Model-Based Adaptation to Novelty in Open-World AI

Submission # 7490

Abstract

Most model-based agents are ill-equipped to handle novel situations in which their model of the environment no longer sufficiently represents the world. We introduce HYDRA, an architecture for a model-based agent operating in open-world, mixed continuous-discrete environments that detects and effectively adapts to novelties encountered during performance. HYDRA does so by using a compositional model of the environment and employing a range of AI techniques including domain-independent automated planning, anomaly detection, model-based diagnosis, and heuristic search. We also present a design for a HYDRA agent that uses PDDL+, a rich domain-independent planning domain description language, to define its compositional model. An empirical evaluation on the well-known balancing cartpole problem, a standard Reinforcement Learning (RL) benchmark, demonstrates that our PDDL+-based HYDRA agent is able to quickly and effectively detect and adapt to some classes of novelties, outperforming an off-the-shelf deep q-network (DQN) agent.

1 Introduction

One hallmark of human cognition is our ability to function in an open world. People navigate to previously unseen places, perform new tasks, and integrate new technology into their lives. In games, human flexibility supports inventing new strategies along with adapting to changing rules. In contrast, current AI systems perform with superhuman capability in many closed-world game domains, but minor perturbations of the game can lead to significant drops in performance. For example, Witty et al. [2018] demonstrated that even changes which made the game easier could cause catastrophic results for superhuman-performing deep Q-learning agents. This mismatch between human cognitive abilities and machine capabilities indicates that effectively responding to novelty is a major problem in current AI systems [Senator, 2019].

In this work, we explore the novelty response problem in the context of an autonomous agent acting in the world. The agent performs actions, collects observations, and accumulates rewards which it aims to maximize. At some stage, a novelty is introduced, that is, the dynamics of the environment changes. To continue to function effectively, the agent must adapt its behavior. A naive approach for this novelty response problem is to re-train the agent on the modified environment using an off-the-shelf RL algorithm. Yet, such re-training may require numerous interactions with the environment and ignores knowledge about aspects of the environment that has not changed.

The first contribution of this work is HYDRA, a model-based architecture of an autonomous agent that plans and acts effectively in a changing environment. A key element in the HYDRA architecture is the use of a *compositional model* of the environment, i.e., a model whose components represent meaningful aspects of the environment, as opposed to a black-box prediction model such as a deep neural network. This compositional model is used to plan which actions to perform, but also to monitor their execution and detect novelties by comparing the outcomes of executed actions with the model's expectation. When novelty is detected, the HYDRA agent responds by employing a model-based diagnosis engine to identify the components of its compositional model that need to be adapted to be consistent with detected novelties. The main benefit of using a compositional model is that planning is more robust to changes and adapting to novelty does not require re-training the agent from scratch.

Our second contribution is a complete implementation of a HYDRA agent that uses PDDL+ [Fox and Long, 2006] to represent its compositional environment model. PDDL+ is a feature-rich domain-independent planning modeling language that has been used to model a range of complex planning problems, including Chemical Batch Plant [Della Penna *et al.*, 2010a], Atmospheric Reentry [Piotrowski, 2018], Urban traffic Control [Vallati *et al.*, 2016], and Planetary Lander [Della Penna *et al.*, 2010b]. Using PDDL+ allows such a HYDRA agent to be used in a broad range of domains, including domains that require discrete and continuous state variables as well as exogenous events and continuous processes. We describe our PDDL+-based HYDRA agent in detail, and highlight the challenges in doing so. The third contribution of this paper is an empirical evaluation of a PDDL+-based HYDRA agent on a standard Reinforcement Learning (RL) benchmark - OpenAI Gym's balancing Cartpole domain [Barto *et al.*, 1983; Brockman *et al.*, 2016]. Our results show that HYDRA is *resilient* to novelties; i.e, the degrada-

tion in its performance is not as severe as the standard deep Q-network (DQN) RL agent. Second, we show that HYDRA adapts *quickly*, requiring less than 20 episodes to return to top performance after novelty has been introduced, much faster than the DQN RL agent. Finally, adaptations in HYDRA are *interpretable*. They are represented in terms of changes to the elements of its domain model enabling inspection of proposed changes.

