Online Planning in POMDPs with Self-Improving Simulators

Jinke He[1] , **Miguel Suau**[1] , **Hendrik Baier**[2] , **Michael Kaisers**[2] and **Frans A. Oliehoek**[1]

[1]Delft University of Technology, The Netherlands
[2]Centrum Wiskunde & Informatica, The Netherlands
{J.He-4, M.SuaudeCastro, F.A.Oliehoek}@tudelft.nl, {hendrik.baier, michael.kaisers}@cwi.nl

Abstract

How can we plan efficiently in a large and complex environment when the time budget is limited? Given the original simulator of the environment, which may be computationally very demanding, we propose to learn online an approximate but much faster simulator that improves over time. To plan reliably and efficiently while the approximate simulator is learning, we develop a method that adaptively decides which simulator to use for every simulation, based on a statistic that measures the accuracy of the approximate simulator. This allows us to use the approximate simulator to replace the original simulator for faster simulations when it is accurate enough under the current context, thus trading off simulation speed and accuracy. Experimental results in two large domains show that when integrated with POMCP, our approach allows to plan with improving efficiency over time.

1 Introduction

Decision making under uncertainty is one of the key problems in artificial intelligence [Kaelbling *et al.*, 1998; Spaan, 2012]. Online planning methods such as POMCP [Silver and Veness, 2010] and DESPOT [Somani *et al.*, 2013] have become popular as they enable highly efficient decision making by focusing the computation on the current context. However, these methods rely heavily on fast simulators that can rapidly perform a large number of Monte Carlo simulations. Unfortunately, many real-world domains are complex and thus slow to simulate, which impedes real-time online planning.

To mitigate the cost of simulation, researchers have explored learning surrogate models that are computationally less expensive [Grzeszczuk *et al.*, 1998; Buesing *et al.*, 2018; Chitnis and Lozano-Pérez, 2020]. However, one disadvantage of these methods is that they learn a model of the *entire* environment, which may be unnecessary and is often a difficult task. A recent approach by He *et al.* [2020] builds on the framework of influence-based abstraction to overcome this problem: They propose to exploit the structure of the environment via a so-called influence-augmented local simulator (IALS), which uses a lightweight simulator to capture the local dynamics that surround the agent and learns an influence predictor $\hat{I}_\theta$ that accounts for the interaction between the local dynamics and the rest of the environment.

However, there are three main limitations of this "two-phase" paradigm, in which a simulator is learned offline and then used as-is for online simulation and planning. First, no planning is possible until the offline learning phase finishes, which can take a long time. Second, the separation of learning and planning raises the question of what data collection policy should be used during training to ensure accurate online prediction during planning. We empirically demonstrate that when the training data is collected by a uniform random policy, the learned influence predictors can perform poorly during online planning, due to distribution shift. Third, completely replacing the original simulator with the approximate one after training implies a risk of poor planning performance in certain situations, which is hard to detect in advance.

In this work, we aim to overcome these drawbacks by investigating whether we can learn the influence predictor $\hat{I}_\theta$ used in an IALS *online, without any pretraining*. As the quality of such an IALS with an untrained influence predictor would be poor initially, we investigate if during online planning, we can selectively use the learned IALS and the accurate but slow global simulator (GS) to balance simulation speed and accuracy. To address this major challenge, we propose a simulator selection mechanism to choose between the GS and IALS based on the latter's accuracy under the current context, which we estimate online with the simulations of the GS. This enables us to use the fast IALS when it is sufficiently accurate, while reverting back to the GS otherwise.

Our experiments reveal that as the influence predictor becomes more accurate, the simulator selection mechanism starts to use the IALS more and more often, which allows for high-accuracy simulations with increasing speed. Planning with such *self-improving simulators* speeds up decision making (given a fixed number of simulations) and increases task performance (given a fixed time budget), without pretraining the influence predictor $\hat{I}_\theta$. Moreover, we find that influence predictors that are trained with online data, i.e., coming from online simulations of the GS, can significantly outperform those trained with offline data, both in terms of online prediction accuracy and planning performance.

Extended version of this paper with supplementary material is available at https://arxiv.org/abs/2201.11404.

2 Background

POMDPs A Partially Observable Markov Decision Process (POMDP) [Kaelbling *et al.*, 1998] is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \Omega, \mathcal{T}, \mathcal{R}, \mathcal{O}, \gamma)$ where $\mathcal{S}$, $\mathcal{A}$ and Ω represent the state, action and observation space. At every time step t, the decision making agent takes an action a_t, which causes a transition of the environment state from s_t to $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$, and receives a reward $r_t = \mathcal{R}(s_t, a_t)$ and a new observation $o_{t+1} \sim \mathcal{O}(\cdot|s_{t+1}, a_t)$. A policy π of the agent maps an action-observation history $h_t = (a_0, o_1, \ldots, a_{t-1}, o_t)$ to a distribution over actions. The value function $V^\pi(h_t)$ denotes the expected discounted return $\mathbb{E}_\pi[\sum_{k=0}^\infty \gamma^k r_{t+k}|h_t]$ when a history h_t is observed and a policy π is followed afterwards. The value for taking action a_t at h_t and following π thereafter is captured by the action-value function $Q^\pi(h_t, a_t)$. The optimal action-value function denotes the highest value achievable by any policy $Q^*(h_t, a_t) = \max_\pi Q^\pi(h_t, a_t)$.

