Influence-Augmented Online Planning for Complex Environments

Jinke He
Department of Intelligent Systems
Delft University of Technology
J.He-4@tudelft.nl

Miguel Suau
Department of Intelligent Systems
Delft University of Technology
M.SuaudeCastro@tudelft.nl

Frans A. Oliehoek
Department of Intelligent Systems
Delft University of Technology
F.A.Oliehoek@tudelft.nl

Abstract

How can we plan efficiently in real time to control an agent in a complex environment that may involve many other agents? While existing sample-based planners have enjoyed empirical success in large POMDPs, their performance heavily relies on a fast simulator. However, real-world scenarios are complex in nature and their simulators are often computationally demanding, which severely limits the performance of online planners. In this work, we propose influence-augmented online planning, a principled method to transform a factored simulator of the entire environment into a local simulator that samples only the state variables that are most relevant to the observation and reward of the planning agent and captures the incoming influence from the rest of the environment using machine learning methods. Our main experimental results show that planning on this less accurate but much faster local simulator with POMCP leads to higher real-time planning performance than planning on the simulator that models the entire environment.

1 Introduction

We consider the online planning setting where we control an agent in a complex environment that is partially observable and may involve many other agents. When the policies of other agents are known, the entire environment can be modeled as a Partially Observable Markov Decision Process (POMDP) (Kaelbling et al., 1998), and traditional online planning approaches can be applied. While sample-based planners like POMCP (Silver and Veness, 2010) have been shown effective for large POMDPs, their performance relies heavily on a fast simulator to perform a vast number of Monte Carlo simulations in a step. However, many real-world scenarios are complex in nature, making simulators that capture the dynamics of the entire environment extremely computationally demanding and hence preventing existing planners from being useful in practice. Towards effective planning in realistic scenarios, this work is motivated by the question: can we significantly speed up a simulator by replacing the part of the environment that is less important with an approximate learned model?

We build on the multi-agent decision making literature that tries to identify compact representations of complex environments for an agent to make optimal decisions (Becker et al., 2003, 2004; Petrik and Zilberstein, 2009; Witwicki and Durfee, 2010). These methods exploit the fact that in many structured domains, only a small set of (state) variables, which we call *local (state) factors*, of the environment directly affects the observation and reward of the agent. The rest of the environment can only impact the agent indirectly through their influence on the local factors. For example, Figure 1a

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

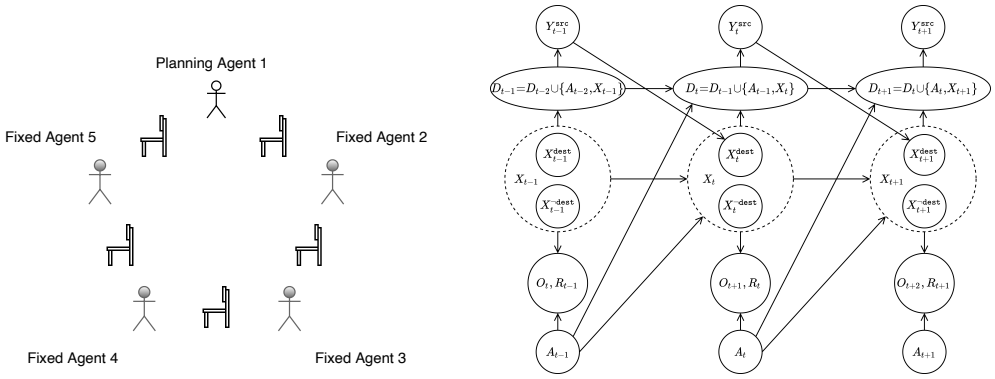


Figure 1: Left: Controlling a single agent in the Grab A Chair game with 4 other agents. Right: Dynamic Bayesian Network for the influence-augmented local model.

shows a game called Grab A Chair, in which there are N agents that, at every time step, need to decide whether they will try to grab the chair on their left or right side. An agent can only secure a chair if that chair is not targeted by the other neighboring agent. At the end of every step, each agent only observes whether it obtains the chair, without knowing the decisions of others. Additionally, there is a noise on observation, i.e., a chance that the agent gets an incorrect observation. In this game, it is clear that to the planning agent, whose goal is to obtain a chair at as many steps as possible, the decisions of neighboring agents 2 and 5 are more important than those of agents 3 and 4 as the former directly determine if the planning agent can secure a chair. In other words, only agents 2 and 5 directly *influence* agent 1's local decision making, while agents 3 and 4 may only do so indirectly.

To utilize this fact, we propose influence-augmented online planning, a principled method that transforms a factored simulator of the entire environment, called *global simulator*, into a faster *influence-augmented local simulator (IALS)*. The IALS simulates only the local factors, and concisely captures the influence of the external factors by predicting only the subset of them, called *source factors*, that directly affect the local factors. Using off-the-shelf supervised learning methods, the influence predictor is learned offline with data collected from the global simulator. Our intuition is that when planning with sample-based planners, the advantage that substantially more simulations can be performed in the IALS may outweigh the simulation inaccuracy caused by approximating the incoming influence. In this paper, we investigate this hypothesis, and show that this approach can indeed lead to improved online planning performance.

In detail, our planning experiments with POMCP show that, by replacing the global simulator with an IALS that learns the incoming influence with a recurrent neural network (RNN), we achieve matching performance while using much less time. More importantly, our real-time online planning experiments show that planning with the less accurate but much faster IALS yields better performance than planning with the global simulator in a complex environment, when the planning time per step is constrained. In addition, we find that learning an accurate influence predictor is more important for good performance when the local planning problem is tightly coupled with the rest of the environment.