2 Problem Definition

HYDRA is designed to interact with the environment in a standard RL setup. That is, the agent plays a sequence of **episodes**, every episode consists of a sequence of **steps**, and every step consists of observing a state, performing an action, observing the resulting state, and receiving reward. The reward of an episode is the sum of rewards in its constituent steps. We assume that the *environment* is a transition system $E = \langle S, A, S_I, S_T, R \rangle$ where S is the possibly infinite set of states; A is the set of possible actions; $S_I \subseteq S$ is a set of possible initial states, $S_T \subseteq S$ is set of terminal states, and R is a reward function $R : S \to \mathbb{R}$, where $R(s)$ returns the reward of reaching state s.[1] An agent plays an episode in the environment E means that the agent selects and performs an action in the environment starting from a randomly selected initial state $s_I \in S_I$ and ending when reaching a terminal state $s_T \in S_T$. The result of an agent playing an episode is a *trajectory*, which comprises a sequence of tuples of the form $\langle s, a, s', r \rangle$, representing that the agent performed action a at state s, reached the state s' and collected a reward r.

Following the *Theory of Environmental Change* [Langley, 2020], we view *novelty* as a transformation of the underlying environment E that occurs at some point in time. Formally, we consider novelty here as a function φ that can be applied to an environment E and outputs an environment $\varphi(E)$ that is different from E in some way.[2] Introducing a novelty φ in an environment E means that the underlying environment E changes to $\varphi(E)$. In an open world, sequences of novelties can, in general, be introduced at multiple points in time. However, in this work we impose constraints and assume that (1) novelty is not introduced during an episode, only between episodes; (2) at most one novelty will be introduced; (3) once a novelty is introduced all subsequent episodes are played in the modified environment. We refer to this setup as the *single persistent novelty setup*.

Definition 1 (The Novelty Response Problem)**.** *A novelty response problem is defined by a tuple $\Pi = \langle E, \varphi, t_N \rangle$ where E is a transition system, φ is a novelty function, and t_N is a non-negative integer specifying the episode in which novelty φ is introduced. In this setup, an agent plays t_N episodes from environment E and all subsequent episodes in environment $\varphi(E)$.*

The objective of an agent faced with this novelty response problem is to maximize its cumulative reward over time. A key challenge for an agent of this setup is that the agent does know the values of neither t_N nor φ. That is, the agent does not known *when* novelty will be introduced and *how* it will change the environment. Next, we describe HYDRA, our proposed architecture for a model-based agent faced with a novelty response problem.

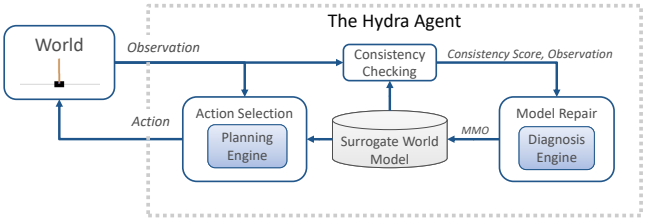

Figure 1: An overview of an HYDRA agent.

3 The HYDRA Agent

HYDRA assumes the availability of a compositional model $\hat{E}$ of the environment E that can make predictions about the expected outcome of the agent's actions. Obtaining a compositional model for a given domain can be done manually by human experts or learned from data by action model learning algorithms such as ARMS [Wu *et al.*, 2007], LOCM [Cresswell *et al.*, 2013], FAMA [Aineto *et al.*, 2019], and SAM Learning [Juba *et al.*, 2021] and others [Asai and Muise, 2020]. Recent work has even demonstrated learning such models from natural language [Feng *et al.*, 2018; Lindsay *et al.*, 2017]. Note that while hydra has access to the compositional model $\hat{E}$, it may not have complete knowledge of the actual environment E, and, more importantly, to the environment after novelty has been introduced ($\varphi(E)$). HYDRA has the following main modules.

- **Action Selection.** The role of this module is to select which action to perform next given the current observed state in the environment.
- **Consistency Checking.** The role of this module is to associate an *inconsistency score* for the current observation, indicating how consistent it is with $\hat{E}$. A high inconsistency score indicates $\hat{E}$ and E are not sufficiently aligned, possibly suggesting that novelty has been introduced.
- **Model Repair.** The role of this module is to modify the relevant components of $\hat{E}$ such that it is consistent with the recent observations.

