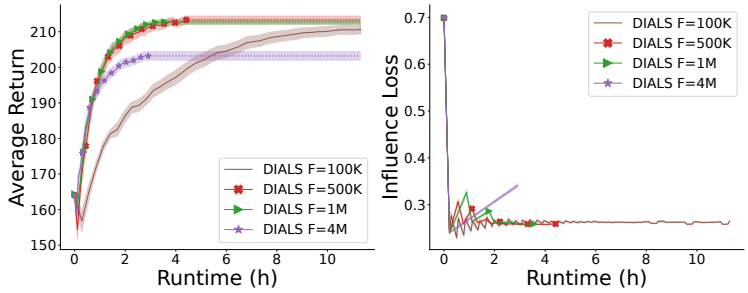
The plots on the right of Figures 7 and 8 show the average cross-entropy (CE) loss of the AIPs evaluated on trajectories sampled from the GS. As explained in Section 4 since all agents learn simultaneously, the influence distributions $\{I(u_i^t|l_i^t)\}_{i\in N}$ are non-stationary. For this reason, we see that the CE loss changes as the policies of the other agents are updated. We can also see how the CE loss decreases when the AIPs are retrained, which happens more or less frequently depending on the hyperparameter F. Note that the CE not only measures the distance between the two probability distributions but also the absolute entropy. In the warehouse domain (Figure 8), the neighbor robots' locations become more predictable (lower entropy) as their policies improve. This explains why the CE loss decreases even though the AIPs are not updated. Also note that, in the warehouse environment (Figure 8), even though by the end of training DIALS $F = 4$M is highly inaccurate, as evidenced by the gap between the purple and the other curves, it is still good enough to train policies that match the performance of those trained with DIALS $F = 500$K and $F = 1$M. This is in line with our results in Section 4: "Multiple influence distributions may induce the same optimal policy."



(a) Traffic 4 agents



(b) Traffic 25 agents



(c) Traffic 49 agents

Figure 7: **Left (a), (b), and (c):** Learning curves for different values of F on the 4, 25, and 49 agent versions of the traffic environment. **Right (a), (b), and (c):** CE loss of the AIPs as a function of runtime.

E Implementation Details

E.1 Approximate Influence Predictors

Due to the sequential nature of the problem, rather than feeding the full past history every time we make a prediction, we use a recurrent neural network (RNN) (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) and process observations one at a time,

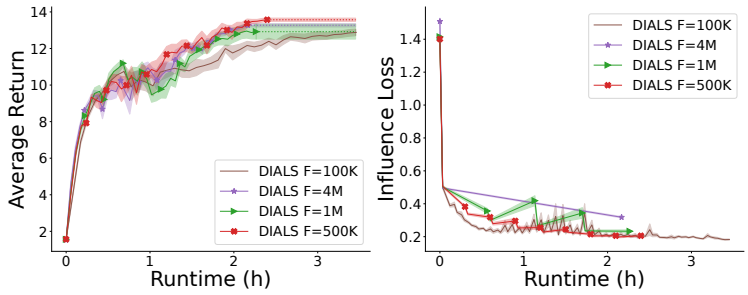
$$P(u_t|l_t) \approx \hat{I}_\theta(u_t|\hat{h}_{t-1}, o_t) = F_{\text{rnn}}(\hat{h}_{t-1}, o_t, u_t), \tag{24}$$

where we use $\hat{h}$ to indicate that the history h is embedded in the RNN's internal memory.

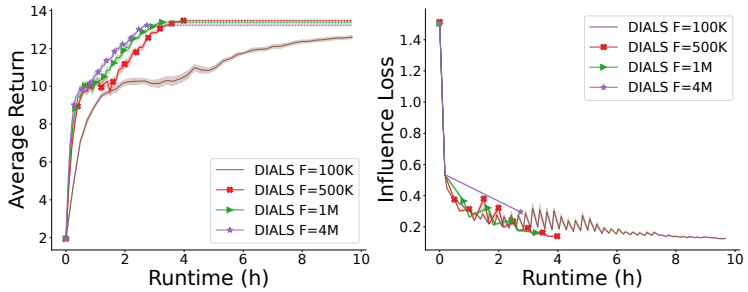
Given that we generally have multiple influence sources $u_t = \langle u_t^1 \dots u_t^M \rangle$, we need to fit M separate models $\hat{I}_{\theta_m}$ to predict each of the M influence sources. In practice, to reduce the computational cost, we can have a single network with a common representation module for all influence sources and output their probability distributions using M separate heads. This representation assumes that the influence sources are independent of one another,

$$I(u_t|l_t) = \prod_{m=0}^{M} P(u_t^m|l_t), \tag{25}$$

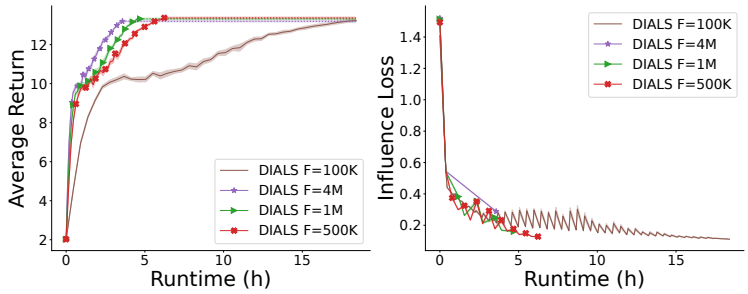
which is true for the two domains we study in this paper.