2 Background

2.1 POMDP

A Partially Observable Markov Decision Process (POMDP) (Kaelbling et al., 1998) models the interactive process of an agent making decisions and receiving feedback in an environment with limited observation. Formally, a POMDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, b_0, \gamma)$ where $\mathcal{S}$, $\mathcal{A}$, Ω are the set of environment states, actions and observations. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(S)$ determines the distribution over the next state S_{t+1} given the previous state S_t and action A_t, where $\Delta(S)$ denotes the space of probability distributions over S. On transition, the agent receives a reward $R_t \sim \mathcal{R}(S_{t+1}, A_t)$ and a new observation $O_{t+1} \sim \mathcal{O}(S_{t+1}, A_t)$. A policy π is a behavioral strategy that maps an action-observation history $h_t = \{s_0, o_1, \ldots, a_{t-1}, o_t\}$ to a distribution over actions. The

belief state $b_t \in \Delta(S)$ is a sufficient statistic of the history h_t, representing the distribution over S_t conditioned on h_t, with b_0 being the initial belief and known. The value function $V^\pi(h_t)$ measures the expected discounted return from h_t by following π afterwards, $V^\pi(h_t) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | H_t = h_t]$, where $\gamma \in [0, 1]$ is the discount factor, with the optimal value function $V^*(h_t) = \max_\pi V^\pi(h_t)$ measuring the maximally achievable value from h_t. The optimal value of a POMDP $\mathcal{M}$ is defined as $V^*_\mathcal{M} = V^*(b_0) = \max_\pi \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_t]$.

In structured domains, the state space $\mathcal{S}$ of a POMDP can be factorized into a finite set of state variables $\mathcal{S} = \{\mathcal{S}^1, \dots, \mathcal{S}^N\}$, whose conditional independence between each other and the observation and reward variables can be utilized to construct a more compact representation of the POMDP called Dynamic Bayesian Network (DBN) (Boutilier et al., 1999). For convenience, we use the notation S_t to refer to both the set of state variables and the joint random variable over them.

2.2 Sample-based Online Planning in POMDPs

Many real-world decision making problems are so complex that finding a policy that performs well in all situations is not possible. In such cases, online planning methods which aim to find a local policy $\pi(\cdot|h_t)$ that maximizes $V^\pi(h_t)$ when observing a history h_t can lead to better performance. In fully observable case, sample-based planning methods that evaluate actions by performing sample-based lookahead in a simulator have been shown effective for large problems with sample complexity irrelevant to the state space size (Kearns et al., 2002). Monte Carlo Tree Search (MCTS) is a popular family of sample-based planning methods (Coulom, 2006; Kocsis and Szepesvári, 2006; Browne et al., 2012) that implement a highly selective search by building a lookahead tree and focusing the search on the most promising branches during the planning process.

POMCP proposed by Silver and Veness (2010) extends MCTS to large POMDPs, addressing both the curse of dimensionality and the curse of history with Monte Carlo simulation in a generative simulator $\mathcal{G}$ that samples transitions. To avoid the expensive Bayesian belief update, POMCP approximates the belief state with an unweighted particle filter. Similar to MCTS, POMCP maintains a lookahead tree with nodes representing the simulated histories h that follow the real history h_t. To plan for an action, POMCP repeatedly samples states from the particle pool $B(h_t)$ at the root node. By simulating a state to the end, with actions selected by the UCB1 algorithm (Auer et al., 2002) inside the tree and a random policy during the rollout, the visited nodes are updated with the simulated return and the tree is expanded with the first newly encountered history. When the planning terminates, POMCP executes the action a_t with the highest average return and prunes the tree by making the history $h_{t+1}{=}h_t a_t o_{t+1}$ the new root node. Notably, POMCP shares the simulations between tree search and belief update by maintaining a pool of encountered particles in every node during the tree search. This way, when h_{t+1} is made the new root node, $B(h_{t+1})$ becomes the new estimated belief state.

2.3 Influence-Based Abstraction

Influence-Based Abstraction (IBA) (Oliehoek et al., 2012) is a state abstraction method (Li et al., 2006) which abstracts away state variables that do not directly affect the observation and reward of the agent, without a loss in the value. In the following, we provide a brief introduction on IBA and refer interested readers to Oliehoek et al. (2019) for more details.

Given a factored POMDP, which we call the global model $\mathcal{M}_{\texttt{global}} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, b_0, \gamma)$, IBA splits the set of state variables that constitute the state space $\mathcal{S}$ into two disjoint subsets, the set of *local state variables* X that include at least the parent variables of the observation and reward variables and the set of *non-local state variables* $Y = \mathcal{S} \backslash X$.

IBA then defines an influence-augmented local model (IALM) $\mathcal{M}_{\texttt{IALM}}$, where the non-local state variables Y are marginalized out. To define the transition function $\mathcal{T}^{\texttt{IALM}}$ on only the local state variables X, IBA differentiates the local state variables $X^{\texttt{dest}} \subseteq X$ that are directly affected by the non-local state variables, called *influence destination state variables*, from those that are not $X^{\neg\texttt{dest}} = X \backslash X^{\texttt{dest}}$. In addition, IBA defines the non-local state variables $Y^{\texttt{src}} \subseteq Y$ that directly affect X as *influence source state variables*. In other words, the non-local state variables Y influences the local state variables X only through $Y^{\texttt{src}}$ affecting $X^{\texttt{dest}}$ as shown in Figure 1b. Since abstracting away $Y_0, \dots, Y_{t-1}$ creates a dependency of $Y_t^{\texttt{src}}$ on the history of local states and actions, the state $S_t^{\texttt{IALM}}$ needs to include both the local state $X_t = (X_t^{\neg\texttt{dest}}, X_t^{\texttt{dest}})$ and the so-called d-separation set

D_t, which encodes the relevant parts of the local history. Given this, $\mathcal{T}^{\texttt{IALM}}$ is defined as follows:

$$\begin{aligned}
&\mathcal{T}^{\texttt{IALM}}(S_{t+1}^{\texttt{IALM}}|S_t^{\texttt{IALM}}, A_t) = \Pr(X_{t+1}, D_{t+1}|X_t, D_t, A_t) \\
&= \Pr(X_{t+1}^{\neg\texttt{dest}}|X_t, A_t)\mathbb{1}(D_{t+1} = d(X_t, A_t, X_{t+1}, D_t)) \Pr(X_{t+1}^{\texttt{dest}}|X_t, D_t, A_t) \\
&= \Pr(X_{t+1}^{\neg\texttt{dest}}|X_t, A_t)\mathbb{1}(D_{t+1} = d(X_t, A_t, X_{t+1}, D_t)) \sum_{y_t^{\texttt{src}}} I(y_t^{\texttt{src}}|D_t) \Pr(X_{t+1}^{\texttt{dest}}|X_t, y_t^{\texttt{src}}, A_t)
\end{aligned}$$