Figure 1 illustrates how these modules operate together. When the agent collects a new observation from the environment, it associates it with a consistency score using the Consistency Checking module. If the inconsistency score is reasonably low, HYDRA finds a state $\hat{s}$ in $\hat{E}$ that best represents the current state in E and selects the next action to perform using the Action Selection module. This is done by running an automated planner to generate a plan that starts at $\hat{s}$ and is expected to maximize the collected reward according to $\hat{E}$.[3] Based on this plan, the Action Selection module

[1] Other reward function formulations are also common, e.g., where the reward function includes both state and action ($R(s, a)$).

[2] Boult et al. [Boult *et al.*, 2021] referred to this type of novelty as a *world novelty*.

[3] Practically, one may wish to limit the number of planning ses-

```
(:process movement
:parameters ()
:precondition (and (ready)(not (total_failure)))
:effect (and (increase (x) (* #t (x_dot)) )
(increase (theta) (* #t (theta_dot)))
(increase (x_dot) (* #t (x_ddot)) )
(increase (theta_dot) (* #t (theta_ddot)) )
(increase (elapsed_time) (* #t 1) ) ))
```

Figure 2: (CartPole) Continuous PDDL+ process updating over time the positions and velocities of the cart and pole.

outputs the next action to perform. This process continues until an episode is completed. At this stage, the Consistency Checking module analyzes the executed trajectory to see if it is consistent with $\hat{E}$. This means analyzing every observed transition, i.e., every $\langle \hat{s}, a, \hat{s}', r \rangle$ tuple to see if all states and transitions are consistent with $\hat{E}$. If this is the case, no change to $\hat{E}$ is needed and HYDRA will start a new episode. Otherwise, the Consistency Checking module outputs a real value γ quantifying the likelihood that this consistency indeed indicates that novelty has been introduced. If this value passes some pre-defined threshold, HYDRA declares that novelty has been detected. This inconsistency value is then passed to the Model Repair module along with the observed trajectory. The Model Repair module attempts to modify $\hat{E}$ such that it will be consistent with the observed trajectory. This is done by applying a model-based diagnosis engine, e.g., GDE [de Kleer and Williams, 1987], to identify the model components that explain the observed inconsistency. To this end, HYDRA requires a set *Model Manipulation Operators* (MMOs), denoted $\mathcal{M}$. Each MMO represents a possible change to $\hat{E}$. The Model Repair searches for a sequence of MMOs such that, when applied to $\hat{E}$, yields a model that is most consistent with the observed trajectory. Then, the selected MMO is applied to $\hat{E}$, and the process repeats for the next episode with the repaired meta model.

4 Implementing HYDRA Agents using PDDL+

Next, we describe an implementation of a HYDRA agent using PDDL+ [Fox and Long, 2006] as our modeling language for $\hat{E}$. PDDL+ is an extension of the well-known Planning Domain Definition Language (PDDL) [McDermott *et al.*, 1998] that allows modeling of temporal hybrid systems as planning domains that exhibit mixed discrete and continuous system dynamics. In PDDL+, actions are defined by specifying their preconditions and effects in PDDL. In addition, PDDL+ supports modeling exogenous behavior with discrete events and continuous processes. Events apply discrete effects instantaneously, whereas processes apply changes over time while their preconditions hold. The agent has no direct control over processes and events, it can only interact with exogenous activity indirectly. Figure 2 demonstrate a process defined in PDDL+ for the movement of a cartpole. We chose PDDL+ since it generalizes many previously proposed planning languages, which highlights the generality of our approach and simplifies supporting additional domains. Also, there exists off-the-shelf PDDL+ planners, such as UPMurphi [Della Penna *et al.*, 2009], which can be used to obtain plans.

sions to save computational efforts.

4.1 Consistency Checking

In simple terms, consistency is a measure of how accurately the compositional model of the environment describes the actual environment and its dynamics. Given a PDDL+ model Π and plan π, one can simulate the trajectory we *expect* to observe when executing π. Our implementation of the Consistency Checking module computes the consistency score of an observed trajectory by measuring how different it is from the expected trajectory. In our implementation, we measured difference by computing the maximal Euclidean distance between matching states along these trajectories (i.e., tracking differences between the *expected* and *observed* positions and velocities of the cart and pole over the course of an episode). However, other ways to measure distances between trajectories can also be applied. A novelty is detected when the discrepancy between the observed and expected sequence of states exceeds a predefined threshold.

