Model-Based Novelty Adaptation for Open-World AI

Submission #440

ABSTRACT

Model-based agents, such as those usually presented in the automated planning literature, are often more reliable, explainable, and effective than agents created by model-free reinforcement learning techniques. However, to function in an open world, an automated agent must be able to adapt to novel situations. Most model-based agents are ill-equipped to handle novel situations in which their model of the environment no longer sufficiently represent the world. We introduce HYDRA, a model-based agent operating in open-world environments, that detects such novelties and adapts to them automatically. HYDRA uses a mixed continuous-discrete planning formalism, namely PDDL+, which allows modeling a broad range of domains. It detects novelties by analyzing inconsistencies between its PDDL+ model and observations, and quickly adapts to novelty by automatically repairing its model. The latter step uses model-based diagnosis to identify possible repairs – diagnoses – that are consistent with the collected observations. We demonstrate our approach on two domains: the well-known balancing cartpole problem and the popular Angry Birds game. Our results show that we are able to quickly and effectively detect and adapt to some classes of novelties.

KEYWORDS

Open world learning, Automated planning, Novelty adaptation

ACM Reference Format:
Submission #440. 2022. Model-Based Novelty Adaptation for Open-World AI. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Auckland, New Zealand, May 9–13, 2022*, IFAAMAS, 7 pages.

1 INTRODUCTION

While current AI systems perform with superhuman capability in many game domains, each of these domains is a closed world, and minor perturbations of the game can lead to significant drops in performance. Witty et al. demonstrated that even changes which made the game easier could cause catastrophic results for superhuman performing deep Q-learning agents [19]. On the other hand, 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 (e.g., consider chess players who play bughouse[1]). This mismatch between human cognitive abilities and machine capabilities indicates that adapting to novelty is a major problem in current AI systems. Our research goal is to impart machines with human-like learning capabilities to make them robust to novelty.

[1]https://en.wikipedia.org/wiki/Bughouse_chess

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Auckland, New Zealand.

As a step towards achieving this goal, we developed the **Hypothesis-Guided Model Revision over Multiple Aligned Representations** (HYDRA) agent. HYDRA is an autonomous agent that operates in a mixed continuous-discrete environment and can detect and adapt to novel changes in that environment. HYDRA is designed to performs two key tasks: **novelty detection** and **novelty adaptation**. Novelty detection is the problem of identifying from the collected observations whether something meaningfully novel has occurred [13]. Novelty response is the problem of adapting the behavior of the agent when novelty is introduced such that it continues to function effectively. While each of these tasks, individually, have been addressed in the literature to some extent, our agent design solves them in a synergetic manner and in the context of acting in a dynamically changing world.

[[Roni: maybe add a bit more here in terms of related works]]

Central to our approach for both novelty detection and adaptation is the use of PDDL+ [5], a feature-rich domain-independent planning modeling language, to represent HYDRA's beliefs about the environment dynamics. PDDL+ is an extension of the well-known Planning Domain Definition Language (PDDL) [8] that allows defining discrete and continuous state variables as well as exogenous events and continuous processes. It has been used to model a range of complex planning problems, including Chemical Batch Plant [3], Atmospheric Reentry [14], Urban traffic Control [18], and Planetary Lander [4]. HYDRA selects which actions to perform by invoking a domain-independent PDDL+ planner over the agent's PDDL+ model of the environment and the task at hand. Novelty detection is performed by comparing the expected outcome of the actions it performed with the observed outcome. Novelty response is performed by automatically diagnosing and then repairing HYDRA's PDDL+ model so that it is consistent with the observations.

We have implemented HYDRA on two domains: OpenAI Gym's balancing Cartpole domain [2], and Science Birds, a version of the popular Angry Birds game developed for research purposes [15]. In this work, we describe our implementation for Cartpole and report on a small-scale set of experiments in the Cartpole domain. The results show that HYDRA can adapt to novelty much faster than standard Reinforcement Learning (RL) techniques, requiring only

[[TODO: HOW MANY]]

interactions with the world to return to top performance after novelty has been introduced. Then, we describe key challenges that arise when using Hydra for ScienceBirds, and outline directions to meet these challenges.