where $\mathbb{1}(\cdot)$ is the indicator function and the notation I is introduced as the influence predictor, $I(y_t^{\texttt{src}}|D_t) = \Pr(y_t^{\texttt{src}}|D_t)$. The function d selects those variables that are relevant to predict the influence sources Y_t^{src}. In this paper, we set $d(X_t, A_t, X_{t+1}, D_t)) = D_t \cup \{A_{t-1}, X_t\}$. That is, even though in general it is possible to condition on the history of a subset of local states and actions, we just use the entire history of local states and actions for simplicity (see Suau et al. (2020) for an exploitation of this aspect of IBA in the context of Deep RL). The IALM is then formally defined as $\mathcal{M}_{\texttt{IALM}} = (\mathcal{S}^{\texttt{IALM}}, \mathcal{A}, \mathcal{T}^{\texttt{IALM}}, \mathcal{R}, \Omega, \mathcal{O}, b_0, \gamma)$ where the observation function $\mathcal{O}$, reward function $\mathcal{R}$ and the initial belief b_0 remain unchanged because of the definition of the local state variables X. Theorem 1 in Oliehoek et al. (2019) proves that this is a lossless abstraction by showing the optimal value of the IALM matches that of the global model, $V^*_{\mathcal{M}_{\texttt{IALM}}} = V^*_{\mathcal{M}_{\texttt{global}}}$.

3 Influence Augmented Online Planning

While IBA results in an IALM $\mathcal{M}_{\texttt{IALM}}$ that abstracts away non-local state variables Y in a lossless way, it is not useful in practice because computing the distribution $I(Y_t^{\texttt{src}}|D_t)$ exactly is in general intractable. Our approach trades off between the time spent before and during the online planning, by approximating $I(Y_t^{\texttt{src}}|D_t)$ with a function approximator $\hat{I}_\theta$ learned offline. The learned influence predictor $\hat{I}_\theta$ will then be integrated with an accurate local simulator $\mathcal{G}_{\texttt{local}}$ to construct an influence-augmented local simulator (IALS) that only simulates the local state variables X but concisely captures the influence of the non-local state variables Y by predicting the influence source state variables $Y^{\texttt{src}}$ with $\hat{I}_\theta$. During the online planning, the integrated IALS will be used to replace the accurate but slow global simulator to speed up the simulations for the sample-based online planners.

Our motivation is that by simulating the local transitions that directly decide the observation and reward of the agent with an accurate local simulator, the simulation inaccuracy caused by approximating the distribution $I(Y_t^{\texttt{src}}|D_t)$ with $\hat{I}_\theta$ can be overcome by the advantage that simulations can be performed significantly faster in the IALS, which is essential to sample-based planners like POMCP (Silver and Veness, 2010), leading to improved online planning performance in realistic scenarios with limited planning time. Our overall approach, influence-augmented online planning, is presented in Algorithm 1, followed by our method to learn an approximate influence predictor with recurrent neural networks (RNNs) (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) and integrate it with a local simulator to form a plannable IALS for sample-based planners.

3.1 Learning Approximate Influence Predictor Offline with RNNs

The dependency of $I(Y_t^{\texttt{src}}|D_t)$ on the d-separation set D_t renders it infeasible to be computed exactly online or offline. In this work we learn an approximate influence predictor offline with RNNs by formalizing it as a supervised sequential classification problem.

For planning with horizon $\mathcal{H}$, we need to predict the conditional distribution over the influence source state $I(Y_t^{\texttt{src}}|D_t)$ for $t = 1$ to $\mathcal{H}-1$. We do not need to predict $I(Y_0^{\texttt{src}}|D_0)$ as it is the initial belief over the influence source state. As RNNs require the input size to be constant for every time step, we drop the initial local state X_0 from D_t so that the input to RNNs at time step t is $\{A_{t-1}, X_t\}$ and the target is $Y_t^{\texttt{src}}$. If there exists a distribution from which we can sample a dataset $\mathcal{D}$ of input sequence $D_{\mathcal{H}-1}$ and target sequence $(Y_1^{\texttt{src}}, \dots, Y_{\mathcal{H}_1}^{\texttt{src}})$, then this is a classic sequential classification setup that can be learned by training a RNN $\hat{I}_\theta$ to minimize the average empirical KL divergence between $I(\cdot|D_t)$ and $\hat{I}_\theta(\cdot|D_t)$ with stochastic gradient descent (SGD) (Ruder, 2016), which yields a cross-entropy loss in practice. While we leave the question on how can we collect the dataset $\mathcal{D}$ in a way that maximizes the online planning performance for future investigation, in this paper we use a uniform random policy to sample $\mathcal{D}$ from the global simulator $\mathcal{G}_{\texttt{global}}$.

Algorithm 1: Influence-Augmented Online Planning

input : a real environment `env`
input : a global simulator $\mathcal{G}_{\texttt{global}}$ and a local simulator $\mathcal{G}_{\texttt{local}}$
input : an exploratory policy $\pi_{\texttt{explore}}$
input : a sample-based planner `planner` with a termination condition T, e.g., a fixed time limit
input : a planning horizon $\mathcal{H}$
Offline *Influence Learning*
  Collect a dataset $\mathcal{D}$ of input sequences $D_{\mathcal{H}-1}{=}(A_{i-1}, X_i)_{i=1}^{\mathcal{H}-1}$ and target sequences $(Y_i^{\texttt{src}})_{i=1}^{\mathcal{H}-1}$ by interacting with the global simulator $\mathcal{G}_{\texttt{global}}$ using the policy $\pi_{\texttt{explore}}$;
  Train an approximate influence predictor $\hat{I}_\theta$ on the dataset $\mathcal{D}$ by minimizing the average empirical KL Divergence between $I(\cdot|D_t)$ and $\hat{I}_\theta(\cdot|D_t)$;
Online *Planning with a sample-based planner*
  Integrate the local simulator $\mathcal{G}_{\texttt{local}}$ and the learned influence predictor $\hat{I}_\theta$ into an IALS $\mathcal{G}^\theta_{\texttt{IALM}}$;
  for $t = 0, \ldots, \mathcal{H}{-}1$ **do**
    plan for an action until T is met: $a_t = \texttt{planner.plan}(\mathcal{G}^\theta_{\texttt{IALM}}, T)$;
    execute the action in the real environment: $o_{t+1} = \texttt{env.act}(a_t)$;
    process the new observation: $\texttt{planner.observe}(o_{t+1})$
  end