This approach to detecting novelties is domain-independent. However, it relies on the accuracy of the PDDL+ model. In our implementation, this was sufficient for the Cartpole domain. However, for other domains the available PDDL+ model may not be accurate enough to exactly match the observed state trajectory. To address this, one may augment the PDDL+-based Consistency Checking approach by creating a domain-specific consistency checking module that incorporates qualitative analysis of the observed and expected trajectories. Alternatively, one may employ representation learning and train a neural network over non-novelty episodes to do next state prediction.

4.2 Meta Model Repair

The key to implementing a PDDL+-based HYDRA agent is to define appropriate MMOs. For a PDDL+-based HYDRA agent, an MMO is a function that modifies some element of the PDDL+ model, e.g., change an action, event, or process, or even add a completely new event or process. In our implementation, we limited the MMOs we consider to only modify the different constants used in the PDDL+ model, e.g., changing the constant corresponding to gravity by some fixed amount (either positive or negative).

To check if a sequence of MMOs should be used, we apply a model-based diagnosis approach, that is, we apply them to the current PDDL+ model, simulate the expected sequence of states according the modified model, and check if this sequence of states is consistent with the observed trajectory using the Consistency Checking module. The space of possible MMO sequences is combinatorial. To focus the search, a model-based diagnosis engine can be used. We implemented To search this space, we implemented a greedy best-first search that uses a heuristic a linear combination of the consistency score returned by the consistency checker and the size of the evaluated MMO sequence. The latter consideration biases the search towards simpler repairs.

5 Experimental Results

Next, we demonstrate the ability of our PDDL+-based HYDRA agent to detect and respond to novelty effectively by performing experiments on the well-known balancing Cartpole domain. We also report on a small-scale comparison with an off-the-shelf RL agent, showing that HYDRA can respond effectively to some novelties using significantly fewer interactions with the environment.

5.1 The Cartpole Domain and Novelties

The Cartpole domain is a classical RL benchmark in which a pole is connected to a cart and the task is to balance the pole in the upright position by pushing the cart either left or right. There are multiple variants to the Cartpole domain. In our experiment, we used the standard Cartpole implementation from OpenAI Gym (https://gym.openai.com). This Cartpole domain is defined by several parameters including mass of the cart, length of the pole, gravity, and friction coefficient. A state in this domain is represented by 4 state variables (velocities and positions of the cart and the pole). In every step, the agent's action is to apply force to the cart in either left or right direction. An episode ends when either 200 steps have been performed or a limit is exceeded (i.e. pole angle or cart position). Every step returns +1 reward, so the total reward for an episode is between 1 and 200.

We ran experiments where the introduced novelty involves changes the values of the pole's gravity and the mass of the cart. We experimented with a range of changes to these values. The results reported are for two such changes: (1) *Type 1*: gravity increases from 9.8 to 12 and the pole length grows from 1.0 to 1.1; (2) *Type 2*: cart mass decreases from 1.0 to 0.9 and the pole length grows from 1.0 to 1.1. We selected to report on these types of novelties as introducing them resulted in a significantly decrease in performance for all agents but the cartpole in the environment after introducing is still controllable. The selected novelty type was introduced in the 8^{th} episode in a trial. Every trial consisted of 30 episodes, and each experiment consisted of 5 trials.

5.2 Baselines and Experiment Setup

In addition to HYDRA, we report on the performance of two baseline agents: a non-adaptive planning agent and a deep Q-network (DQN) reinforcement learning agent. We refer to these agents as the planning agent and DQN agent, respectively. The planning agent uses the same PDDL+ modeling and planner as HYDRA, but does not attempt any model repair. The DQN agent uses a standard deep q-network implementation with experience replay memory [Mnih *et al.*, 2013]. It is built with an input layer (4×16), a hidden layer (16×16), and an output layer (16×2) and uses the Rectified Linear Unit (ReLU) activation function. Both baseline agents were designed and trained to achieve perfect performance in the non-novelty case prior to the experiment.

The experiments we performed involved running each agent through a series of trials. Each trial starts by running the agent through a sequence of episodes in the non-novel environment. Then, we introduce one of the novelty types described above to the environment, and continue to run the agent on episodes from the now novel environment. The performance of the agent on the non-novel episodes reflects its performance on the environment it has been designed or trained for, while the performance on the novel episodes measure its ability to respond to the introduced novelty.