2 PROBLEM DEFINITION

HYDRA is designed to interact with the environment in a standard Reinforcement Learning (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* can be modeled as a transition system $E = \langle S, A, Tr, I, T, R \rangle$ where S is the possibly infinite set of states; A is the set of possible actions; Tr is a transition function $Tr : S \times A \times S \rightarrow [0, 1]$, where $Tr(s, a, s')$ returns the probability of reaching a state s' when performing action a in state s; $S_I \subset S$ is a set of possible initial states, $S_T \subset S$ is set of terminal states, and R is a reward function $R : S \rightarrow \mathbb{R}$, where $R(s)$ returns the reward of reaching state s.[2] The agent has a possibly partial view of the environment E and the states it encounters, but this view includes the rewards it collects. An agent plays an episode in an environment $E = \langle S, A, Tr, S_I, S_T, R \rangle$ 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* [7], we view novelty as a *transformation* of the underlying environment E that occurs 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.[3] 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 simplify 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 2.1 (Single persistent novelty setup). A single persistent novelty setup is defined by a tuple $\Pi = \langle E, \varphi, i \rangle$ where E is a transition system, φ is a novelty function, and i is a non-negative integer specifying the episode in which novelty φ is introduced. In this setup, an agent plays i episodes from environment E and all subsequent episodes in environment $\varphi(E)$.

A key aspect of this setup is that the agent does know the value of i or φ, i.e., the agent is not given *when* novelty will be introduced and *how* it will change the environment. We focus on two challenges that arise in this setup when an agent plays multiple episodes: *novelty detection* and *novelty response*. The former challenge refers to identifying which episodes are played in $\varphi(E)$ and the latter challenge refers to maximizing the rewards collected over time. Note that to directly address the novelty detection challenge, the game playing agent outputs its belief about whether or not novelty has been introduced after playing every episode.

For novelty detection, performance can be measured using standard anomaly detection metrics (false positives, false negatives, etc.). For novelty response, performance can be measured by the cumulative reward collected by the agent after novelty has been introduced.

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

[3]Boult et al. [1] referred to this type of novelty as a *world novelty*.

2.1 Domains

The single persistent novelty setup and the novelty detection and response challenges can be studied on virtually any domain. In addition, the design of the agent is domain-independent, as well as most of its algorithmic components. However, we use the following two domains as running examples as well as a test bed for evaluating HYDRA. For completeness, we provide a brief description of these domains.

Cartpole. A classical RL benchmark domain 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. The variant we are using is defined as follows. 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. An example of a novelty in this domain can be as simple as a change in the mass of the pole or as difficult as adding wind or even an extra pole that interferes with the original pole.

Science Birds. This is a free version of the popular Angry Birds game developed for research purposes [15]. In this single-player game, the player launches a finite set of birds in sequence at a 2D structure made out of different material blocks and occupied by one or more pigs. The goal is to destroy all the pigs. Different birds and structures have different actions (e.g., yellow birds accelerate when the player taps the screen during flight) and properties (e.g., TNT objects explode when damaged by birds or other falling objects). Science Birds is a challenging domain for AI agents due to the continuous action space and the large state space of resulting block configurations, and has maintained a yearly competition since 2012 [15].

The agent interacts with the game through a server with the following API. After the level is loaded, the agent is given a list of objects with their outer hull polygons and a color-map that specifies the amount of each color inside the polygon[4]. The agent specifies shots by providing an (X, Y) position to launch from and a time t to tap the screen. The screen tap initiates actions based on bird type (e.g., a black bird will explode when tapped, a yellow bird will speed up, etc.). Thus, a state in this domain is the list of objects specified above and the possible actions are "shoot at angle α" and "tap". Note that after shooting a bird the shoot action is not available until the target bird has become inactive and the level elements have settled. Similarly, after tapping a bird the tap action is not available until the next bird is released. In some time steps, there are no available actions, in which case a dummy "no-op" action is assumed. Shots that result in killing pigs or breaking blocks increase the score, which constitutes our reward function. A step constitutes shooting and potentially tapping a single bird. An episode here constitutes playing a single level. Playing a level starts with a finite set of birds that can be shot, and ends when either all pigs have been killed or

[4]Raw pixels for the entire image are also available, but we do not use them in our system.

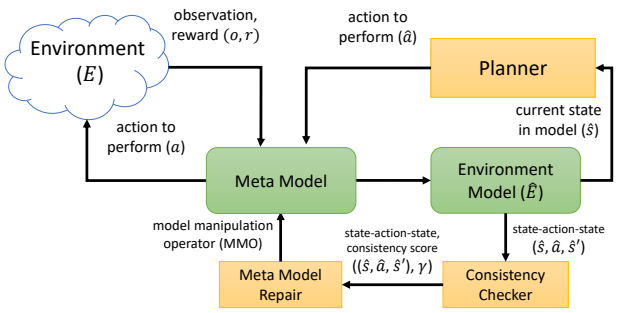

Figure 1: An overview of the HYDRA agent.

when all the given birds were shot and have exploded. Examples of novelties in this domain are an increase in the force of gravity or an addition of a new type of bird with unique powers.