POMCP Classical POMDP techniques [Kaelbling *et al.*, 1998; Spaan, 2012] scale moderately as they explicitly represent the exact beliefs over states. To improve scalability, POMCP [Silver and Veness, 2010] performs online planning based on Monte Carlo Tree Search [Browne *et al.*, 2012] to focus the computation on the current context h_t, while applying particle filtering to avoid representing the full exact belief.

To make a decision after observing history h_t, POMCP repeatedly performs Monte Carlo simulations to incrementally build a lookahead search tree with a generative simulator $\mathcal{G}$. Each node in the tree represents a simulated history with visit count and average return maintained for every action. POMCP starts simulation i by sampling a particle $s_t^{\mathcal{G}}$ (a possible current environment state) from the root node, which is then used to simulate a trajectory τ_i. The simulated actions are chosen by the UCB1 algorithm [Auer *et al.*, 2002] to balance exploration and exploitation when the simulated histories are represented in the tree, and otherwise by a random rollout policy. After simulation, the statistics of the tree nodes are updated with the simulated trajectory and exactly one node is added to the tree to represent the first newly encountered history. The whole planning process can be seen as approximating the local optimal action-value function $\hat{Q}^*(h_t, \cdot)$. As the result of planning, the action that maximizes the average return $a_t = \arg\max_a \hat{Q}^*(h_t, a)$ is returned.

Influence-Augmented Local Simulators While POMCP has led to great improvement in planning efficiency, it does not address the problem of representing a very large POMDP. For concise representations, Factored POMDPs [Hansen and Feng, 2000] formulate a state s_t as an assignment of state variables and exploit conditional independence between them to compactly represent the transition and reward models with a so-called Two-stage Dynamic Bayesian Network (2DBN) [Boutilier *et al.*, 1999], as illustrated in Figure 1. Such compact representations enable us to represent very large domains with thousands of state variables, but sampling all of them at every simulation is still too costly. By using influence-based abstraction (IBA) [Oliehoek *et al.*, 2021] to abstract away state variables that do not directly affect the agent's observation and reward, it is possible to simulate and plan more
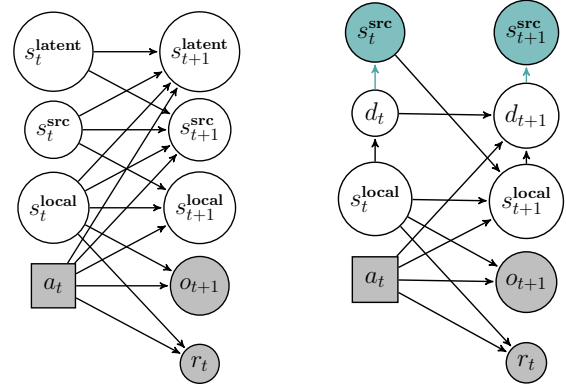

Figure 1: Two-stage Dynamic Bayesian Network (2DBN); observable variables (grayed), and control input (square). Full model (left), and the model of IALS (right) with local history dependent influence source state distribution (colored).

efficiently in large POMDPs [He *et al.*, 2020].

Consider the left 2DBN in Figure 1, a model that can be readily used for simulation, which, however, might be too slow if there are many state variables. Fortunately, in many problems, the agent only directly interacts with a small part of the environment. Formally, we define the *local state* $s_t^{\text{local}} \subseteq s_t$ as the subset of state variables that do directly affect the observation and reward of the agent. Then, the *influence source state* $s_t^{\text{src}} \subseteq s_t \backslash s_t^{\text{local}}$ is defined as the state variables that directly affect the local state. By combining the observation and reward models with the *local transition function* that models the transition of the local state $s_{t+1}^{\text{local}} \sim \mathcal{T}^{\text{local}}(\cdot|s_t^{\text{local}}, s_t^{\text{src}}, a_t)$, an abstract *local model* can be constructed, which concisely models the agent's observation and reward with the only external dependency being the influence source state s_t^{src}. Figure 1 (Right) illustrates the influence-augmented local simulator (IALS), which consists of a simulator of the local model $\mathcal{G}_{\text{local}}$ that can be extracted from the 2DBN, and an influence predictor I. The influence predictor models the influence source distribution $I(s_t^{\text{src}}|d_t)$ conditioned on the local states and actions $d_t = (s_0^{\text{local}}, a_0, s_1^{\text{local}}, \ldots, a_{t-1}, s_t^{\text{local}})$. Note that the latent state variables $s_t \backslash (s_t^{\text{local}} \bigcup s_t^{\text{src}})$ are abstracted away in this local simulator, compared to the global simulator that simulates all the state variables in the 2DBN, thereby boosting simulation efficiency (see Table 1 for comparison).

