In NeurIPS 2023 Workshop on Generalization in Planning (GenPlan 2023)
New Orleans, LA, USA

Epistemic Exploration for Generalizable Planning and Learning in Non-Stationary Stochastic Settings

Rushang Karia*, Pulkit Verma*, Gaurav Vipat, Siddharth Srivastava
School of Computing and Augmented Intelligence
Arizona State University
Tempe, AZ 85281
`{rushang.karia,verma.pulkit,gvipat,siddharths}@asu.edu`

Abstract

Reinforcement Learning (RL) provides a convenient framework for sequential decision making when closed-form transition dynamics are unavailable and can frequently change. However, the high sample complexity of RL approaches limits their utility in the real-world. This paper presents an approach that performs meta-level exploration in the space of models and uses the learned models to compute policies. Our approach interleaves learning and planning allowing data-efficient, task-focused sample collection in the presence of non-stationarity. We conduct an empirical evaluation on benchmark domains and show that our approach significantly outperforms baselines in sample complexity and easily adapts to changing transition systems across tasks.

1 Introduction

Sequential Decision Making (SDM) provides a convenient and flexible framework for computing solutions to real-world problems in a cornucopia of settings. There are several considerations that influence the design of techniques that solve SDM problems: (a) Are states of the problem fully observable?, (b) Is the transition system known and available in closed-form, (c) Is the reward system known and available in closed-form? and (d) Are the states atomic or represented using factored representations?

Reinforcement Learning (RL) allows solving problems where the states are fully observable but the transition and reward system are only available as simulators and are not available in closed-form. Furthermore, RL also allows transition systems to change arbitrarily. RL methods typically interact with the simulator to collect experience and use this experience to solve the problem. One key metric in determining the efficacy of an RL approach is the *sample complexity* - the amount of experience needed to be collected - that the RL method requires in order to effectively solve the task. Due to their low input requirements, RL is widely applicable to a large proportion of real-world SDM problems and researchers have invested significant effort into improving the sample complexity of RL methods [Watkins, 1989, Mnih et al., 2013, Schulman et al., 2017].

Another important aspect in SDM approaches is the representational language that is used to represent states of problems. Approaches that utilize image-based representations have been demonstrated to successfully solve RL problems [Mnih et al., 2013]. However, many real-world problems are more intuitively represented using factored representations using a symbolic language such as first-order logic. For example, databases that store the inventory, order information, customer information etc. of an e-commerce website such as Amazon is readily expressed as a relational representation. Converting these factored representations into image-based inputs to RL approaches requires significant domain-

[*]These authors contributed equally to this work

specific knowledge and is quite difficult. As a result, researchers have developed several SOTA methods that solve RL problems expressed using such relational representations [Dzeroski et al., 2001, Tadepalli et al., 2004, Karia and Srivastava, 2022]. We provide a complete overview of related work in Sec. 5.

In this paper, we focus on problem settings where the states are fully observable and relational, and the user is presented with a stream of different tasks to accomplish. The transition system is unknown and can change arbitrarily between tasks. The overall objective is to enable solving all tasks in a sample-efficient fashion. This scenario typically captures many real-world applications of AI systems.
Running Example As a running example, consider a robot that is to be deployed at a warehouse to assist with solving tasks. The robot does not know in advance the set of tasks to solve. Additionally, rewards are task-dependent and can change based on the current task. Finally, as the robot is interacting with the environment, the transition dynamics can arbitrarily change, e.g., the coefficient of friction of the floor changes.

RL approaches like Q-Learning [Watkins, 1989] cannot effectively transfer between tasks since they do not learn any meta-level knowledge that can benefit later tasks. Model-based RL (MBRL) learns meta-level knowledge in the form of transition models and can use them to transfer and bootstrap the solutions for later tasks. However, existing MBRL approaches are not sample efficient since they are often not task-focused and need to learn the complete model. Additionally, they cannot efficiently handle changes in the transition systems requiring the learning process to be re-invoked from scratch leading to increased sample complexity.