We conducted two types of experiments: (1) *system detection* experiments, in which the presence of novelty was not indicated to the agent and it is expected to infer the presence of novelty and react to it autonomously, and (2) *given detection* experiments, in which an oracle indicated the presence of novelty to the agent. Given detection experiments allow us to measure separately the agent's ability to respond to novelties, while system detection experiments better simulate how agents act in the real-world, where novelty detection oracles are not available. The planning agent does not attempt to learn or adapt to novelty, so its behavior for both experiment types (system detection and given detection) is the same. The DQN agent does not explicitly attempt to detect novelty, thus its behavior in the system detection experiments is to continue to run with its trained Q-network. For the given detection experiments, we allow the DQN agent to reset its learning factor and make Bellman updates to its trained Q-network when novelty is indicated.

5.3 Results

The results are summarized in Figure 3. The x-axis shows the episodes in a trial and the y-axis shows the total reward collected by the agent per episode, represented as the proportion of its score with a perfect controller. The red line indicates the episode where the selected novelty was introduced. The blue line shows the results for the "system detection" experiments and the orange line shows the results for the "given detection" experiments. The shaded areas for each represent the 95% confidence interval computed over 5 trials.

As shown in Figure 3, all agents achieve perfect performance at the beginning of the trial and then experience a significant drop in performance as soon as the novelty is introduced (episode 8). This shows the selected novelties are meaningful for all agents. The are three key observations to be made.

Resilience: The performance drop observed when novelty is introduced is significantly more drastic for the DQN agent than for the planning agent and HYDRA. For both types of novelties, the performance of the DQN agent drops to approximately 25% of its pre-novelty performance and below, while the performance of the planning agent and HYDRA drop by only 50%. This difference can be explained by the agents' design. Both the planning agent and HYDRA are built with *composable* component models. Even in novel situations, a majority of the component models are still relevant and can be exploited to drive behavior. In contrast, the knowledge that drives behavior in end-to-end learning systems like the deep Q-network is distributed and cannot be re-purposed to drive behavior in an environment with changing dynamics. This observation demonstrates that agents based on a composable component models can be reselient to novelties.

Quick adaptation: Since the planning agent does not react to novelties directly, its performance does not improve over time in both experimental conditions (given detection and system detection). The DQN agent does attempt to adapt to