Although the IALS is proven to be unbiased with an exact influence predictor I, i.e., optimal planning performance can still be achieved when replacing the global simulator with the IALS [Oliehoek *et al.*, 2021], the required inference to obtain I is generally intractable. To address this, He *et al.* [2020] propose a two-phase approach that first learns an approximate influence predictor with recurrent neural networks (RNNs) [Cho *et al.*, 2014] offline and then uses it as-is as part of the IALS, to replace the global simulator for faster simulations. Experimental results show that this leads to an increase in simulation efficiency, which improves the online planning performance under limited planning budget. Algorithm 2 (appendix) summarizes the procedure of applying this approach.

Simulator	Feature	State	Simulation
the global simulator $\mathcal{G}_{\texttt{global}}$	accurate & slow	s_t	$s_{t+1}, o_{t+1}, r_t \sim \mathcal{G}_{\texttt{global}}(s_t, a_t)$
the IALS $\mathcal{G}_{\texttt{IALS}}^{\theta} = (\mathcal{G}_{\texttt{local}}, \hat{I}_{\theta})$	approximate & fast	$s_t^{\texttt{IALS}} = (s_t^{\texttt{local}}, d_t)$	$s_t^{\texttt{src}} \sim \hat{I}_{\theta}(\cdot \| d_t)$ $s_{t+1}^{\texttt{local}}, o_{t+1}, r_t \sim \mathcal{G}_{\texttt{local}}(s_t^{\texttt{local}}, s_t^{\texttt{src}}, a_t)$

Table 1: Comparison between the global simulator (GS) and the influence-augmented local simulator (IALS).

3 Planning with Self-improving Simulators

To address the limitations of the two-phase approach mentioned before (the need for pre-training, the distribution mismatch between states visited during training and planning, and the risk of catastrophic planning performance implied by replacing the original simulator completely with the approximate one after training), we introduce a new approach that 1) starts planning from the start while collecting the data to train the surrogate model, 2) trains the influence predictor with this data, which is now more pertinent to planning, and 3) reverts back to the global simulator $\mathcal{G}_{\texttt{global}}$ when the IALS is considered inaccurate for the current context (i.e., the current history and future trajectories we are likely to reach). Additionally, by switching back to $\mathcal{G}_{\texttt{global}}$ when the IALS is inaccurate, we provide more training to improve it precisely for those inaccurate contexts. As such the IALS will improve over time, and be used more frequently, such that overall simulation efficiency improves over time. Therefore, we refer to our approach as a *self-improving simulator (SIS)*.

We propose and test the integration of these ideas in POMCP, although the principle is transferable to other planning methods such as DESPOT that also iteratively perform simulations. Algorithm 1 outlines our approach. The planning starts with an arbitrarily initialized influence predictor $\hat{I}_{\theta}$, which is combined with the local simulator $\mathcal{G}_{\texttt{local}}$ that captures the local dynamics $(\mathcal{T}^{local}, \mathcal{O}, \mathcal{R})$ to form an IALS $\mathcal{G}_{\texttt{IALS}}^{\theta} = (\mathcal{G}_{\texttt{local}}, \hat{I}_{\theta})$. For every simulation, a simulator is selected between $\mathcal{G}_{\texttt{global}}$ and $\mathcal{G}_{\texttt{IALS}}^{\theta}$ to produce a new trajectory τ_i (line 5). This trajectory is processed as in regular POMCP to update statistics and expand one new tree node (line 6), regardless which simulator was used. Note that this means that the constructed tree carries information from both simulators. In line 7, however, only when τ_i comes from $\mathcal{G}_{\texttt{global}}$, we extract and store training data from τ_i. Action selection (line 9) is the same as in regular POMCP.

The data stored, denoted as $\mathcal{D} = \{(d_k, s_k^{\texttt{src}})\}$ for any $0 \le k \le \mathcal{H} - 1$, can then be used as a replay buffer to further improve $\hat{I}_{\theta}$ regularly. For example, Algorithm 1 suggests doing so at the end of each real episode. To train the approximate influence predictor, parameterized as a recurrent neural network, we follow [He *et al.*, 2020] and treat it as a sequential classification task where the cross entropy loss is minimized by stochastic gradient descent [Ruder, 2016]:

$$\mathcal{L}(\theta; \mathcal{D}) = - \mathop{\mathbb{E}}_{d_k, s_k^{\texttt{src}} \sim \mathcal{D}} \log \hat{I}_{\theta}(s_k^{\texttt{src}} \| d_k).$$

Overall, we expect to mainly select the global simulator for earlier simulations, since $\hat{I}_{\theta}$ is not yet very accurate. Over time, we expect the IALS to become more accurate, and thus more frequently used. However, global simulations are needed to assess the accuracy of the IALS, thus leading to a complex exploration/exploitation problem.