We solve this problem by developing an approach that interleaves planning and learning to compute solutions in a few-shot fashion. We accomplish this by learning the transition dynamics (or models) by performing a search at the meta-level, i.e., in the space of possible models. In order to do learn effectively, we utilize active learning to efficiently interact with the simulator in order to learn models in a sample-efficient fashion. We use the planned models to compute policies bypassing RL completely. We conduct an empirical evaluation on three benchmark domains and show that our approach significantly outperforms baselines in terms of the sample complexity.

The rest of the paper is organized as follows. Sec. 2 provides the necessary background. We describe our approach in Sec. 3. We discuss our empirical setup and obtained results in Sec. 4. We then compare and contrast our work with related work in the area (Sec. 5) followed by a brief discussion of conclusions and possible future work (Sec. 6).

2 Background

Relational Markov Decision Processes (RMDPs). Our focus is on RMDPs expressed in a symbolic representational language such as the Probabilistic Planning Domain and Definition Language (PPDDL) [Younes et al., 2005]. An RMDP problem consists of two components. An RMDP *domain* $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ is a tuple consisting of a set of lifted predicates $\mathcal{P}$and a set of parameterized actions $\mathcal{A}$. An RMDP problem is defined as a tuple $M = \langle \mathcal{D}, O, S, A, \delta, R, s_0, g, \gamma, H \rangle$ where O is a set of objects. An atom $p(o_i, \ldots, o_n)$ where $p \in \mathcal{P}$ and $o_i \in O$ is formed by grounding p with objects from O. A state s is a collection of such atoms. The set of all possible subsets of atoms forms the state space S of the RMDP M. Similarly, the action space A of M is formed by instantiation each parameterized action $a \in \mathcal{A}$ with the appropriate number of objects from O. $\delta : S \times A \times S \rightarrow [0, 1]$ is the transition function and is implemented by a simulator. For a given *transition* (s, a, s'), $\delta(s, a, s')$ specifies the probability of executing action $a \in A$ in a state $s \in S$ and reaching a state $s' \in S$. Naturally, $\sum_{s' \in S} \delta(s, a, s')$ for any $s \in S$ and $a \in A$. $R : S \times A \rightarrow \mathbb{R}$ is the reward function and $r(s, a)$ indicates the reward obtained for executing action a in state s. s_0 is the initial state and g is an optional conjunctive first-order logic goal formula using $\mathcal{P}$and O. A goal state $s_g \in S$ is a state such that $s_g \models g$. $\gamma \in [0, 1)$ is the discount factor and $H \in \mathbb{N} \cup \infty$ is the horizon. For an RMDP, execution begins in the initial state and terminates when a goal state is reached or when the horizon limit has been exceeded.
Example The amazon warehouse robot described earlier could be described using a domain consisting of predicates $\{\texttt{robot-at}(r_x, l_y), \texttt{package-at}(r_x, p_y), \texttt{holding}(r_x, p_y), \texttt{handempty}(r_x)\}$ where r_x, l_y, p_y are parameters of the predicates that can be substituted with objects. These predicates help describe the state of environment. For example, the predicate $\texttt{holding}(r_x, p_y)$ helps describe whether the robot r_x is holding a particular package p_y. The action set would consist of actions such

as $\texttt{move-from}(r_x, l_x, l_y)$, $\texttt{pick-up}(r_x, l_x, p_x)$ etc. with their dynamics implemented by a simulator. For example, the action $\texttt{pick-up}(r_x, l_x, p_x)$ would instruct the robot r_x to pickup package p_x at location l_x. The simulator dynamics could model the robot's gripper as slippery such that grapsing p_x could fail with a certain probability leaving the state unchanged.