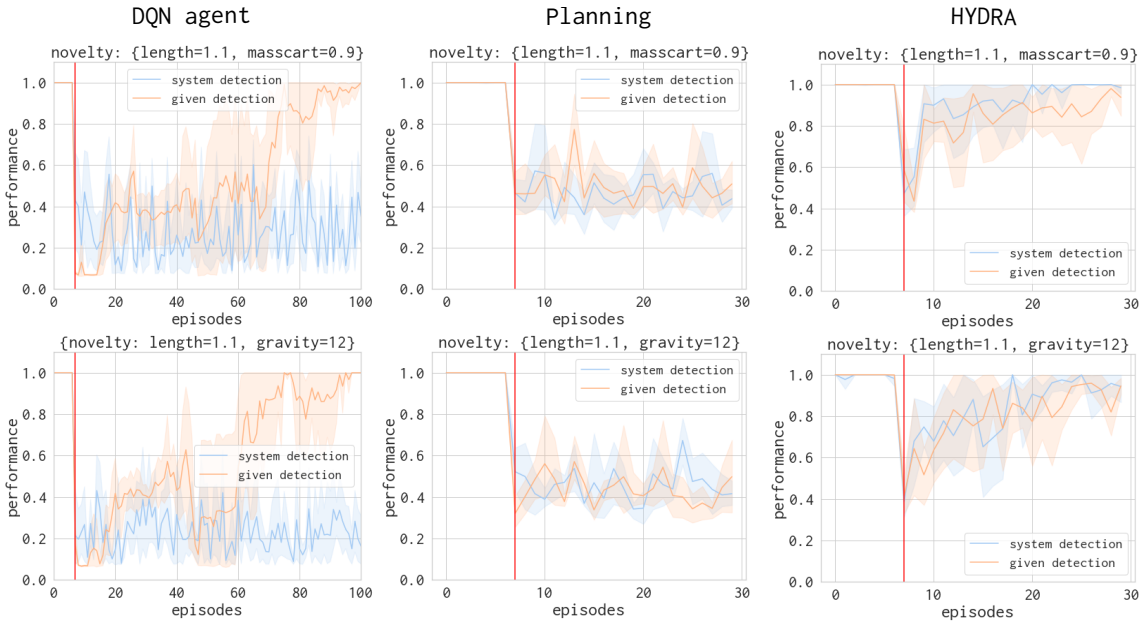

Figure 3: Graphs showing performance of DQN agent, planning agent, and HYDRA in novelty experiments. Episodes in a trial are on the x-axis and reward earned is shown on the y-axis. The results are averaged over 5 trials. Red line indicates the episode where novelty has been introduced. In the top row, the novelty introduced is increasing the pole length to 1.1 and decreasing the cart mass to 0.9 (novelty type 1). In the bottom row, the novelty introduced is the same increase in pole length and setting gravity to be 12. The results clearly show that HYDRA adapts very quickly to the novelties in our experiment, returning to optimal performance in 10 episodes.

```
1. {"m_cart": 0, "l_pole": 0.2, "m_pole": 0.1, "force_mag": 0, "gravity":
0, "angle_limit": 0, "x_limit": 0}
2. {"m_cart": 0, "l_pole": 0.30000000000000004, "m_pole": 0, "force_mag":
0, "gravity": 0, "angle_limit": 0, "x_limit": 0}
```

Figure 4: Example adaptations produced by HYDRA represented as changes to various elements in the model (presented as delta from the default value of each variable, 0 denotes no change).

the observed novelty. While it starts with worse performance in the given detection condition, it improves towards the later parts of the trial. This observation suggests that the agent is learning a new Q-function that supports the new dynamics of the environment under novelty conditions. However, this learning is slow, requiring multiple interactions with the environment to return to optimal performance ($\sim$ 75 episodes). In contrast, HYDRA adapts very quickly in less than 20 episodes for all the types of novelty and experimental conditions we considered. This observation supports our central thesis: HYDRA's model-based design enable fast novelty response via localized novelty detection and adaptation.

Interpretability: HYDRA adapts to presented novelties by proposing how various elements of its model of the environment can be changed and consequently, its learning can be inspected an analyzed by a human. Figure 4 shows examples of adaptation in hydra in response to the posed novelty of Type 2.

6 Related Work

The topic of how to detect and react to novelties has been gaining significant attention in the AI literature. The novelty problems we address in this work has been described by Senator [Senator, 2019] and analyzed by Boult et al. [Boult *et al.*, 2021] and Langely [Langley, 2020]. Boult et al. [Boult *et al.*, 2021; Langley, 2020] did not propose an agent design for this setting, but rather suggested a framework for characterizing different types of novelties.

Langely also identified four elements an agent architecture needs to properly address novelty detection and response — "performance, monitoring, diagnosis, and repair." HYDRA implements these elements. It uses its meta model to generate plans, act (*performance*), and detect novelties (*monitor*). Then, it uses heuristic search to identify elements of the meta model that are incorrect (*diagnosis*) and modifies the meta model accordingly (*repair*).

We are not the first to design a novelty-robust agent and novelty detection and response algorithms. DiscoverHistory [Molineaux *et al.*, 2012] is an algorithm for inferring the possible values of unobservable state variables from observations. While our current experiments are similar in nature, HYDRA is designed to be more general, including novelties such as adding new variables, process, and object types. MIDCA [Paisner *et al.*, 2014] is a cognitive architecture for designing novelty-robust agents, where novelty is limited to which goal to pursue next. The recently proposed OpenMIND system [Musliner *et al.*, 2021] addresses as similar problem setup to ours, but on a different set of domains and using a less expressive planning formalism.

More recently, there have been attempts at defining more general frameworks for handling novelty in open-world environments. Muhammad et al. [2021] proposed an approach for a cognitive agent to detect, characterize, and accommodate

novelties based on knowledge and inference from the agent's internal predictions and the observed ground truth. While it also exploits a planning-centric architecture and defines the composition of the world in a planning paradigm, there exist differences, the key of which is the richness of the environments that our approaches operate in and reason with. Most notably, they do not consider scenarios with external activity (i.e. beyond the executive of the agent), or continuous change in the environment.

Other recent developments in the area have tackled yet another class of application domains, such as the game of Monopoly [Gopalakrishnan *et al.*, 2021], a multi-player game heavily skewed towards uncertainty (from dice rolls, card draws, or adversaries' actions). The agent behaves according to a policy with a state-value function based on a short-horizon lookahead approach, prioritizing robustness to novelties in an already unpredictable game.

7 Discussion

Novelty types	Domain adjustment	Novelty examples
Spatio-temporal	Fluent changes	Increased gravity
Structures	New objects and fluents	Ball added to cart
Processes	New and/or changing existing processes	Introduced wind
Constraints	New preconditions and/or changed events	Limit direction change frequency

Table 1: Description of example novelties that can be encountered in cartpole, changes to the PDDL+ model required to accommodate them, and their corresponding novelty types defined by [Langley, 2020].