Algorithm 1 Planning with the Self-improving Simulator

1: initialize the influence predictor $\hat{I}_{\theta}$
2: **for** every real episode **do**
3:     **for** every time step t **do**
4:         **for** every simulation $i = 0, \dots$ **do**
5:             select either $\mathcal{G}_{\texttt{global}}$ or $\mathcal{G}_{\texttt{IALS}}^{\theta}$ and simulate trajectory τ_i
6:             update the search tree with τ_i
7:             if $\mathcal{G}_{\texttt{global}}$ generated τ_i, add training data to $\mathcal{D}$
8:         **end for**
9:         take greedy action, and prune tree with new observation
10:     **end for**
11:     train the influence predictor $\hat{I}_{\theta}$ on $\mathcal{D}$ for N steps
12: **end for**

---

4 Online Simulator Selection

The major challenge in our approach is how to select between the simulators online for fast *and* accurate simulations. Since the decision quality of IALS pivots on the influence source distribution aligning with what is knowable about the actual random influences in the global simulator, our solution begins with an information-theoretic accuracy measure of $\hat{I}_{\theta}$, which we then show can be estimated online. Subsequently, we describe how to address the exploration/exploitation problem when selecting simulators online with the UCB1 algorithm.

4.1 IALS Accuracy Measure

To address the question of measuring the accuracy of the IALS, one idea is to measure the Kullback–Leibler (KL) divergence between the true influence source distribution and the one predicted by the influence predictor. This is motivated by theory [Congeduti *et al.*, 2021], which shows that the maximum KL divergence $D_{KL}(I(S_k^{\texttt{src}} \| d_k) \| \hat{I}_{\theta}(S_k^{\texttt{src}} \| d_k))$ of any history of local states and actions d_k can be used to bound the value loss. However, our accuracy measure should not only consider the accuracy of $\hat{I}_{\theta}$ at the current simulated step but should also take the upcoming simulated steps into account.

Formally, at every real time step t given the action-observation history h_t, we denote the (partial) search tree at the beginning of POMCP simulation i as $\texttt{tree}_i$ and the tree policy as $\pi^{\texttt{tree}_i}$. For every simulated time step $k \ge t$, we define $P_k(D_k \| h_t, \pi^{\texttt{tree}_i})$ as the distribution over local histories D_k at that time step [1]. That is, for a local history $d_k = (s_0^{\texttt{local}}, a_0, s_1^{\texttt{local}}, \dots, a_{k-1}, s_k^{\texttt{local}})$, $P_k(d_k \| h_t, \pi^{\texttt{tree}_i})$ defines the probability of the local history at time step k being d_k, if the policy $\pi^{\texttt{tree}_i}$ is executed from time step t, knowing that the history at time step t is h_t. Then, the expected KL divergence between the true influence source distribution I and

---

[1]We use the uppercase and lowercase letters to distinguish between random variables and their realizations.

the approximate one given by $\hat{I}_\theta$ is defined as:

$$\mathcal{L}_i^k \triangleq \mathop{\mathbb{E}}_{d_k \sim P_k(\cdot|h_t, \pi^{\mathrm{tree}_i})} D_{\mathrm{KL}}\left(I(S_t^{\mathtt{src}}|d_k) || \hat{I}_\theta(S_t^{\mathtt{src}}|d_k) \right)$$

Averaging the expected KL divergence over all simulated time steps, we have $\mathcal{L}_i \triangleq \frac{1}{\mathcal{H}-t} \sum_{k=t}^{\mathcal{H}-1} \mathcal{L}_i^k$. The intuition here is that if we are now at real time step t with history h_t, and have performed $i-1$ POMCP simulations, then $\mathcal{L}_i$ measures the expected average KL divergence between the true influence source distribution and the one approximated by $\hat{I}_\theta$, with the local history d_k under the distribution induced by drawing a new POMCP simulation from an IALS with an exactly accurate influence predictor I.

4.2 Estimating the Accuracy

While we do not have access to the true influence source distribution, i.e., $I(\cdot|d_k)$, we can use the simulations from the global simulator to construct an estimate of the accuracy.

For every simulated step $t \le k \le \mathcal{H}-1$ during simulation i,

$$\mathcal{L}_i^k = \mathop{\mathbb{E}}_{d_k \sim P_k} [\underbrace{H(I(S_t^{\mathtt{src}}|d_k), \hat{I}_\theta(S_t^{\mathtt{src}}|d_k))}_{\text{cross entropy}} - \underbrace{H(I(S_t^{\mathtt{src}}|d_k))}_{\text{entropy}}].$$

For the cross entropy part, it is possible to show the expectation of it equals $\mathbb{E}_{d_k, s_k \sim P_k(\cdot|h_t, \pi^{\mathrm{tree}_i})} - \log \hat{I}_\theta(s_k^{src}|d_k)$, with full derivation in Appendix A.1. This means that this term can be estimated from samples from the global simulator, as long as we keep track of the local history d_k during simulation.

Unlike cross entropy, we cannot directly estimate entropy from samples due to the lack of an unbiased estimator for entropy [Paninski, 2003]. Here we choose to use the entropy of the same distribution but instead conditioned on the global state, as a lower bound of the entropy to estimate (see full derivation in Appendix A.2):

$$\mathop{\mathbb{E}}_{d_k} [H(I(S_k^{\mathtt{src}}|d_k))] \ge \mathop{\mathbb{E}}_{s_{k-1}, a_{k-1}} [H(S_k^{src}|s_{k-1}, a_{k-1})],$$

where both expectations are with respect to $P_k(\cdot|h_t, \pi^{\mathtt{tree}})$. This inequality holds intuitively because the global state s_k, as a Markovian signal, contains more information than the local history d_k, on the random variable $S_k^{\mathtt{src}}$. The entropy $H(S_k^{src}|s_{k-1}, a_{k-1})$ can be directly computed with the 2DBN, which we assume is given. Importantly, the use of a lower bound on the entropy means that we estimate an upper bound on the KL divergence, making sure we do not theoretically underestimate the inaccuracy of the IALS simulations.