A solution to an RMDP is a *deterministic policy* $\pi : S \to A$ that maps states to actions. The value of a state s when following a policy π is defined as the expected reward obtained when executing $a = \pi(s)$ in s and following π thereafter, i.e., $V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} \delta(s, a, s') V^\pi(s')$ The objective of an RMDP is to compute an *optimal* policy π^* that maximizes the expected reward obtained by following it.[*] RMDP algorithms compute $\pi^*(s_0)$ by solving the *Bellman Optimality Equation* iteratively starting from s_0 [Sutton and Barto, 1998]:

$$V^*(s) = \max_a \left[r(s, a) + \gamma \sum_{s' \in S} \delta(s, a, s') V^*(s') \right]$$

The above equation requires access to closed-form knowledge of the transition function. When such information is unavailable, RMDP algorithms uses Q-values of states instead. The Q-value of a state s when executing action a in state s is defined as the expected reward obtained when executing a in s and following the policy thereafter, i.e. $Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{H} \gamma^t R(S_t, A_t) | S_0 = s, A_0 = a \right]$.

$$Q^*(s, a) = Q(s, a) + \alpha \left[R(s, a) + \gamma \max_{a' \in A} Q(s', a') - Q(s, a) \right]$$

The latter half of the equation above is known as the temporal update, TD(0) rule for updating the Q-values. $\alpha \in [0, 1]$ is the learning rate. Algorithms using TD(0) with a decaying learning rate have been shown to converge to the optimal policy [Sutton and Barto, 1998]. Thus, many RL algorithms compute the optimal policy by applying temporal difference updates to the Q-values of states

PPDDL transition models Our approach learns interpretable PPDDL transition models that can be used for stochastic planning using the Bellman optimality equations. Given a domain $\mathcal{D}$, a PPDDL model for a parameterized action $a(o_1, \ldots, o_n) \in \mathcal{A}$ is a tuple $\langle \mathit{Pre}, \mathit{Prob}, \mathit{Eff} \rangle$ where the precondition *Pre* is a conjunctive formula of literals $l \in \mathcal{P}$ parameterized using the parameters. *Prob* is a list of probabilities such that $\sum_i \mathit{Prob}[i] = 1$. *Eff* is a list of effects where each effect $\mathit{Eff}[i] \in \mathit{Eff}$ is a set of literals parameterized using the parameters. We omit the parameters when it is clear from context. An action a is *applicable* in a state s iff $s \models \mathit{Pre}$. An effect $\mathit{Eff}[i]$ when applied to a state s results in a state $s \setminus \mathit{Eff}[i]^- \cup \mathit{Eff}[i]^+$ where $\mathit{Eff}[i]^-$ ($\mathit{Eff}[i]^+$) are the set of negative (positive) literals of *Eff*[i]. Applying an action a to a state s results in effect *Eff*[i] being applied with probability *Prob*[i] if the action is applicable else the state remains unchanged.

Example The $\texttt{pick-up}(r_x, l_x, p_x)$ action described in the running example could be modeled as a PPDDL model where $\mathit{Pre} = \{\texttt{robot-at}(r_x, l_x), \texttt{handempty}(r_x), \texttt{package-at}(p_x, l_x)\}$ to indicate that the action is applicable only when the robot is not holding anything and when both it and the package are at the same location. The effects could be modeled as $\mathit{Eff}[0] = \{\neg\texttt{package-at}(p_x, l_x)$, $\texttt{holding}(r_x, p_x)$, $\neg\texttt{handempty}(r_x)\}$ to indicate that the robot successfully picked up the package and is currently holding it. Similarly, another effect $\mathit{Eff}[1] = \{\}$ with $\mathit{Prob}[1] = 0.1$ could be used to model a slippery gripper that would leave the state unchanged with a 10% chance (for example if the robot could not grip the object firmly to pick it up).

3 Our Approach

We now begin by describing the problem that we address followed by a detailed overview of our approach.