Most modules in the HYDRA agent are domain-independent and we are currently implementing it on two other domains that can be expressed using PDDL+. To implement HYDRA in a new domain that can be represented by PDDL+, one needs to only (1) create the initial PDDL+ compositional model, and (2) select the appropriate MMOs for the Model Repair module. In our current implementation, we focused in MMOs that modify the value of constants in the PDDL+ model such as gravity, pole length, and pole mass. An open research challenge is how to identify the necessary and sufficient set of MMOs for a given domain and types of novelties. Table 1 maps possible types of MMOs to types of novelties as defined by Langley [Langley, 2020] along with examples from the cartpole domain. The number of MMOs may be very large and thus finding a sequence of MMOs that may yield a consistent model is a challenging combinatorial search problem. We expect to need heuristics to guide the search in an efficient manner. In our current implementation, we run a Greedy Best-First Search algorithm that uses a heuristic that prefers shorter MMO sequences that yield models that are more consistent. Future work may explore more sophisticated search techniques for this task.

The requirement to define MMOs a-priori also raises the question: how will HYDRA work on novelties that cannot be property characterized with the selected set of MMOs. Indeed, HYDRA will fail if no sequence of MMOs can reach a compositional model that sufficiently captures the novel aspects of the environment. However, note that even if the MMOs do not allow exactly defining the modifications to the environment, they might still allow finding a compositional environment model that is close enough to allow reasonable action selection. For example, consider the case where the novelty introduced is an increase in air resistance, but the available HYDRA agent only has an MMO that modify the amount of force applied to the cart when pushed by the agent. The Model Repair module of HYDRA might then apply that MMO and incorrectly repair its model to assume the force it applies is weaker than it really. While the new compositional model is different from the actual changed environment, it would allow effective acting for slow cart speeds. Nevertheless, such an agent will under-perform when the cart reaches higher velocities. More broadly, one may ask whether the novelty response problem is not amenable to a theory equivalent to the "no free lunch" theorem, where any novelty response solution will work on some novelties but fail on others. This is another topic for future work.

8 Conclusions and Future Work

We presented an architecture for a model-based agent called HYDRA, which can plan and act in an environment, as well as detect and adapt to novel changes in the environment as they occur. The key idea in the HYDRA architecture is the use of compositional model of the environment. HYDRA uses this model to (1) generate plans, using an automated planner; (2) detect novelties, by matching observations with model predictions; and (3) adapt to novelties, by applying model-basd diagnosis to identify the model components that need to be repaired. . We described a complete implementation of HYDRA using the PDDL+ to describe its compositional model of the environment. Finally, we empirically evaluated our PDDL+-based HYDRA agent in the Cartpole domain, a well-known DQN benchmark. Our results demonstrate several types of novelty that our PDDL+-based agent can detect and adapt to very rapidly, requiring at most 20 episodes before returning to normal performance, which is significantly faster than an off-the-shelf DQN for this domain.

References

[Aineto *et al.*, 2019] Diego Aineto, Sergio Celorrio, and Eva Onaindia. Learning action models with minimal observability. *Artificial Intelligence*, 275:104–137, 05 2019.

[Asai and Muise, 2020] Masataro Asai and Christian Muise. Learning neural-symbolic descriptive planning models via cube-space priors: The voyage home (to STRIPS). In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2676–2682, 2020.

[Barto *et al.*, 1983] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[Boult *et al.*, 2021] TE Boult, PA Grabowicz, DS Prijatelj, Roni Stern, Lawrence Holder, Joshua Alspector, M Jafarzadeh, Toqueer Ahmad, AR Dhamija, Chunchun Li,

et al. Towards a unifying framework for formal theories of novelty. In *AAAI Conference on Artificial Intelligence*, pages 15047–15052, 2021.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Cresswell *et al.*, 2013] Stephen N Cresswell, Thomas L McCluskey, and Margaret M West. Acquiring planning domain models using locm. *The Knowledge Engineering Review*, 28(2):195–213, 2013.

[de Kleer and Williams, 1987] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[Della Penna *et al.*, 2009] Giuseppe Della Penna, Daniele Magazzeni, Fabio Mercorio, and Benedetto Intrigila. Upmurphi: a tool for universal planning on pddl+ problems. In *Nineteenth International Conference on Automated Planning and Scheduling*, 2009.