Overall this constructs an upper bound on the expected average KL divergence between the true influence source distribution and the one approximated by $\hat{I}_\theta$, for simulation i:

$$\mathcal{L}_i \le \frac{1}{\mathcal{H}-t} \sum_{k=t}^{\mathcal{H}-1} \mathop{\mathbb{E}}_{d_k, s_k \sim P(\cdot|h_t, \pi^{\mathtt{tree}})} - \log \hat{I}_\theta(s_k^{src}|d_k)$$
$$- \mathop{\mathbb{E}}_{s_{k-1}, a_{k-1} \sim P(\cdot|h_t, \pi^{\mathtt{tree}})} H(S_k^{src}|s_{k-1}, a_{k-1})$$

As mentioned, the quantity above can be estimated with samples from the global simulator by augmenting the state s_t with the history of local states and actions d_t. Specifically,

to perform a POMCP simulation at time step t, we start by sampling a possible pair (s_t, d_t) from the root node. For every simulated step k, we first perform the simulation with the global simulator $s_{k+1}, r_k, o_{k+1} \sim \mathcal{G}_{\mathtt{global}}(s_k, a_k)$ and then let $d_{k+1} = d_k a_k s_{k+1}^{\mathtt{local}}$. We denote the simulated trajectory as $\tau_i = (s_t, d_t, a_t, s_{t+1}, r_t, o_{t+1}, \ldots, a_{\mathcal{H}-1}, s_{\mathcal{H}}, r_{\mathcal{H}-1}, o_{\mathcal{H}})$. The empirical average KL divergence is then:

$$l_i = \frac{1}{\mathcal{H}-t} \sum_{k=t}^{\mathcal{H}-1} [-\log \hat{I}_\theta(s_k^{\mathtt{src}}|d_k) - H(S_k^{\mathtt{src}}|s_{k-1}, a_{k-1})]$$

4.3 Deciding between the Simulators

Finally, we want to use the estimated accuracy to decide whether to use the IALS or the global simulator for a simulation. However, alluded to before, the simulations from the GS will not only be used for planning but also for estimating the accuracy of the IALS, and this will need to be done every time we plan for a decision: Even if in the past we determined that $\hat{I}_\theta$ is accurate for the subtree of the current history h_t, it is possible that due to learning for other histories the accuracy for this h_t has decreased. Moreover, due to the constant updating of the search tree during a planning step, the tree policy also changes, which can also invalidate past accuracy estimates. As such, we face an ongoing exploration (assessing accuracy) and exploitation (of accurate $\hat{I}_\theta$) problem.

To address this, we propose to model the question of which simulator to use for the i-th simulation as a bandit problem, and apply the UCB1 algorithm [Auer *et al.*, 2002]. In the following, we define the values of using the global simulator $\mathcal{G}_{\mathtt{global}}$ and the IALS $\mathcal{G}_{\mathtt{IALS}}^\theta$ for every simulation i:

$$V_i^{\mathcal{G}_{\mathtt{IALS}}^\theta} = \hat{\mathcal{L}} + c^{\mathtt{meta}} \sqrt{\frac{\log(N^{\mathcal{G}_{\mathtt{IALS}}^\theta})}{i}}$$

$$V_i^{\mathcal{G}_{\mathtt{global}}} = -\lambda + c^{\mathtt{meta}} \sqrt{\frac{\log(N^{\mathcal{G}_{\mathtt{global}}})}{i}}$$

Here, $N^{\mathcal{G}_{\mathtt{global}}}$ and $N^{\mathcal{G}_{\mathtt{IALS}}^\theta}$ are the number of calls to the GS and the IALS, respectively. $\hat{\mathcal{L}}$ is an average of the empirical average KL divergence, and λ quantifies the extra computation cost of using the global simulator for a simulation, compared to the IALS. In practice, it is treated as a hyperparameter as it also reflects how willingly the user may sacrifice simulation accuracy for efficiency. Note that even though UCB1 is theoretically well understood, bounding the sample complexity here is impossible as both the history-visit distribution and the accuracy of the IALS can continually change.

5 Empirical Analysis

In this section, we first evaluate the main premise of our approach: *can selecting between a global simulator and an online learning IALS lead to a self-improving simulator?* Such improvement can manifest itself in faster decision making (when fixing the number of simulations per real time step), or in better performance (when fixing the time budget for each real time step). We investigate both. We also compare our online learning approach to the existing two-phase approach.

Experimental Setup   We perform the evaluation on two large POMDPs introduced by [He *et al.*, 2020], the Grab A Chair (GAC) domain and the Grid Traffic Control (GTC) domain, of which descriptions can be found in Appendix C.1. In all planning experiments with self-improving simulators, we start with an IALS that makes use of a completely untrained $\hat{I}_\theta$, implemented by a GRU; after every real episode it is trained for 64 gradient steps with the accumulated data from the global simulations. The results are averaged over 2500 and 1000 individual runs for the GAC and GTC domains, respectively. Further details are provided in Appendix C.