3.2 Integrating the Local Simulator and RNN Influence Predictor for Online Planning

To plan online in a POMDP, sample-based planners like POMCP (Silver and Veness, 2010) require a generative simulator that supports sampling the initial states and transitions. As shown in Figure 1b, to sample a transition in the IALS $\mathcal{G}^\theta_{\texttt{IALM}}$, we need to first sample an influence source state $Y_t^{\texttt{src}}$ and then sample the local transitions in the local simulator $\mathcal{G}_{\texttt{local}}$. While in the original formulation of IBA, $\hat{I}_\theta(Y_t^{\texttt{src}}|D_t)$ conditions on the d-separation set D_t which grows with actions A_t and new local states X_{t+1} at every time step, we avoid feeding the entire D_t into RNNs for every prediction of $Y_t^{\texttt{src}}$ by taking the advantage of RNNs whose hidden state Z_t is a sufficient statistic of the previous inputs. As a result, we use $S_t^{\texttt{IALM}} = (X_t, Y_t^{\texttt{src}}, Z_t)$ as the state of the IALS in practice. The transition $s_{t+1}^{\texttt{IALM}}, o_{t+1}, r_{t+1} \sim \mathcal{G}^\theta_{\texttt{IALM}}(s_t^{\texttt{IALM}}, a_t)$ can then be sampled in two steps:

- sample the next local state, observation and reward: $x_{t+1}, o_{t+1}, r_t \sim \mathcal{G}_{\texttt{local}}(x_t, y_t^{\texttt{src}}, a_t)$
- sample the next RNN hidden state and influence source state : $z_{t+1}, y_{t+1}^{\texttt{src}} \sim \hat{I}_\theta(\cdot|z_t, a_t, x_{t+1})$

The initial state $S_0^{\texttt{IALM}}$ of the IALS can be easily sampled by first sampling a full state $s \sim b_0$ and then extracting the local state and the influence source state $(x_0, y_0^{\texttt{src}})$ from s.

4 Experiments

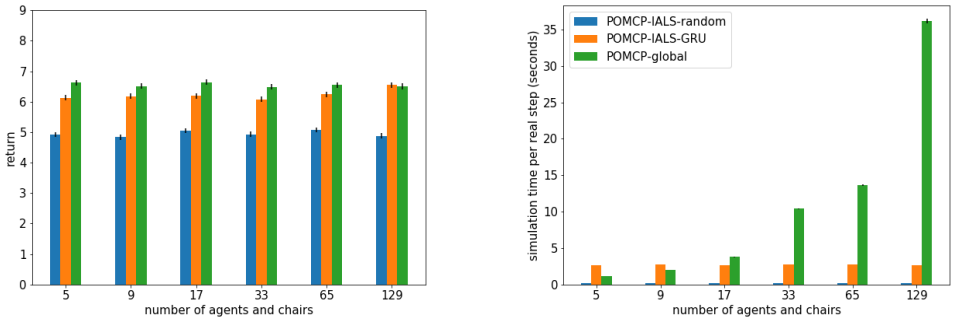
We perform online planning experiments with the POMCP planner (Silver and Veness, 2010) to answer the following questions: when learning approximate influence predictors with RNNs,

- can planning with an IALS be faster than planning with the global simulator while achieving similar performance, *when the same number of simulations are allowed per planning step*?
- can planning with an IALS yield better performance than planning with the global simulator, *when the same amount of planning time is allowed per planning step*?

Experimental Setup

Our codebase was implemented in C++, including a POMCP planner and several benchmarking domains [1]. We ran each of our experiments for many times on a computer cluster with the same amount of computational resources. To report results, we plot the means of evaluation metrics with standard errors as error bars. Details of our experiments are provided in the supplementary material.

[1] available at `https://github.com/INFLUENCEorg/IAOP`



(a) Average return

(b) Average simulation time per step (seconds)

Figure 2: Performance of POMCP with different simulators in Grab A Chair games of various sizes. While the IALS with GRU influence predictor achieves matching returns with the global simulator, the simulation is significantly faster in scenarios with many other agents.

Grab A Chair

The first domain we use is the Grab A Chair domain mentioned in Section 1. In our setting, the other agents employ a policy that selects chairs randomly in the beginning and greedily afterwards according to the frequency of observing to obtain a chair when visiting it.

Our intuition is that the amount of speedup we can achieve by replacing $\mathcal{G}_{\texttt{global}}$ with $\mathcal{G}^{\theta}_{\texttt{IALM}}$ depends on how fast we can sample influence source state variables $Y^{\texttt{src}}$ from the approximate influence predictor $\hat{I}_\theta$ and the size of hidden state variables $Y \backslash Y^{\texttt{src}}$ we can avoid simulating in $\mathcal{G}^{\theta}_{\texttt{IALM}}$. We perform planning with different simulators in games of $\{5, 9, 17, 33, 65, 129\}$ agents for a horizon of 10 steps, where a fixed number of 1000 Monte Carlo simulations are performed per step.

To obtain an approximate influence predictor $\hat{I}_\theta$, we sample a dataset $\mathcal{D}$ of 1000 episodes from the global simulator $\mathcal{G}_{\texttt{global}}$ with a uniform random policy and train a variant of RNN called Gated Recurrent Units (GRU) (Cho et al., 2014) on $\mathcal{D}$ until convergence. To test if capturing the incoming influence is essential for achieving good performance when planning on $\mathcal{G}^{\theta}_{\texttt{IALM}}$, we use an IALS with a uniform random influence predictor as an additional baseline, denoted as $\mathcal{G}^{\texttt{random}}_{\texttt{IALM}}$.