3 THE HYDRA AGENT

To address the novelty detection and response challenges in the persistent single novelty setup, we propose the Hydra agent and describe it here in detail. Hydra is a model-based agent, in the sense that it uses a surrogate model of the environment that allows it to make *state predictions*, i.e., predict the expected outcome of performing actions in the environment E. This surrogate model of the environment, denoted $\hat{E} = \langle \hat{S}, \hat{A}, \hat{Tr}, \hat{I}, \hat{T}, \hat{R} \rangle$, is used by the agent to plan its actions, detect novelty, and adapt to it.

At the core of the Hydra agent is a *meta model*, which serves as a bridge between the actual environment E and the agent's internal representation of it, $\hat{E}$. This meta model has two core functionalities. The first is to track that state of the world by monitoring the observations accepted from E and using them to create or update the surrogate model $\hat{E}$ and identify a state in $\hat{S}$ that best represents the current state of the real environment E. The second task of the meta model is to translate an action of the surrogate model $\hat{a} \in \hat{A}$ to an appropriate action $a \in A$ in the real environment E.

[[Klenk: I think a gray box around all of the colored boxes here representing the HYDRA agent would be helpful. The meta model does not perform actions.]]

Figure 1 illustrates Hydra's design. HYDRA has three algorithmic components:

- **Planner.** The planner accepts a state $\hat{s} \in \hat{S}$ in the surrogate model and generates a plan that, according to $\hat{E}$, will maximize the expected reward. Then, it outputs the action $\hat{a} \in \hat{A}$ to perform now according to this plan.
- **Consistency checker (CC).** The consistency checker uses $\hat{E}$ to predict the expected states according to the previously performed actions and compares them with the observed states.
- **Meta model repair (MMR).** The model repair accepts the same input as the consistency checker as well as the output of the consistency checker. It suggests which changes should be made to the meta model so that it yields an updated surrogate model $\hat{E}$ that better predicts the observations.

These components operate together in the following way. When a new episode starts, Hydra observes the current state and uses the meta-model to translates this observation o to a state $\hat{s}$ in its environment model $\hat{E}$. Then, a planner generates a plan, aiming to maximize the expected reward collected starting from state $\hat{s}$ and operating in $\hat{E}$. Based on this plan, the planner outputs the next action to perform $\hat{a}$, which the meta model then translates to an action a that can be applied in the real environment. This process continues until an episode is completed. At this stage, the HYDRA CC analyzes the executed trajectory to see if it is consistent with the current surrogate environment model. This means analyzing every observed transition, i.e., every $\langle \hat{s}, \hat{a}, \hat{s}' \rangle$ triple, where $\hat{s}$ is an observed state, $\hat{a}$ is the action HYDRA performed in $\hat{s}$, and $\hat{s}'$ is the state observed immediately afterwards. If all states and transitions are consistent with the states and transitions specified in $\hat{E}$, then no change to the meta model is needed and HYDRA will start a new episode.

Otherwise, the CC 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 MMR component along with the observed trajectory. The MMR attempts to modify the current meta model so that it generates a surrogate environment model that is consistent with the observed trajectory. This is done by defining a set of *Meta Model Manipulation Operators* (MMOs), denoted $\mathcal{M}$. Each MMO represents a possible change to the meta model. The MMR searches for a sequence of MMOs such that, when applied to the current meta model, yields a meta model that is most consistent with the observed trajectory (i.e., a meta model that generates a surrogate model consistent with the observed trajectory). Then, the selected MMO is applied to the meta model, and the process repeats for the next episode with the repaired meta model.

4 IMPLEMENTING HYDRA AGENTS USING PDDL+

Most of Hydra's design is domain independent. To demonstrate this, we implemented HYDRA on two very different benchmark domains — Cartpole and Science Birds— and highlight in this section the details of this implementation. In particular, we highlight which parts of our implementation are domain-independent and which parts are design choices that are domain specific.