Online Planning with Fixed Number of Simulations   In this experiment, we fix the number of POMCP simulations to 100 per planning step. We ask two questions with this experiment: 1) can planning with self-improving simulators enable increasing planning efficiency? and 2) how does the choice of the hyperparameter λ affect the performance of planning? Since traffic domains have fixed time constraints, we only evaluate on GAC. In addition, GAC as the simpler domain allows us to explore many different settings for lambda λ.

Figure 2a shows how many of the 100 simulations are performed using the fast IALS (on average over all real time steps in the episode) for different λ. Remember that higher λ implies the planner is more willing to sacrifice the simulation accuracy for efficiency. For moderate λs, there is a clear trend that the use of IALS is increasing over time, due to the increasing accuracy of the approximate influence predictor $\hat{I}_\theta$, which is confirmed by the reduced training loss provided in Appendix D. This translates into the increasing planning speed shown in Figure 2b.[2] Note that large λs like 1.5 and 3.0 produce very fast planning directly from the first episode. However, as shown in Figure 2c, the price for that is poor performance at the earlier episodes, because they begin by heavily using the IALS with untrained influence predictors. Moreover, Figures 2b and 2c show that planning time for $\lambda = 1.0$ becomes comparable to $\lambda = 1.5$ and $\lambda = 3.0$ after roughly 10 episodes, while having much better performance at that point. This can be explained by the lack of training data caused by not using the global simulator often enough. Importantly, it seems that there is no clear sign of decreasing performance for moderate λs, which supports our key argument that in this setting, our approach enables planning with increasing speed without sacrificing task performance.

Real-Time Online Planning   In this experiment, we evaluate the performance of our approach under limited time budget. On the one hand, the IALS is an approximate model that introduces errors, certainly when the accuracy of its $\hat{I}_\theta$ is still poor in early episodes. On the other hand, using the faster IALS means that we can do many more simulations per real time step. As such, we investigate the ability of our proposed simulator selection mechanism to *dynamically* balance between these possible risks and benefits of the IALS.

We perform planning experiments in both the grab a chair domain and the grid traffic control domain, allowing $1/64$ and

---

$1/16$ seconds for each decision correspondingly, with results shown in Figure 3. We also compare against the baseline that only uses the global simulator and plot their performance as dotted lines. We see that there is an overhead with using the self-improving simulators, as shown by the differences in the starting number of simulations. This is the cost of the simulator selection method when estimating the KL divergence with the approximate influence predictor $\hat{I}_\theta$, which can potentially be reduced by a faster implementation of the GRUs. This gap is also expected to diminish in even larger domains.

In the grab a chair domain, we see that the self-improving simulators can perform more simulations for later episodes. This is expected: for later episodes the accuracy of the approximate influence predictor $\hat{I}_\theta$ improves, which means that the IALS is selected more frequently, thus enabling more simulations per real time step on average. The figure also shows that this larger number of planning simulations translates to better performance: for appropriate values of λ, after planning for a number of real episodes, the performance increases to a level that planning with the global simulator alone cannot reach. We stress that, in line with the results in the previous subsection, the increase in return is not *only* explained by doing more simulations: the quality of the $\hat{I}_\theta$ does matter, as is clearly demonstrated by $\lambda = 2.0$ in GAC: even though it performs many simulations per step from episode 1, its performance only surpasses that of the global simulator baseline after 20 episodes. Overall, these results show that self-improving simulators enables planning with increasing efficiency, and can outperform using an exact global simulator without pre-training an influence predictor.

Further comparison to the two-phase approach   Another limitation of the two-phase approach is: the offline learned influence predictors may generalize poorly when making online predictions due to the distribution shift issue caused by training with data collected by an exploration policy. We investigate if this is a real issue that can affect performance and if learning with online data, as in our approach, can help.

To evaluate the two-phase approach, we reuse the real-time planning setting in GTC. Following [He *et al.*, 2020], we start by collecting a number of datasets of varying sizes with a uniform random policy from the global simulator. We refer to these datasets as the offline data. For comparison, we also construct a number of datasets with different sizes from the replay buffer of the self-improving simulator (with λ=0.7), after planning for 1500 episodes. We refer to these datasets as the online data, because the data is generated during actual planning. Then, different influence predictors are trained on these datasets in an offline phase, before they are used as part of the IALSs for online planning, with results shown in Figure 2d. While it seems that more offline training data can be helpful, overall the performance of the influence predictors that are trained with online data completely dominates those that are trained with offline data: with much less online data, we can achieve a level of performance that is impossible with much more offline data. Figure 8 in Appendix shows that this indeed might be caused by a distribution shift issue: when evaluated on a test dataset that is collected by running POMCP on the global simulator, the influence predictors be-