3.1 The Task Transfer Problem

Given any two RMDP's M_i and M_j, we define $M_i \neq M_j$ iff any one of the following conditions is met. (a) the domains are different, $\mathcal{D}_{M_i} \neq \mathcal{D}_{M_j}$, (b) the objects are different $O_{M_i} \neq O_{M_j}$, (c) the transition systems are different, $\delta_{M_i} \neq \delta_{M_j}$, (d) the reward systems are different $R_{M_i} \neq R_{M_j}$ or (e) the initial state and/or goal formulae are different.

[*]We limit ourselves to optimal policies that are optimal w.r.t. the initial state s_0.

Algorithm 1 Interleaved learning and planning

```
Require: RMDP $M$ and its Simulator $\Delta_M$, Simulator Budget $\mathcal{S}$, Learned Model $\delta'$
1: $\pi \leftarrow$ computePolicy$(\delta', M)$
2: $s \leftarrow s_0$
3: $step \leftarrow 0$
4: $h \leftarrow 0$
5: while $step < \mathcal{S}$ do
6:     $step \leftarrow step + 1$
7:     $h \leftarrow h + 1$
8:     $a \leftarrow \pi(s)$
9:     $s' \leftarrow \Delta_M(s, a)$
10:    if transitionConforms$(s, a, s', \delta')$ then
11:        $\delta' \leftarrow$ updateMLECounts$(s, a, s')$
12:    else
13:        $\delta' \leftarrow$ QACE$(s, a, s', \delta')$
14:        $\pi \leftarrow$ computePolicy$(\delta', M)$
15:    end if
16:    if $s \models g$ or $h \geq H$ then
17:        $s \leftarrow s_0$
18:    else
19:        $s \leftarrow s'$
20:    end if
21: end while
22: return $\delta'$
```

Given a stream of RMDP tasks $\mathcal{M} = \langle M_1, \ldots, M_n \rangle$ where $M_{i-1} \neq M_i \neq M_{i+1}$ and a simulator budget of $\mathcal{S}$ steps per RMDP M_i, our objective is to learn meta-level knowledge that can transfer across RMDPs to maximize the total tasks completed within the total simulator budget.

3.2 Active Query-based Model Learning

We improve and adapt our work on QACE; an active query-based model learning approach that has been demonstrated to be efficient in capability assessment [Verma et al., 2023]. The vanilla version of QACE works as follows. Any given predicate can appear in three modes $(+|-|\emptyset)$ in a precondition or effect of an action. Either (a) the predicate is required to be true $(+)$ for an action to be applicable or the predicate becomes true on application of the effect, (b) the predicate is required to be false $(-)$ for an action to be applicable or the predicate becomes false on application of the effect, and (c) the predicate can appear in any mode $(\emptyset)$ for the action to be applicable and application of the effect leaves the predicate unchanged.

QACE prunes the space of possible models by formulating *distinguishing* queries (different versions of models differing in the mode of a single predicate) that allow to effectively prune the mode of a predicate in a precondition or effect. These queries are computed using a forward search algorithm like PRP that computes a policy that helps prune out the mode of the predicate that does not conform to the simulator's model. Finally, QACE learns probabilities of these models by using Maximum Likelihood Estimation and using the simulator to repeatedly execution the same action.

The vanilla version of QACE does not adapt well to the task transfer setting since all actions must be relearned from scratch. We modify QACE to only consider actions that do not conform with the model. For example, if a predicate p that occurs in the precondition of an action a_i of an RMDP M_i does not appear in the precondition of an action a_i in RMDP M_{i+1}, then a will no longer be applicable in M_{i+1}. We flag the action a as non-conforming and run QACE to learn learn a assuming all other actions as conforming. This procedure can repeatedly cause QACE to relearn many actions that have encountered a change in their dynamics.