Figure 2a shows the performance of planning with different simulators in scenarios of various sizes. It is clear that planning on $\mathcal{G}^{\theta}_{\texttt{IALM}}$ achieves significantly better performance than planning on $\mathcal{G}^{\texttt{random}}_{\texttt{IALM}}$, emphasizing the importance of learning $\hat{I}_\theta$ to capture the influence. While planning on $\mathcal{G}^{\theta}_{\texttt{IALM}}$ can indeed achieve matching performance with $\mathcal{G}_{\texttt{global}}$ as shown by the small differences in their returns, the advantage of the IALS, its speed, is shown in Figure 2b. In contrast to $\mathcal{G}_{\texttt{global}}$ which slows down quickly because of the growing number of state variables to simulate, the computation time of both $\mathcal{G}^{\theta}_{\texttt{IALM}}$ and $\mathcal{G}^{\texttt{randomly}}_{\texttt{IALM}}$ barely increases. This is because those state variables added by more chairs and agents are abstracted away from the simulations in the IALS with their influence concisely captured by $\hat{I}_\theta$ in the distribution of the two neighboring agents' decisions. Note that $\mathcal{G}^{\theta}_{\texttt{IALM}}$ is slower than $\mathcal{G}_{\texttt{global}}$ in scenarios with few agents due to the overheads of feedforward passing in the GRU.

To further investigate how will influence-augmented online planning perform in environments with different *influence strengths*, by which we mean the degree to which the local states are affected by the influence source states, we repeat our experiments above in a variant of the 5-agent Grab A Chair game where the only difference is that when two agents target the same chair, both of them have the same probability $p \in [0, 1]$ to obtain the chair [2]. The intuition is that when p is lower, the influence from the rest of the environment will be stronger as the decisions of the two neighboring agents will be more decisive on whether the planning agent can secure a chair. In this case, higher prediction accuracy on the decisions of the two neighboring agents will be required for the agent to plan a good action. Figure 3 shows the planning performance with all simulators under decreasing p which

[2]Note that this leads to a physically unrealistic setting since it is possible that two agents obtain the same chair at a time step. However, it gives us a way to investigate the impact of the influence strength from the rest of the environment.

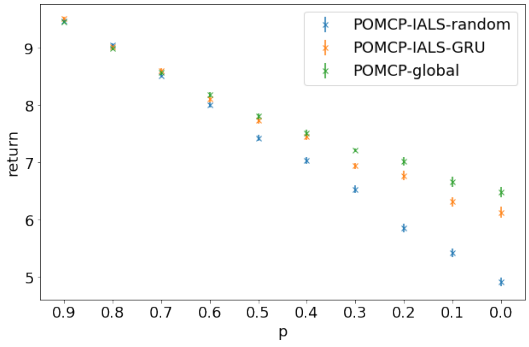


Figure 3: Performance of POMCP with different simulators in the modified Grab A Chair game under decreasing p, which implies stronger influence from the rest of the environment. The smaller performance difference between $\mathcal{G}^{\theta}_{\texttt{IALM}}$ and $\mathcal{G}_{\texttt{global}}$ under higher p suggests that learning an accurate influence predictor is more important to achieve good planning performance when the local planning problem is more tightly coupled with the rest of the environment.

implies stronger influence strength from the rest of the environment. While the same amount of effort was put into training the approximate influence predictor $\hat{I}_\theta$, the performance difference between planning with $\mathcal{G}^{\theta}_{\texttt{IALM}}$ and $\mathcal{G}_{\texttt{global}}$ is smaller under higher p. This suggests that in environments where the local planning problem is more tightly coupled with the rest of the environment, learning an accurate influence predictor $\hat{I}_\theta$ is more important to achieve good planning performance.

Real-Time Online Planning in Grid Traffic Control

The primary motivation of our approach is to improve online planning performance in realistic settings where the planning time per step is constrained. For this reason, we conduct real-time planning experiments in a more realistic domain called Grid Traffic Control, which simulates a busy traffic system with 9 intersections, each of which consists of 4 lanes with 6 grids as shown in Figure 4a, with more details provided in the supplementary material.

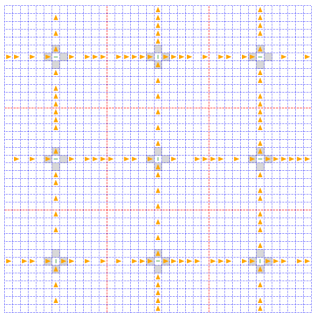
The traffic lights are equipped with sensors providing 4-bit information indicating if there are vehicles in the grids around them. While the other traffic lights employ a hand-coded switching strategy that prioritizes lanes with vehicles before the lights and without vehicles after the lights, the traffic light in the center is controlled by planning, with the goal to minimize the total number of vehicles in this intersection for a horizon of 30 steps.

As mentioned in Section 2.2, POMCP approximates the belief update with an unweighted particle filter that reuses the simulations performed during the tree search. However, in our preliminary experiments, we observed the particle depletion problem, which occurred when POMCP ran out of particles because none of the existing particles was evidenced by the new observation. While to alleviate this problem we use a workaround inspired by Silver and Veness (2010) [3], when particle depletion still occurs at some point during an episode, the agent employs a uniform random policy.

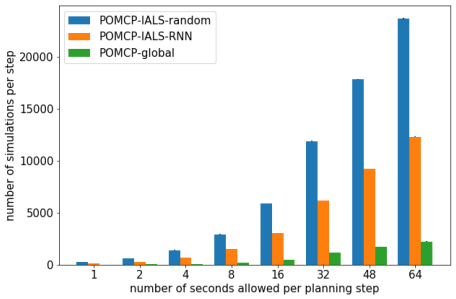
We train an influence predictor with a RNN and evaluate the performance of all three simulators $\mathcal{G}^{\texttt{random}}_{\texttt{IALM}}$, $\mathcal{G}^{\theta}_{\texttt{IALM}}$ and $\mathcal{G}_{\texttt{global}}$ in settings where the allowed planning time is fixed per step. Our hypothesis is that $\mathcal{G}^{\theta}_{\texttt{IALM}}$ will outperform the $\mathcal{G}_{\texttt{global}}$ when the planning time allowed is very constrained because in that case, the advantage on simulation speed will dominate the disadvantage on simulation accuracy caused by approximating the influence with $\hat{I}_\theta$.