---

[2]Planning time with λ=0.0 grows slowly due to the practical issue of memory fragmentation, which can be fixed with the memory pool technique from David Silver's implementation of POMCP.
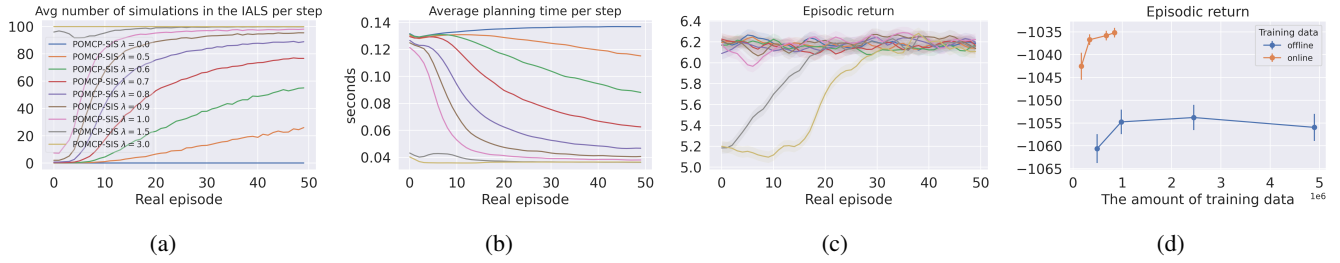
Figure 2: (a-c) Simulation controlled planning results for grab a chair. Planning time reduces without significant drop in return for small λ's. (d) Time controlled planning results for grid traffic control, with IALSs that make use of influence predictors trained on offline (from a uniform random policy) and online (from self-improving simulator with λ=0.7) data. This experiment is repeated for 20 times.

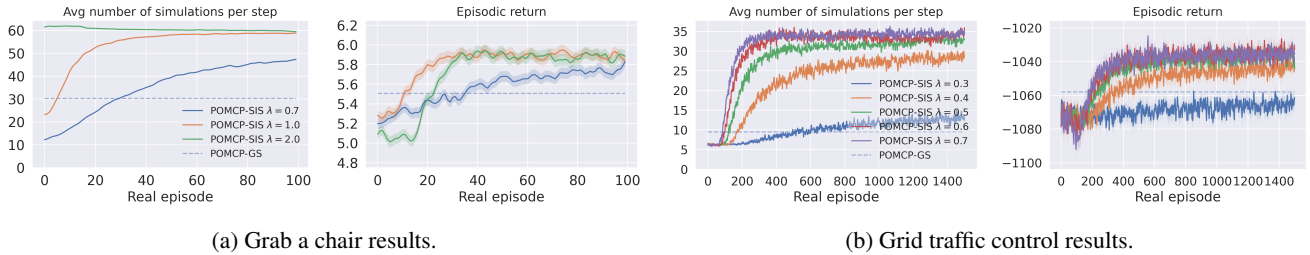(a) Grab a chair results.   (b) Grid traffic control results.

Figure 3: Time controlled planning results. Performance improves because of increasingly accurate IALS and more simulations.

ing trained on the offline data can have an increasing test loss. Moreover, training on the online data from the self-improving simulator results in a much lower test loss. As such, we can conclude that there is indeed a distribution shift issue with the two-phase approach that can harm the planning performance, which is addressed in our approach.

6 Related Work

Surrogate models have been widely used in many applications, such as aerodynamic design prediction [Pehlivanoglu and Yagiz, 2012], spacecraft path planning [Peng and Wang, 2016] and satellite reconfiguration [Chen *et al.*, 2020]. It has been proposed as a general approach to constraint optimization problems [Audet *et al.*, 2000].

There is a strong connection between our work and multi-fidelity methods [Kennedy and O'Hagan, 2000; Peherstorfer *et al.*, 2018], which combine models with varying costs and fidelities to accelerate outer-loop applications such as optimization and inference. In essence, the simulator selection mechanism in our approach does the same: It selects between a slow high-fidelity global simulator and a fast low-fidelity IALS, to accelerate planning. In our case however, the fidelity of the IALS may change over time due to the training of the influence predictor, and also depends on the current context. While our simulator selection mechanism is based on an accuracy measure that is theoretically inspired [Congeduti *et al.*, 2021], future work could incorporate ideas from multi-fidelity methods more explicitly.

There is also a body of work that applies abstraction methods inside MCTS [Hostetler *et al.*, 2014; Jiang *et al.*, 2014; Anand *et al.*, 2015b; Anand *et al.*, 2015a; Anand *et al.*, 2016; Bai *et al.*, 2016]. Such methods perform abstraction on the MCTS search tree: they aggregate search tree nodes, thus reducing the size of search tree. This is complementary to our technique, which speeds up the simulations themselves.

7 Conclusion

In this paper, we investigated two questions. First, given a global simulator (GS) of a complex environment formalized as a factored POMDP, can we learn online a faster approximate model, i.e., a so-called IALS? Second, can we balance between such a learning IALS and the GS for fast *and* accurate simulations to improve the planning performance?

The answers to these questions lead to a new approach called *self-improving simulators*, which selects between the online learning IALS and the GS for every simulation, based on an accuracy measure for IALS that can be estimated online with the GS simulations. As the accuracy of the influence predictor $\hat{I}_\theta$ would be poor initially, the GS will be used mainly for simulations in earlier episodes, which can then be used to train $\hat{I}_\theta$. This leads to increasing simulation efficiency as the IALS becomes more accurate and is used more often, which improves planning performance (time and return) over time.