The first design choice is the modeling language used to define the surrogate environment model $\hat{E}$. Both domains require dealing with mixed discrete and continuous dynamics. In addition, Science Birds dynamics are governed by chains of reactions triggered by the agent's actions, unlike many other planning problems where most world changes are the direct result of agent actions. Due to these properties, we chose PDDL+ [5] as our modeling language for $\hat{E}$. The defining characteristic of PDDL+ is the ability to model exogenous behaviour with discrete events and continuous processes. Events apply discrete effects instantaneously, whereas processes apply changes continuously while their preconditions hold. The agent has no direct control over processes and events, and can only interact with exogenous activity indirectly. As noted above, Science Birds is overwhelmingly governed by processes and events,

```
(: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.

```
(:action pa-twang
   :parameters (?b - bird)
   :precondition (and
       (= (active_bird) (bird_id ?b))
       (not (angle_adjusted))
       (not (bird_released ?b)) )
   :effect (and
       (assign (vy_bird ?b)
           (* (v_bird ?b) SIN(angle)))
       (assign (vx_bird ?b)
           (* (v_bird ?b) COS(angle)))
       (bird_released ?b)
       (angle_adjusted)) )
```

Figure 3: (Science Birds) Action releasing the bird from the slingshot, assigning initial velocities of the launched bird (approximations of sine and cosine functions are obscured).

making PDDL+ an appropriate modeling language for our setting. In addition, PDDL+ generalizes many previously proposed planning languages, which simplifies supporting additional domains.

4.1 Planner

Following the choice of PDDL+ as a modeling language, Hydra requires a meta model that outputs a PDDL+ problem and domain file. To this end, we created two such meta models, one for each domain. The physics of Cartpole are well-known, and thus creating a meta model for such domain is straightforward. Science Birds is a more challenging domain, with a complex dynamics and a large number of objects of different types. There is no known PDDL+ model for Science Birds and fully modeling the game is beyond the scope of this research. Nevertheless, we have created a PDDL+ model that solves a variety of Science Birds levels. Planning in this domain can result in search-space explosion, due to the tight integration of planning and scheduling over a continuous timeline. To improve the performance, our Science Birds model relies heavily on the *Theory of Waiting* [9] and currently employs only one action responsible for the release of the bird from the slingshot at maximum initial velocity (i.e. at full stretch of the slingshot). This reduces the number of decision points in the search, which significantly reduces the branching factor. For the dynamics, events model collisions between birds, pigs, blocks, platforms, TNT blocks, and the ground, whereas processes capture the ballistic motion of birds under gravity and changing the possible angle of launch.

While each domain has its own meta model, the HYDRA planner itself is a domain-independent PDDL+ planner. That is, both domains use the same planner, and adding a new domain would only require creating a corresponding meta model for it.

4.2 Consistency Checking

Given a PDDL+ domain D, problem Π, and plan π, one can simulate the trajectory we *expect* to observe when executing π to solve Π in domain D. Thus, we can apply our consistency checker as described above to detect novelties. That is, when playing an episode, HYDRA observes the game and collects periodic states from the game API. Then, HYDRA maps these states to PDDL+ states using the meta model. It determines whether the sequence of observed states is consistent with the sequence of states it expected to observe according to the model. 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 Science Birds our PDDL+ model is not accurate enough to exactly match the observed state trajectory. Note that it was accurate enough to allow the off-the-shelf PDDL+ planner to create plans that yielded state-of-the-art results in the non-novelty version of Science Birds. To address this, we have explored additions: (1) Using representation learning, we have trained a neural network over non-novelty levels to do next state prediction. (2) We created a domain-specific consistency checking component that incorporates qualitative analysis of the observed and expected trajectories. In particular, we matched the initial trajectory of the bird that was shot, the set of blocks destroyed at the end of each shot, and the types of objects that appears in the level. The current implementation of the Science Birds' consistency checker is a work-in-progress, and improving its accuracy is a topic for ongoing research.

[[Roni: not sure how to say all this in a nice way. Maybe push this to the experiments, Klenk: Agreed Just Replace the Sentence and say something like, HYDRA combines these sources of information to perform novelty detection. Wiktor: I edited it to sound a bit more researchy, see what you think.]]