Figure 4b demonstrates the ability of the IALS to perform more than twice the number of simulations that can be performed by the global simulator within the same fixed time. This is directly translated into the ability of POMCP to plan for more time steps before the particle depletion occurs as shown in Figure 4c. The more important effect of faster simulation is that our approach performs much better

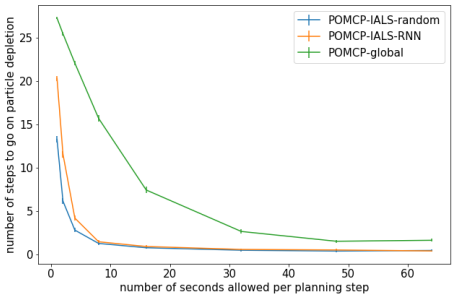
[3] While more advanced particle filters like Sequential Importance Resampling can reduce this problem, we chose to use POMCP in unmodified form to make it easier to interpret the benefits of our approach. Our workaround is that when the search tree is pruned because of a new observation, we add $N/6$ additional particles sampled from the initial belief b_0 to the current particle pool where N is the number of remaining particles.



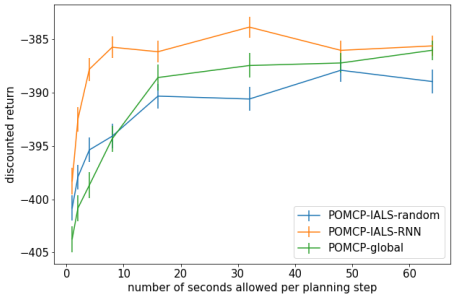
(a) The Grid Traffic Control domain



(b) Number of simulations performed per planning step



(c) Number of steps to go on particle depletion



(d) Discounted return

Figure 4: Performance of POMCP with different simulators while allowing different numbers of seconds per planning step in the Grid Traffic Control domain. While the planning performance of the IALS with trained influence predictor dominates the global simulator when the planning time is constrained, the performance difference decreases when more time is allowed.

than planning on the global simulator especially when the planning time is limited. This suggests that there does exist a trade-off between simulation speed and simulation accuracy that allows planning on the IALS with an approximate influence predictor to achieve better online performance.

Figure 6 in the supplementary material performs a similar time-constrained evaluation in the Grab A Chair domain. The finding there is that the advantage of the IALS on the simulation speed is clearer when the global model of the problem is more complex, in which cases the IALS with an approximate influence predictor shows a superior performance compared to the global simulator.

5 Related Work

The idea of utilizing offline knowledge learning for improved online planning performance has been well-studied (Gelly and Silver, 2007, 2011; Silver et al., 2016, 2017, 2018; Anthony et al., 2017). These approaches can be categorized as 1) learning value functions or policies to guide the tree search, 2) improving default policy for more informative rollouts, 3) replacing rollouts with learned value functions and 4) initializing state-action value estimates. Our approach takes a distinct path by speeding up computationally expensive forward simulations, which allows the planner to sample more trajectories for each decision.

Closest to our work is the approach by Chitnis and Lozano-Pérez (2020), which exploits *exogenous variables* to reduce the state space of the model for more efficient simulation and planning. While both of the approaches learn a more compact model by abstracting away state variables, exogenous variables are fundamentally different from the non-local variables that we abstract away. By definition, exogenous variables refer to those variables that are beyond the control of the agent: they cannot be affected, directly or indirectly, by the agent's actions (Boutilier et al., 1999; Chitnis and Lozano-Pérez, 2020). In contrast, the non-local variables that are abstracted away in IBA (Oliehoek et al., 2012) can be chosen more freely, as long as they do not *directly* affect the agent's observation and reward. Therefore, the exogenous variables and non-local variables are in general two different sets

of variables that can be exploited to reduce the state space size. For instance, in the traffic problem of Figure 4a, there are no exogenous variables as our action can directly or indirectly effect the transitions at other intersections (by taking or sending vehicles from/to them). This demonstrates that our approach allows us to reduce the state space of this problem beyond the exogenous variables.

The idea of replacing a computationally demanding simulator with an approximate simulator for higher simulation efficiency has been explored in many fields under the name of *surrogate model*, such as computer animation (Grzeszczuk et al., 1999), network simulation (Kazer et al., 2018), the simulation of seismic waves (Moseley et al., 2018) and so on. Our work explores this idea in the context of sample-based planning in structured domains.

Recent works in deep model-based reinforcement learning (Oh et al., 2017; Farquhar et al., 2018; Hafner et al., 2019; Schrittwieser et al., 2019; Van der Pol et al., 2020) have proposed to learn an approximate model of the environment by interacting with it, and then plan a policy within the learned model for better sample efficiency. Our method considers a very different setting, in which we speed up the simulation for sample-based planning by approximating part of the global simulator, that is, the influence from the rest of the environment, and retain the simulation accuracy by explicitly utilizing a light and accurate local simulator.

6 Conclusion

In this work we aim to address the problem that simulators modeling the entire environment is often slow and hence not suitable for sample-based planning methods which require a vast number of Monte Carlo simulations to plan a good action. Our approach transforms an expensive factored global simulator into an influence-augmented local simulator (IALS) that is less accurate but much faster. The IALS utilizes a local simulator which accurately models the state variables that are most important to the planning agent and captures the influence from the rest of the environment with an approximate influence predictor learned offline. Our empirical results in the Grid Traffic Control domain show that in despite of the simulation inaccuracy caused by approximating the incoming influence with a recurrent neural network, planning on the IALS yields better online performance than planning on the global simulator due to the higher simulation efficiency, especially when the planning time per step is limited. While in this work we collect data from the global simulator with a random exploratory policy to learn the influence, a direction for future work is to study how this offline learning procedure can be improved for better performance during online planning.

Broader Impact

The potential impact of this work is precisely its motivation: making online planning more useful in real-world decision making scenarios, enabling more daily decisions to be made autonomously and intelligently, with promising applications including autonomous warehouse and traffic light control.

Unlike simulators constructed by domain experts, which are in general easier to test and debug, influence-augmented local simulator contains an approximate influence predictor learned from data, which may fail with rare inputs and result in catastrophic consequences especially when controlling critical systems. This suggests that extensive testing and regulation will be required before deploying influence-augmented local simulators in real-world decision making scenarios.

Acknowledgments and Disclosure of Funding

This project had received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 758824 —INFLUENCE).

References

Anthony, T., Tian, Z., and Barber, D. (2017). Thinking Fast and Slow with Deep Learning and Tree Search. In *Advances in Neural Information Processing Systems*, pages 5360–5370.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256.

Becker, R., Zilberstein, S., and Lesser, V. (2004). Decentralized Markov decision processes with event-driven interactions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2004*, volume 1, pages 302–309.