3.3 Interleaved learning and planning

The key observation to solving RMDPs is that if models are available then the sample efficiency can be significantly improved since these models can be provided to model-based RMDP solvers that use the Bellman equations in computing optimal policies. There exist many methods for learning models effectively but they suffer from several drawbacks that make them unsuitable for use in the task transfer problem. Firstly, most methods require an expert dataset of positive and negative transitions to effectively learn models. We mitigate this by using an intelligent active learning model-learning process that guide the exploration process to learn data that enables sample efficient model learning. Second, model-learning approaches often compute a complete model. This is not desirable since a tasks appear as a stream and have a limited budget. Our approach only learns models that are required to solve the task. This aspect is further exacerbated by non-stationarity that can render large parts of the learning process obsolete. We intelligently detect changes in models and adapt the learned models using few-shot sampling. We now describe our approach for active model learning and our overall process for interleaved learning and planning.

Alg. 1 describes our overall process for interleaved learning and planning. The algorithm takes as input an RMDP task M, a simulator Δ_M for M and the simulator step budget $\mathcal{S}$ for the task and a learned model δ'. This is easily computed by using the transition (s, a, s') and δ' and checking if applying a to s using δ' would be able to yield s'. Initially, the learned model is empty. Line 1 uses δ' to compute a policy for M using a model-based RMDP solver such as LAO* [Hansen and Zilberstein, 2001]. Lines 2-4 initialize state variables. As long as the simulator budget is remaining, the algorithm tries to follow the computed policy using the simulator (lines 8-9). Once an action a is executed in a state s, line 10 checks if the transition (s, a, s') conforms with δ'. If it does, then the MLE counts are simply updated (line 11). If it does not conform, then that means that δ' does not match the dynamics of Δ_M. In that case, QACE is re-run with the non-conforming transition to relearn the correct dynamics of a (line 13). Once this is done, a new policy for M is computed (line 14). Finally, the state is reset to the initial state or s' depending upon whether the goal was reached or if the horizon was exceeded (lines 16-20). Once the simulator budget has been exhausted, the algorithm returns the learned model δ' (line 22). A new RMDP $M' \neq M$ can use the learned model δ' to continue the interleaved learning and planning process.

4 Experiments

We implemented our approach (Alg. 1) in Python and performed an empirical evaluation on three benchmark domains using an AMD Ryzen 5 5500 CPU with 64 GB of RAM. We found that our approach leads to significantly better transfer performance as compared to the baselines. We describe our empirical setup that we used for conducting the experiments followed by a discussion of the obtained results (Sec. 4.1).

Tasks We used benchmark domains from the International Probabilistic Planning Competition (IPPC) [Long and Fox, 2003, Younes et al., 2005] and Icarte et al. [2018] for our experiments. Since the transition dynamics of these domains are typically fixed, we modified them to allow a thorough evaluation of the task transfer problem that we focus on in this paper.

Simple Office World We modified the popular Officeworld environment [Icarte et al., 2018] for the task transfer problem. We used a 7×7 grid-based environment with locations containing resources such as coffee, mail etc. Certain grid locations are special locations and are prefixed as such, e.g., the office. Example tasks in this domain include delivering coffee to the office, delivering coffee and mail to the office etc. We also modified this domain to have stochasticity in the effects of the actions. For example, while picking up an object, we modeled a slippery gripper that could fail to grasp the object with a probability of 0.1.

We used three RMDPs (tasks) in this domain where the initial states and goals were different for all three tasks. The first task used the officeworld domain along with the modifications we described above. The second task added new actions such as doors and keys to the locations so that new dynamics were introduced. The third task removed the slippery gripper and reconstructed the original officeworld environment.

Logistics + Stacking This is the classical Logistics domain where packages need to be delivered to their destinations via trucks or airplanes. We modified this domain to be stochastic with loading and unload actions succeeding with an eighty percent chance.

This domain contained four tasks. The first two tasks are logistics tasks that simply involve delivering packages to locations. However, the second task changes the probability distributions so that it is advantageous to use road transport. The latter two tasks add stacking actions so that packages not only need to be delivered to their destination, they also need to be stacked on pallets in a specific order. An unlimited number of packages can be stacked at once. Finally, the last task modifies the effects of stacking so that only a single package can be stacked at a time.