4.3 Meta Model Repair

In our implementation we limited the MMOs we consider to only modify the different constants used in the domain, e.g., gravity and pole length, by a fixed amount (either positive or negative). As noted above, to check if a sequence of MMOs is consistent with the observations, 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 sequence of states observed in the game. The space of possible MMO sequences is combinatorial. 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.

This implementation of the MMR is domain independent and is, in fact, the exact same source code for both domains. However, in general, the set of possible MMOs to use for the meta-model repair

is a domain-specific design choice. We discuss these choices later in the Discussion section.

5 EXPERIMENTAL RESULTS

We are interested in designing adaptive systems that can deal with novelties introduced during their deployment robustly. There are two desired properties of such systems. First, they should detect and react to novelty *quickly* with few interactions with the environment. This is desired because often, it is infeasible to collect large amounts of data during deployment. Secondly, the adaptation in the system should be *interpretable* by a human. This dimension is useful for adaptation during deployment as it enables a human designer or operator to inspect why the system behavior has evolved and ascertain if that adaptation is correct. Below we evaluate our proposed approach in Hydra along these dimensions and analyze its performance.

TODO: Shorten paragraph above

Experiments Our experiments were conducted in the CartPole domain. This domain is defined by specific values of several parameters including mass of the cart, length of the pole, gravity, and friction coefficient.

Ideally, we would put here the analytical control solution for this, which uses all these parameters and explain them. Not critical

Each trial starts with the non-novel environmental setup in which we measure an agent's performance when the environment reflects what it has been designed or trained for. After few episodes, we introduce a novelty in the environment by changing some of the parameters that define the dynamics of the domain. Our experiments had two conditions: *system detection* 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 *given detection* in which an oracle indicated the presence of novelty to the agent.

We ran our experiments for two novelties: (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. The selected novelty was introduced in the 8^{th} episode in a trial. Every trial consisted of 30 episodes, and each experiment consisted of 5 trials.

Baselines In addition to Hydra, we report performance to 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. These baseline agents were designed and trained to achieve perfect performance in the non-novelty case prior to the experiment. The planning agent does not attempt to learn or adapt to novelty, and so its behavior for both experiment conditions (system detection and given detection) is the same. The DQN agent makes Bellman updates to its trained Q-network when novelty was indicated to it in the given detection experiment condition.

What about the learning factor? do we reset it to learn from scratch?

Results The results are summarized in Figure 4. 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 a 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" condition and the orange line shows the results for the "given detection" condition.

As shown in Figure 4, all agents begins at perfect performance at the beginning of the trial and then experience a significant drop in performance as soon as the novelty is introduced in episode 8. This shows the selected novelties are meaningful for all agents. The planning agent and Hydra drop to 50% performance for both novelties and both experiment conditions. The performance drop in the DQN agent is more drastic at 25% and below for both conditions. This difference is expected – 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 supports the claim that composable component models can be more robust to novelties.

Last sentence is tricky?

Since the planning agent does not react to novelties directly, so its performance does not improve over time in both experimental conditions (given detection and system detection). The DQN agent does attempt to adapt to 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 (75 episodes, not shown in the figure) with the environment to return to optimal performance. Hydra adapts very quickly in less than 20 episodes in both experimental conditions. This observation supports the central thesis in this paper that model-based methods enable localization of novelty adaptation, making the corresponding learning very efficient. Hydra performs similarly in both experimental conditions, suggesting that Hydra can reliably detect novelty even when it is not indicated using consistency checking.

Ideally, we would hav novelty detection results for these novelties to support this

6 RELATED WORK