[Della Penna *et al.*, 2010a] Giuseppe Della Penna, Benedetto Intrigila, Daniele Magazzeni, and Fabio Mercorio. A pddl+ benchmark problem: The batch chemical plant. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2010.

[Della Penna *et al.*, 2010b] Giuseppe Della Penna, Benedetto Intrigila, Daniele Magazzeni, and Fabio Mercorio. Resource-optimal planning for an autonomous planetary vehicle. *International Journal of Artificial Intelligence & Applications (IJAIA)*, 1(3):15–29, 2010.

[Feng *et al.*, 2018] Wenfeng Feng, Hankz Hankui Zhuo, and Subbarao Kambhampati. Extracting action sequences from texts based on deep reinforcement learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4064–4070, 2018.

[Fox and Long, 2006] Maria Fox and Derek Long. Modelling mixed discrete-continuous domains for planning. *JAIR*, 27:235–297, 2006.

[Gopalakrishnan *et al.*, 2021] Sriram Gopalakrishnan, Utkarsh Soni, Tung Thai, Panagiotis Lymperopoulos, Matthias Scheutz, and Subbarao Kambhampati. Integrating planning, execution and monitoring in the presence of open world novelties: Case study of an open world monopoly solver. *arXiv preprint arXiv:2107.04303*, 2021.

[Juba *et al.*, 2021] Brendan Juba, Hai S. Le, and Roni Stern. Safe learning of lifted action models. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 379–389, 2021.

[Langley, 2020] Pat Langley. Open-World Learning for Radically Autonomous Agents. *AAAI*, 2020.

[Lindsay *et al.*, 2017] Alan Lindsay, Jonathon Read, Joao F Ferreira, Thomas Hayton, Julie Porteous, and Peter Gregory. Framer: Planning models from natural language action descriptions. In *International Conference on Automated Planning and Scheduling*, 2017.

[McDermott *et al.*, 1998] Drew McDermott, Malik Ghallab, Craig Knoblock, Adele Howe, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl — the planning domain definition language. Technical report, AIPS-98 Planning Competition Committee, 1998.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[Molineaux *et al.*, 2012] Matthew Molineaux, Ugur Kuter, and Matthew Klenk. Discoverhistory: Understanding the past in planning and execution. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 989–996, 2012.

[Muhammad *et al.*, 2021] Faizan Muhammad, Vasanth Sarathy, Gyan Tatiya, Shivam Goel, Saurav Gyawali, Mateo Guaman, Jivko Sinapov, and Matthias Scheutz. A novelty-centric agent architecture for changing worlds. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 925–933, 2021.

[Musliner *et al.*, 2021] David J. Musliner, Michael J. S. Pelican, Matthew McLure, Steven Johnston, Richard G. Freedman, and Corey Knutson. Openmind: Planning and adapting in domains with novelty. In *Annual Meeting of the Cognitive Science Society*, 2021.

[Paisner *et al.*, 2014] Matthew Paisner, Michael Cox, Michael Maynord, and Don Perlis. Goal-driven autonomy for cognitive systems. In *Annual Meeting of the Cognitive Science Society*, 2014.

[Piotrowski, 2018] Wiktor Mateusz Piotrowski. *Heuristics for AI Planning in Hybrid Systems*. PhD thesis, King's College London, 2018.

[Senator, 2019] Ted Senator. Science of ai and learning for openworld novelty (sail-on). Technical report, DARPA, 2019.

[Vallati *et al.*, 2016] Mauro Vallati, Daniele Magazzeni, Bart De Schutter, Lukás Chrpa, and Thomas Leo McCluskey. Efficient macroscopic urban traffic models for reducing congestion: A pddl+ planning approach. In *AAAI Conference on Artificial Intelligence*, 2016.

[Witty *et al.*, 2018] Sam Witty, Jun Ki Lee, Emma Tosch, Akanksha Atrey, Michael Littman, and David Jensen. Measuring and characterizing generalization in deep reinforcement learning. *arXiv preprint arXiv:1812.02868*, 2018.

[Wu *et al.*, 2007] Kangheng Wu, Qiang Yang, and Yunfei Jiang. ARMS: An automatic knowledge engineering tool for learning action models for ai planning. *The Knowledge Engineering Review*, 22(2):135–152, 2007.