First Responders This domain involves tasks that involve emergency services such as putting out the fire in the building and treating any burn victims either on-site or transporting them to the hospital. There is an element of resource management since the fire hydrant can run out with a specific probability. We used three tasks for this domain. The first task only requires putting out fires whereas the second task requires saving civilians as well. Finally, the last task changes the medic actions so that treatment can fail and they need to be transported to the hospital.

Baselines For our baselines, we used Q-Learning [Watkins, 1989] configured with a learning rate $\alpha = 0.3$. We also used our work on QACE [Verma et al., 2023] as a second baseline in a learn-then-plan setting. This baseline, which we call learn-then-plan, performs model-learning on each task from scratch using QACE. Once a model is learned we compute a policy using the LAO* [Hansen and Zilberstein, 2001] RMDP solver and follow it until the step budget is exhausted for the task.

Evaluation Metric All tasks in our setup are goal-oriented, sparse reward tasks where a reward of 0 is awarded once a goal state is reached else a reward of -1 is awarded for every executed action. We used a horizon of 40 and a fixed simulator budget $\mathcal{S} = 5000$ for each task. Once the simulator budget is exhausted the simulator is loaded with the new task. As a result, we evaluate each method w.r.t. the total number of tasks that they were able to complete within the simulator's step budget. We consider a task to be completed iff a goal state is reached after which simulator is reset to the initial state of the task.

4.1 Analysis of Results

Fig. 1 shows the obtained results. We now provide an analysis of the results.

It is clear from Fig. 1 that our approach of interleaved planning and learning outperforms both non-transfer based approaches and approaches that learn the complete model and then plan. For example, in the Simple Office World domain, our approach adapts immediately to Task 2 where there are keys and doors since it only learns the newly introduced actions. The Learn-then-plan baseline on the other hand expends effort completely learning the model from scratch including actions whose dynamics did not sufficiently change.

Our approach significantly outperforms all baselines in First Responders. In fact, Q-Learning was not able to solve tasks in this domain since it has a very high branching factor. First Responders is a relatively difficult domain for model learning since there is a fair bit of exploration that needs to be performed for learning the complete model. This is evident from our results, where the learn-then-plan approach repeatedly expends effort in this costly exploration process relearning the model whereas our approach only learns when failures occur and is able to adapt much quickly.

Our approach uses a greedy task-focused approach to solving RMDP tasks. Once a policy is computed and provided the simulator's dynamics agree with the learned model we save on learning updates compared to RL algorithms by directly computing policies without interaction with the simulator. This allows our approach to quickly find models that can solve the tasks and identify any drift in the transition models quickly.

Limitations There are some key limitations that need to be addressed in future work. For example, in task 2 of logistics and stacking, our approach was unable to outperform the learn-then-plan approach. In this task, policies using the airplanes are penalized significantly and good policies prefer the use of road transportation.

The learn-then-plan approach directly learns a new model from scratch and therefore RMDP solvers using this model can find good policies. Our approach uses the probability distributions from the earlier tasks and the MLE estimates are too slow to converge to the new values thus leading to poor policies being computed and as a consequence fewer tasks being completed.

When is it better to interleave learning and planning We now turn to an important question that believe needs to be answered. Under what conditions is it better to restart the model learning process from scratch? We are able to see that when the transition dynamics change only in the distribution