Becker, R., Zilberstein, S., Lesser, V., and Goldman, C. V. (2003). Transition-Independent Decentralized Markov Decision Processes. In *Proceedings of the International Conference on Autonomous Agents*, volume 2, pages 41–48.

Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11:1–94.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of Monte Carlo tree search methods.

Chitnis, R. and Lozano-Pérez, T. (2020). Learning compact models for planning with exogenous processes. In *Conference on Robot Learning*, pages 813–822. PMLR.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., 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)*, pages 1724–1734.

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, volume 4630 LNCS, pages 72–83.

Farquhar, G., Rocktäschel, T., Igl, M., and Whiteson, S. (2018). TreeqN and ATreEC: Differentiable tree-structured models for deep reinforcement learning. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*.

Gelly, S. and Silver, D. (2007). Combining online and offline knowledge in UCT. In *Proceedings of the 24th international conference on Machine learning*, volume 227, pages 273–280.

Gelly, S. and Silver, D. (2011). Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175:1856–1875.

Grzeszczuk, R., Terzopoulos, D., and Hinton, G. (1999). Fast neural network emulation of dynamical systems for computer animation. In *Advances in Neural Information Processing Systems*, pages 882–888.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019). Learning Latent Dynamics for Planning from Pixels. In *International Conference on Machine Learning*, pages 2555—-2565.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134.

Kazer, C. W., Sedoc, J., Ng, K. K., Liu, V., and Ungar, L. H. (2018). Fast network simulation through approximation or: How blind men can describe elephants. In *HotNets 2018 - Proceedings of the 2018 ACM Workshop on Hot Topics in Networks*, pages 141–147.

Kearns, M., Mansour, Y., and Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine learning*, 49(2-3):193—-208.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *European conference on machine learning*, volume 4212 LNAI, pages 282–293.

Li, L., Walsh, T. J., and Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. In *9th International Symposium on Artificial Intelligence and Mathematics, ISAIM 2006*.

Moseley, B., Markham, A., and Nissen-Meyer, T. (2018). Fast approximate simulation of seismic waves with deep learning. *arXiv preprint arXiv:1807.06873*.

Oh, J., Singh, S., and Lee, H. (2017). Value prediction network. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 6119–6129.

Oliehoek, F. A., Witwicki, S., and Kaelbling, L. P. (2019). A Sufficient Statistic for Influence in Structured Multiagent Environments. *arXiv preprint arXiv:1907.09278*.

Oliehoek, F. A., Witwicki, S. J., and Kaelbling, L. P. (2012). Influence-based abstraction for multiagent systems. In *Twenty-sixth AAAI conference on artificial intelligence*.

Petrik, M. and Zilberstein, S. (2009). A Bilinear programming approach for multiagent planning. *Journal of Artificial Intelligence Research*, 35:235–274.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. (2019). Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *arXiv preprint arXiv:1911.08265*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359.

Silver, D. and Veness, J. (2010). Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems*, pages 2164—-2172.

Suau, M., Congeduti, E., Starre, R., Czechowski, A., and Olihoek, F. (2020). Influence-aware memory for deep reinforcement learning. *arXiv preprint arXiv:1911.07643*.

Van der Pol, E., Kipf, T., Oliehoek, F. A., and Welling, M. (2020). Plannable Approximations to MDP Homomorphisms: Equivariance under Actions. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1431—-1439.

Witwicki, S. J. and Durfee, E. H. (2010). Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In *ICAPS 2010 - Proceedings of the 20th International Conference on Automated Planning and Scheduling*, pages 185–192.

7 Supplementary Material

In this supplementary material, we provide the details of our experimental setups for reproducibility.

7.1 Grab A Chair

Environment

The Grab A Chair game is an N-agent game where at every time step, each agent has an action space of two, trying to grab the chair on its left or right side. An agent only secures a chair if its targeted chair is not targeted by a neighboring agent. At the end of a time step t, each agent with $s_{t+1} \in \{0, 1\}$ indicating if this agent obtains a chair receives a reward $r_t = s_{t+1}$ and a noisy observation o_{t+1} on s_{t+1} which has a probability 0.2 to be flipped.

In the experiments of Figure 3, when two agents target the same chair, both of them have a probability of $p \in [0, 1]$ to secure the chair, which means that there is a probability that two neighboring agents obtain the same chair. The following setup applies to all the experiments in this domain.

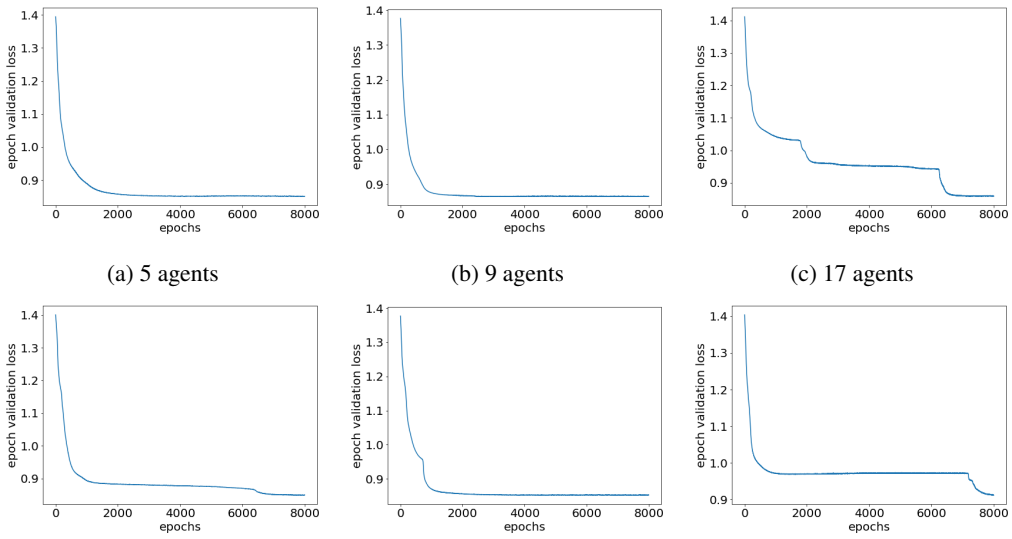
Experimental Setup

Influence Learning

In this domain, the approximate influence predictor $\hat{I}_\theta$ is parameterized by a GRU (Cho et al., 2014) classifier with 8 hidden units. The dataset $\mathcal{D}$ consists of 1000 episodes collected from the global simulator $\mathcal{G}_{\texttt{global}}$ with a uniform random policy, where 800 episodes are used as the training set and the other 200 episodes are used as the validation set. The hyperparameters used to train the GRU influence predictors in scenarios with $\{5, 9, 17, 33, 65, 129\}$ agents are shown in Table 1 and their learning curves are shown in Figure 5.