Our approach has three main advantages against the two-phase approach, which trains the influence predictor offline before using it for planning [He *et al.*, 2020]. First, planning can start without pre-training. Second, learning $\hat{I}_\theta$ online prevents the distribution shift issue that occurs when transferring the offline learned $\hat{I}_\theta$ to make online predictions. Third, selecting the simulator online enables reverting back to the GS when the IALS is not sufficiently accurate for the current context. Future work can investigate different ways of estimating the IALS accuracy that make less use of the expensive GS.

Acknowledgements

References

[Anand *et al.*, 2015a] Ankit Anand, Aditya Grover, and Parag Singla. A novel abstraction framework for online planning. In *AAMAS*, 2015.

[Anand *et al.*, 2015b] Ankit Anand, Aditya Grover, Parag Singla, et al. ASAP-UCT: Abstraction of state-action pairs in UCT. In *IJCAI*, 2015.

[Anand *et al.*, 2016] Ankit Anand, Ritesh Noothigattu, Parag Singla, et al. OGA-UCT: On-the-go abstractions in UCT. In *ICAPS*, 2016.

[Audet *et al.*, 2000] Charles Audet, J Denni, Douglas Moore, Andrew Booker, and Paul Frank. A surrogate-model-based method for constrained optimization. In *8th symposium on multidisciplinary analysis and optimization*, page 4891, 2000.

[Auer *et al.*, 2002] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, may 2002.

[Bai *et al.*, 2016] Aijun Bai, Siddharth Srivastava, and Stuart J Russell. Markovian state and action abstractions for MDPs via hierarchical MCTS. In *IJCAI*, 2016.

[Boutilier *et al.*, 1999] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *JAIR*, 11, 1999.

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[Buesing *et al.*, 2018] Lars Buesing, Theophane Weber, Sébastien Racaniere, SM Eslami, Danilo Rezende, David P Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, et al. Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*, 2018.

[Chen *et al.*, 2020] Qifeng Chen, Caisheng Wei, Yongfeng Shi, and Yunhe Meng. Satellite swarm reconfiguration planning based on surrogate models. *Journal of Guidance, Control, and Dynamics*, 43(9):1750–1756, 2020.

[Chitnis and Lozano-Pérez, 2020] Rohan Chitnis and Tomás Lozano-Pérez. Learning compact models for planning with exogenous processes. In *CoRL*, 2020.

[Cho *et al.*, 2014] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

[Congeduti *et al.*, 2021] Elena Congeduti, Alexander Mey, and Frans A Oliehoek. Loss bounds for approximate influence-based abstraction. In *AAMAS*, 2021.

[Grzeszczuk *et al.*, 1998] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 1998.

[Hansen and Feng, 2000] Eric A Hansen and Zhengzhu Feng. Dynamic programming for POMDPs using a factored state representation. In *AIPS*, 2000.

[He *et al.*, 2020] Jinke He, Miguel Suau de Castro, and Frans Oliehoek. Influence-augmented online planning for complex environments. In *NeurIPS*, 2020.

[Hostetler *et al.*, 2014] Jesse Hostetler, Alan Fern, and Tom Dietterich. State aggregation in Monte Carlo tree search. In *AAAI*, 2014.

[Jiang *et al.*, 2014] Nan Jiang, Satinder Singh, and Richard Lewis. Improving UCT planning via approximate homomorphisms. In *AAMAS*, 2014.

[Kaelbling *et al.*, 1998] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, may 1998.

[Kennedy and O'Hagan, 2000] Marc C Kennedy and Anthony O'Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, 2000.

[Oliehoek *et al.*, 2021] Frans Oliehoek, Stefan Witwicki, and Leslie Kaelbling. A sufficient statistic for influence in structured multiagent environments. *JAIR*, 70, 2021.

[Paninski, 2003] Liam Paninski. Estimation of entropy and mutual information. *Neural computation*, 2003.

[Peherstorfer *et al.*, 2018] Benjamin Peherstorfer, Karen Willcox, and Max Gunzburger. Survey of multifidelity methods in uncertainty propagation, inference, and optimization. *Siam Review*, 60(3):550–591, 2018.

[Pehlivanoglu and Yagiz, 2012] Y Volkan Pehlivanoglu and Bedri Yagiz. Aerodynamic design prediction using surrogate-based modeling in genetic algorithm architecture. *Aerospace Science and Technology*, 2012.

[Peng and Wang, 2016] Haijun Peng and Wei Wang. Adaptive surrogate model-based fast path planning for spacecraft formation reconfiguration on libration point orbits. *Aerospace Science and Technology*, 54:151–163, 2016.

[Ruder, 2016] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, sep 2016.

[Silver and Veness, 2010] David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. In *NeurIPS*, 2010.

[Somani *et al.*, 2013] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. In *NeurIPS*, 2013.

[Spaan, 2012] Matthijs TJ Spaan. Partially observable markov decision processes. In *Reinforcement Learning*, pages 387–414. Springer, 2012.