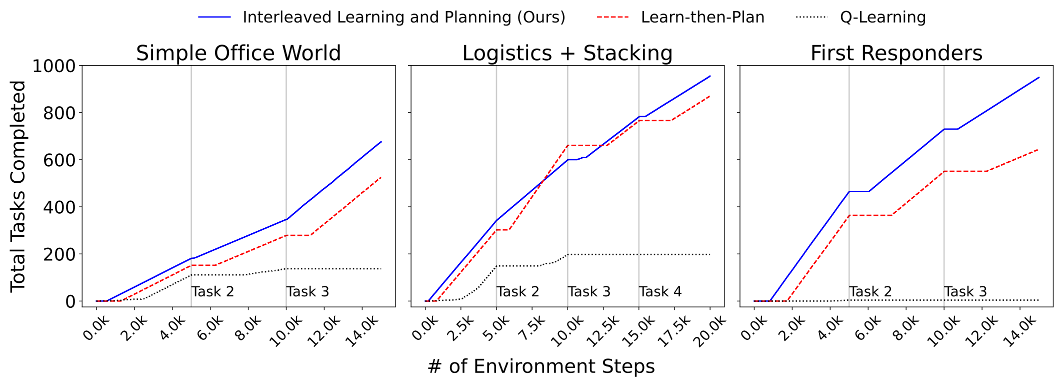

Figure 1: Results from our experiments (higher values better). The x-axis corresponds to the number of steps performed using the simulator (in thousands). The y-axis represents the total number of tasks successfully completed. Vertical lines indicate the end of the simulators budget for an RMDP task and the annotations describe the next task that starts (we always start with Task 1).

and not the perceived (local) behavior then it is difficult to assess whether the global behavior has been impacted. As seen in logistics, the local behavior did not change but significantly impacted the global behavior. However, it was easier to adapt to the stacking task (Task 3) rather than starting the learning process from scratch (the handicap on air transport was lifted in Task 3). Thus, for related tasks where transition dynamics do not change significantly it is often easier to adapt to new tasks by retaining the meta-level knowledge and fix this knowledge when failures occur.

Naturally, in a controlled environment where transition dynamics are not expected to change significantly, it is easier to adapt to changes rather than use a costly learn-from-scratch approach to every task. Another benefit of our approach is the task-focused aspect which only tries to learn the minimal information needed to solve the task. Complete model learners would often waste exploration effort on learning actions that do not play a role in solving the task. This issue is compounded by the fact that there is a limited simulator budget and that transition dynamics are non-stationary. However, greedy task-focused approaches can lead to sub-optimal behavior since the learned actions need not result in optimal policies.

Finally, most model-learners learn through exploration. When exploration is tough, model learners will expend tremendous effort for learning the model as is seen in First Responders. In such cases, it would be better to interleave learning and planning approaches that can bootstrapthe RMDP process with some useful knowledge without significant use of the simulator's resources.

A formal characterization of this question with known priors about the non-stationarity and the task distribution is an interesting research direction that we leave to future work.

5 Related Work

There has been plenty of work for transfer in RL. We focus on approaches that focus on transfer between relational RL tasks. Tadepalli et al. [2004] provides an extensive overview for relational RL approaches.

The Dyna framework [Sutton, 1990] forms the basis of many model-based reinforcement learning (MBRL) approaches wherein experience from the environment is used to simultaneously learn a model and use the model to generate synthetic experience that is used for learning updates. Ng and Petrick [2021] use conjunctive first-order features to learn models and learn generalizable policies that transfer to related classes of RMDPs. Their approach is unable to efficiently learn models and relies on the exploration of the RL process for generating data. The policies they learn can only transfer to related RMDPs with the same domain and transition function. Our approach can easily and automatically handle changes between RMDPs even when the domains change between tasks. REX [Lang et al., 2012] enables MBRL to automatically learn tasks autonomously. One challenge with this approach is the ability to learn accurate models since exploration can be scarce when using REX.