Table 1: Hyperparameters used to train the GRU influence predictors for experiments in the Grab A Chair domain, where the weight decay was fine tuned within the range until there is no clear sign of overfitting.

Learning rate	0.0005
Batch size	128
Number of epochs	8000
Weight decay	$[1 \times 10^{-5}, 5 \times 10^{-5}]$



(a) 5 agents (b) 9 agents (c) 17 agents

(d) 33 agents (e) 65 agents (f) 129 agents

Figure 5: Learning curves of influence predictors in the Grab A Chair domain.

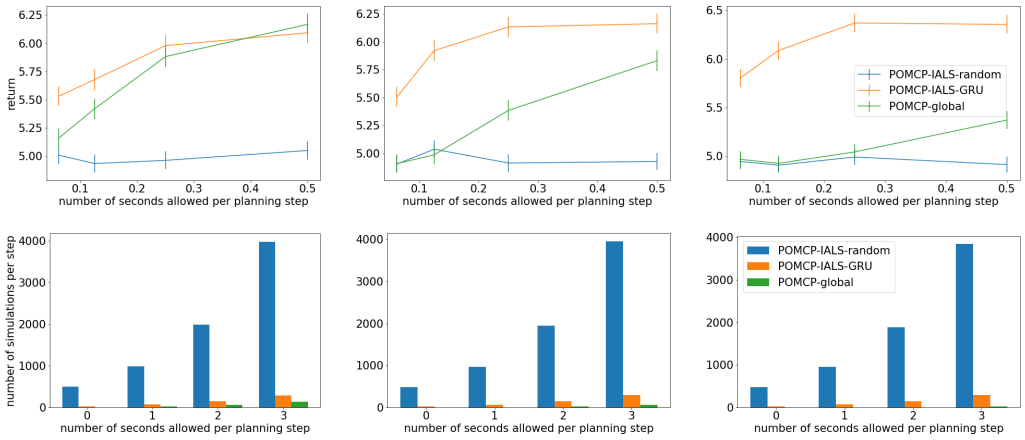


Figure 6: Performance of POMCP with different simulators while allowing different numbers of seconds per planning step in Grab A Chair games with 33 (left), 65 (middle) and 129 (right) agents. The advantage of IALS with GRU influence predictor over global simulator becomes clearer as the global model of the planning problem gets more complex.

Planning with POMCP

The parameters used in the planning experiments with POMCP are shown in Table 2.

Table 2: Parameters for the planning experiments with POMCP in the Grab A Chair domain.

Discount factor	1.0
Horizon	10
Number of simulations per step	1000
Number of initial particles	1000
Exploration constant in the UCB1 algorithm (c)	100.0

Real-time Online Planning in Grab A Chair domain

We conduct a time-constrained evaluation in this domain with $\{33, 65, 129\}$ agents, similar to the one performed in the Grid Traffic Control domain, where different amount of time is allowed to plan an action. Results in Figure 6 show that the advantage of the IALS with GRU influence predictor is clearer when the global model of the planning gets more complex.

7.2 Grid Traffic Control

Environment

The Grid Traffic Control environment simulates a traffic system of 9 intersections as shown in Figure 4a. The vehicles, plotted as yellow arrows, move from the left to right and the bottom to top, governed by the traffic lights in the center of each intersection. While they are initially generated with a probability of 0.7 in each grid, new vehicles will enter the traffic system at entrances on the left and bottom borders whenever they are not occupied at last time step. When reaching the right and bottom borders, with a probability of 0.3, vehicles leave the traffic system.

While the other traffic lights are controlled by fixed switching strategies, the traffic light in the center intersection is controlled by the planning agent, whose action space consists of actions to set the light green for each lane. After an action a_t is taken which results in the movement of vehicles, the agent receives an observation consisting of four Boolean variables o_{t+1}=$\{$`left_occupied`, `right_occupied`, `up_occupied`, `bottom_occupied`$\}$ indicating if the four grids around the traffic light are occupied. The reward r_t is the negative number of grids that are occupied in this intersection after the transition at time step t.

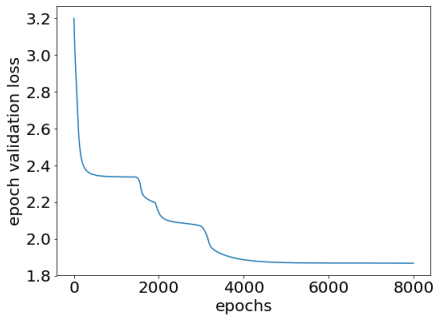


Figure 7: The learning curve of the influence predictor in the Grid Traffic Control domain.

Experimental Setup

Influence Learning

In this domain, the approximate influence predictor $\hat{I}_\theta$ is parameterized by a RNN classifier with 2 hidden units. The dataset $\mathcal{D}$ consists of 1000 episodes collected from the global simulator $\mathcal{G}_{\texttt{global}}$ with a uniform random policy, where 800 episodes are used as the training set and the other 200 episodes are used as the validation set. The hyperparameters used to train the RNN influence predictor are shown in Table 3 and its learning curve is shown in Figure 7.

Table 3: Hyperparameters used to train the RNN influence predictor for experiments in the Grid Traffic Control domain.

Learning rate	0.00025
Batch size	128
Number of epochs	8000
Weight decay	1×10^{-4}

Planning with POMCP

The parameters used in the planning experiments with POMCP are shown in Table 4, where effective horizon is the maximal depth from the root node that a search or a rollout will be performed.

Table 4: Parameters for the planning experiments with POMCP in the Grid Traffic Control domain.

Discount factor	0.95
Horizon	30
Number of seconds allowed per planning step	$\{1, 2, 4, 8, 16, 32, 48, 64\}$
Number of initial particles	1000
Exploration constant in the UCB1 algorithm (c)	10.0
Effective horizon	18