(a) Warehouse 4 agents



(b) Warehouse 25 agents



(c) Warehouse 49 agents

Figure 8: **Left (a), (b), (c), and (d):** Average return as a function of the number of timesteps with GS, DIALS $F = 1$M, and untrained-DIALS on the warehouse environment. **Right (a), (b), (c), and (d):** Total runtime of training for 4M timesteps, y-axis is in $\log_2$ scale.

Finally, although according to the POMDP framework we should condition the AIPs on the full AOH, in many domains, one can exploit the structure of the transitions function to find a subset of variables in the AOH that is sufficient to predict the next observation. This subset is known as the d-separating set (Oliehoek et al., 2021), and as shown in Suau et al. (2022a) conditioning the AIPs on this rather than the full AOH can ease the task of approximating the influence distribution.

E.2 Local regions

When choosing the local regions to build the simulators, the only restriction in terms of size is that these should contain all the necessary information to compute local observations and rewards. In our experiments, we use one simulator per agent since, given that the simulators run in parallel, this is the most computationally efficient way of factorizing the environment. Yet, in certain applications, due to hardware limitations (e.g. not enough CPUs or memory available), it might be necessary to partition the environment into fewer local regions than the number of agents in the environment. Moreover, in some environments (including the two we explore here) better results may be obtained by grouping some of them together in the same simulator. In fact, one could potentially treat the agents in the same group/simulator as a single agent and train a policy to control all of them simultaneously. Note, however, that this is orthogonal to our work as we are mainly concerned with computational speedups.

F Simulators

Figure 9 shows two screenshots of the global simulator (GS) for the traffic (left) and warehouse (right) environments with 25 agents each. Figure 10 shows two screenshots of the local simulator (LS) for the traffic (left) and warehouse environments (right). Since all local regions are the same (i.e. $\dot{T}_i$, $\dot{R}_i$, and $\dot{O}_i$ do not change) in the two environments, we use the same LS for all of them. However, because depending on where these are located they are influenced differently by the rest of the system, we train separate AIPs, $\{\hat{I}_{\theta_i}\}_{i \in N}$, for each of them. Note that, we chose the local regions to be the same for simplicity. However, the method can readily be applied to environments with different local transition dynamics $\dot{T}_i$, different local observations $\dot{O}_i$, and/or different local rewards $\dot{R}_i$ for every agent $i \in N$.

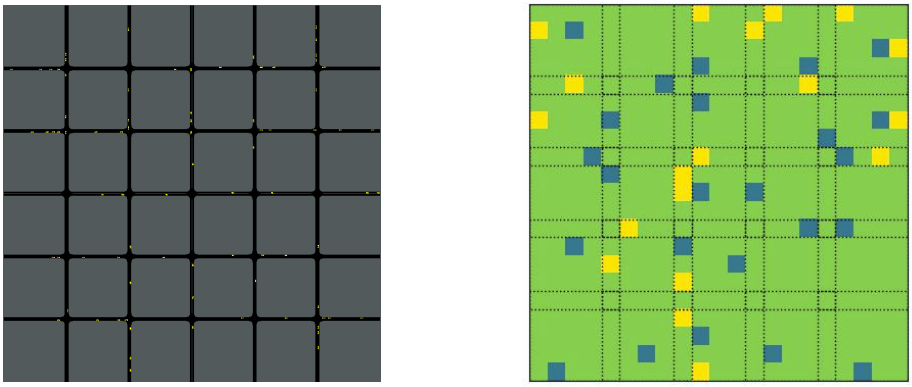


Figure 9: A screenshot of the global simulators for the 25-agent variants of the traffic control (left) and warehouse (right) environments

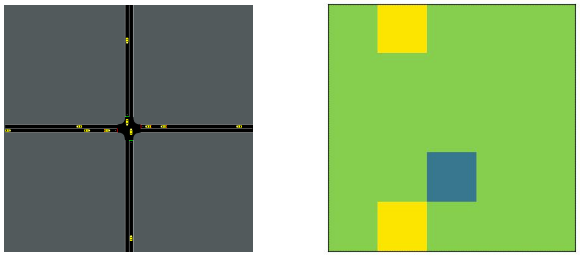


Figure 10: A screenshot of the local simulators for the traffic (left) and warehouse (right) environments. Since all local regions are the same in the two environments, we use the same LS for all of them.

G Runtimes

The two tables below show a breakdown of the runtimes for the two environments and the three simulators. These were measured on a machine with 128 CPUs of the type AMD EPYC 7452 32-Core Processor. We used this machine for all our measurements because it is the only one in our computer cluster that can fit DIALS when training on the 100-agent variants of the environments. However, the experiments that required less than 100 CPUs were actually run on different machines.