V-MIN [Martínez et al., 2017] integrates model-learning and planning with the Relational Dynamic Influence Diagram Language (RDDL) for reinforcement learning by requesting demonstrations from a teacher if it cannot find a policy whose expected value is greater than a certain threshold. The requirement of an available teacher limits the transfer capabilities of this approach. Taskable RL (TRL) [Illanes et al., 2020] and RePReL [Kokel et al., 2023] show how Hierarchical Reinforcement Learning (HRL) using the options framework can be used for taskable RL (TRL). They use symbolic plans to guide the RL process. A key disadvantage of their approach is that the models are provided as input and are not learned. Our work does not require any expert knowledge and generates its own data for learning models using an active learning process.

GRL [Karia and Srivastava, 2022] train a neural network to learn reactive policies that can transfer to problems from the same domain but with different state spaces. Their approach is limited to only changes in the state space and cannot adapt to changes in the transition dynamics. DAAISy [Nayyar et al., 2022] learn models for non-stationary environments that can be integrated into the interleaved learning and planning loop. However, their approach only learns deterministic models and requires the use of optimal agents and observation traces to identify changes in transition dynamics. Our approach can organically learn models during the RL process.

6 Conclusions and Future Work

We developed a sample-efficient method for transferring epistemial knowledge between by an interleaved learning and planning process. Our approach can easily handle non-stationary environments, is task-focused and our results show that it significantly outperforms baselines.

Future Work There are several interesting research directions for future work. Firstly, a formal characterization of what kinds of differences between RMDPs allow for better transfer using the interleaved process would aid AI systems in deciding to use interleaved learning and planning or use a model-learning process from scratch. Next, given a prior on RMDP's would it be possible to optimize the learning process to automatically generate a curriculum that allows sample efficient learning is another interesting direction.

References

S. Dzeroski, L. D. Raedt, and K. Driessens. Relational reinforcement learning. *Mach. Learn.*, 43 (1/2):7–52, 2001.

E. A. Hansen and S. Zilberstein. Lao: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proc. ICML*, 2018.

L. Illanes, X. Yan, R. T. Icarte, and S. A. McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *Proc. ICAPS*, 2020.

R. Karia and S. Srivastava. Relational abstractions for generalized reinforcement learning on symbolic problems. In *Proc. IJCAI*, 2022.

H. Kokel, S. Natarajan, B. Ravindran, and P. Tadepalli. Reprel: a unified framework for integrating relational planning and reinforcement learning for effective abstraction in discrete and continuous domains. *Neural Comput. Appl.*, 35(23):16877–16892, 2023.

T. Lang, M. Toussaint, and K. Kersting. Exploration in relational domains for model-based reinforcement learning. *J. Mach. Learn. Res.*, 13:3725–3768, 2012.

D. Long and M. Fox. The 3rd international planning competition: Results and analysis. *J. Artif. Intell. Res.*, 20:1–59, 2003.

D. M. Martínez, G. Alenyà, T. Ribeiro, K. Inoue, and C. Torras. Relational reinforcement learning for planning with exogenous effects. *J. Mach. Learn. Res.*, 18:78:1–78:44, 2017.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *arXiv*, abs/1312.5602, 2013.

R. K. Nayyar, P. Verma, and S. Srivastava. Differential assessment of black-box AI agents. In *Proc. AAAI*, 2022.

J. H. A. Ng and R. Petrick. First-order function approximation for transfer learning in relational mdps. In *ICAPS PRL workshop*, 2021.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In B. W. Porter and R. J. Mooney, editors, *Machine Learning*, 1990.

R. S. Sutton and A. G. Barto. *Reinforcement learning - an introduction*. MIT Press, 1998. ISBN 978-0-262-19398-6.

P. Tadepalli, R. Givan, and K. Driessens. Relational reinforcement learning: An overview. In *ICML RRL workshop*, 2004.

P. Verma, R. Karia, and S. Srivastava. Autonomous capability assessment of sequential decision-making systems in stochastic settings. In *Proc. NeurIPS*, 2023.

C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

H. L. S. Younes, M. L. Littman, D. Weissman, and J. Asmuth. The first probabilistic track of the international planning competition. *J. Artif. Intell. Res.*, 24:851–887, 2005.