Table 1: Runtimes for the traffic control environment

	Agents training (h)				Data collection + influence training (h)				Total (h)			
Number of agents	2	25	49	100	2	25	49	100	2	25	49	100
GS	7.24	46.96	105.41	261.06	-	-	-	-	7.24	46.96	105.41	261.06
DIALS F=100K	1.48	1.93	2.70	3.70	0.66	3.74	8.60	22.38	2.14	5.67	11.30	26.08
DIALS F=500K	1.48	1.93	2.70	3.70	0.13	0.75	1.72	4.48	1.61	2.68	4.42	8.18
DIALS F=1M	1.48	1.93	2.70	3.70	0.07	0.37	0.86	2.24	1.55	2.30	3.56	5.94
DIALS F=4M	1.48	1.93	2.70	3.70	0.02	0.09	0.21	0.56	1.50	2.02	2.91	4.26
untrained-DIALS	1.48	1.93	2.70	3.70	-	-	-	-	1.48	1.93	2.70	3.70

Table 2: Runtimes for the warehouse environment

	Agents training (h)				Data collection + influence training (h)				Total (h)			
Number of agents	2	25	49	100	2	25	49	100	2	25	49	100
GS	14.84	97.04	208.18	468.46	-	-	-	-	14.84	97.04	208.18	468.46
DIALS F=100K	2.13	2.56	3.19	5.55	1.32	7.11	15.19	45.45	4.45	9.67	18.38	51.00
DIALS F=500K	2.13	2.56	3.19	5.55	0.26	1.42	3.04	9.09	2.39	3.98	6.23	14.64
DIALS F=1M	2.13	2.56	3.19	5.55	0.13	0.71	1.52	4.54	2.26	3.27	4.71	10.09
DIALS F=4M	2.13	2.56	3.19	5.55	0.03	0.18	0.38	1.13	2.16	2.74	3.57	6.68
untrained-DIALS	2.13	2.56	3.19	5.55	-	-	-	-	2.13	2.56	3.19	5.55

H Memory Usage

The table below shows the peak memory usage of the GS and the DIALS. For the latter we provide the memory usage per process and in total. The memory needed for the GS seems to grow logarithmically with the number of agents, whereas for DIALS the memory usage per process stays relatively constant. However, the total amount of memory needed to run DIALS (aggregate of all processes) increases linearly with the number of agents and is considerably larger than that of the GS.

Table 3: Peak Memory Usage in Megabytes (MB)

Environment		Traffic				Warehouse			
Number of agents		4	25	49	100	4	25	49	100
GS		375.3	392.7	412.5	457.4	339.3	391.8	469.6	607.4
DIALS	Per process	219.5	221.0	225.8	228.7	195.6	201.9	203.7	207.5
	Total	878.0	5525.0	11064.2	22870.0	782.4	5047.5	9981.3	20750.0

I Hyperparameters

The hyperparameters used for the AIPs are reported in Table 4. Since feeding past local states did not seem to improve the performance of the AIPs in the traffic environment we modeled them with FNNs. In contrast, adding the past ALSHs does decrease the CE loss in the warehouse environment, and thus we used GRUs (Cho et al., 2014) instead. The size of the networks was chosen as a compromise between low CE loss and computational efficiency. On the one hand, we need accurate AIPs to properly capture the influence distributions. On the other, we also want them to be small enough such that we can make fast predictions. The hyperparameter named seq. length determines the number of timesteps the GRU is backpropagated. This was chosen to be equal to the horizon such that episodes did not have to be truncated. The rest of the hyperparameters in Table 4, which refer to the training setup for the AIPs, were manually tuned.

Table 4: Hyperparameters for approximate influence predictors (AIPs).

	Architecture	Num. layers	Num. neurons	Seq. length	Learning rate	Dataset size	Batch size	Num. epochs
Traffic	FNN	2	128 and 128	-	1e−4	1e4	128	100
Warehouse	GRU	2	64 and 64	100	1e−4	1e4	32	300

The hyperparmeters used for the policy networks are given in Table 5. We chose again GRUs for the warehouse environment and FNNs for the traffic domain, since feeding the previous AOHs did not seem to improve the agents' performance in the latter. The network size and the sequence length parameter for the GRUs were manually tuned on the smallest scenarios with 4 agents.

Table 5: Hyperparmeters for policy networks.

	Architecture	Num. layers	Num. neurons	Seq. length
Traffic	FNN	2	256 and 128	-
Warehouse	GRU	2	256 and 128	8

As for the hyperparameters specific to PPO (Table 6), we used the same values reported by (Schulman et al., 2017), and only tuned the parameter T, which depends on the rewards and the episode length. T determines for how many timesteps the value function is rollout before computing the value estimates.

Table 6: PPO hyperparameters.

Rollout steps T	16 traffic and 8 warehouse
Learning rate	2.5e-4
Discount γ	0.99
GAE λ	0.95
Memory size	128
Batch size	32
Num. epoch	3
Entropy β	1.0e-2
Clip ϵ	0.1
Value coeff. c_1	1