The topic of how to detect and react to novelties has been gaining significant attention in the AI literature. The novely problems we address in this work has been described by Senator [16] and analyzed by Boult et al. [1] and Langely [7]. Boult et al. [1, 7] 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*).

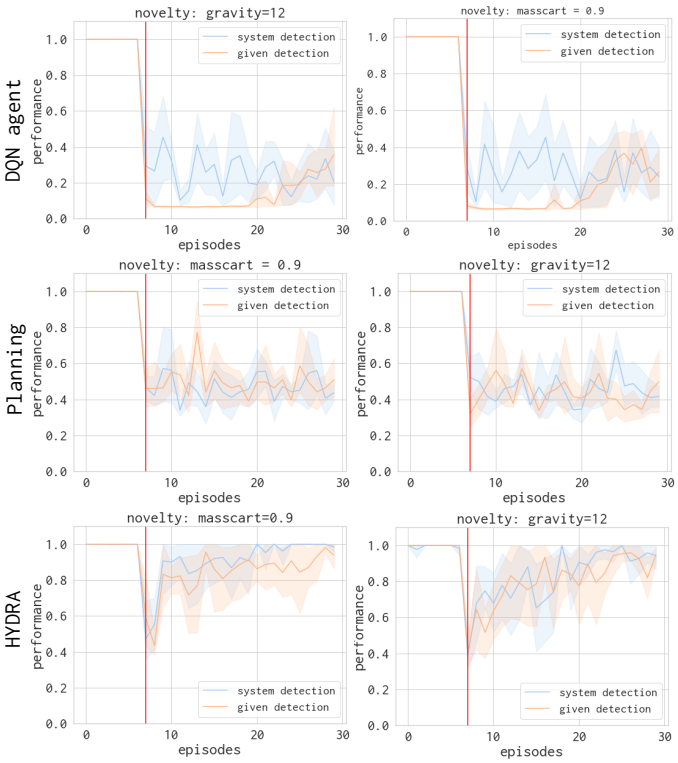

Figure 4: Total reward (y-axis) for each episode (x-axis) in a trial. Red line indicates the episode where novelty has been introduced. The results clearly show that HYDRA adapts very quickly to the novelties in our experiment, returning to optimal performance in 10 episodes.

The top line has the gravity 12 as the left plot but in the other lines it is reversed. Can you fix this? Not critical

We are not the first to design a novelty-robust agent and novelty detection and response algorithms. DiscoverHistory [10] 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 [12] is a cognitive architecture for designing novelty-robust agents, where novelty is limited to which goal to pursue next.

More recently, there have been attempts at defining more general frameworks for handling novelty in open-world environments. Muhammad et al. [11] 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 [6], 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 CONCLUSIONS AND FUTURE WORK

We presented a design for an agent called HYDRA, which can detect and adapt to novel situations automatically. HYDRA is a model-based agents, in the sense that to choose how to act in an environment it uses an internal surrogate model of the environment. Mapping observations and actions from the environment to the surrogate environment model is done using a *meta model* that describes such conversions. To detect when novelty occurs, HYDRA compares the observations it collects with the one it expects according to its surrogate environment model. To adapt to novelties, HYDRA automatically attempts to repair its meta model based on observations. This is done by diagnosing its meta model and employing meta model manipulation operators (MMOs) to change the meta model as needed. We demonstrate HYDRA in the Cartpole domain, a well-known DQN benchmark. Our results show that introducing novelties in this domain is possible and results in catastrophic performance for both planning and DQN agents. HYDRA is able to detect and adapt to novelties in this domain very quickly, requiring as many as 20 episodes before returning to normal performance.

Novelty types	Domain adjustment	Novelty examples
Spatio-temporal trans.	Fluent changes	Increased gravity
Structures trans.	New objects and fluents	New type of bird
Processes trans.	New and/or changing existing processes	Introduced wind
Constraints trans.	New preconditions and/or changed events	Only explosions kill pigs

Table 1: Description of example novelties that can be encountered in Science Birds, changes to the PDDL+ model required to accommodate them, and their corresponding novelty types defined by [7].

Most components in the HYDRA agent is domain-independent and we are currently implementing it on two other domains. To implement HYDRA in a new domain, one needs to only (1) create the appropriate meta model, and (2) select the appropriate MMOs for novelty repair. In our current implementation, we focused in MMOs that modify the value of constant fluents 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 [7] along with examples from Science Birds, one of the domains we are implementing HYDRA for. 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.

REFERENCES

[1] TE Boult, PA Grabowicz, DS Prijatelj, Roni Stern, Lawrence Holder, Joshua Alspector, M Jafarzadeh, Toqueer Ahmad, AR Dhamija, Chunchun Li, et al. 2021. Towards a Unifying Framework for Formal Theories of Novelty. In *AAAI Conference on Artificial Intelligence*. 15047–15052.

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

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

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

[5] Maria Fox and Derek Long. 2006. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research* 27 (2006), 235–297.

[6] Sriram Gopalakrishnan, Utkarsh Soni, Tung Thai, Panagiotis Lymperopoulos, Matthias Scheutz, and Subbarao Kambhampati. 2021. 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).

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

[8] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. PDDL - The Planning Domain Definition Language. (1998).

[9] Drew V McDermott. 2003. Reasoning about Autonomous Processes in an Estimated-Regression Planner.. In *International Conference on Automated Planning and Scheduling*.

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

[11] Faizan Muhammad, Vasanth Sarathy, Gyan Tatiya, Shivam Goel, Saurav Gyawali, Mateo Guaman, Jivko Sinapov, and Matthias Scheutz. 2021. A Novelty-Centric Agent Architecture for Changing Worlds. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 925–933.

[12] Matthew Paisner, Michael Cox, Michael Maynord, and Don Perlis. 2014. Goal-driven autonomy for cognitive systems. In *Annual Meeting of the Cognitive Science Society*, Vol. 36.

[13] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. 2014. A review of novelty detection. *Signal Processing* 99 (2014), 215–249.

[14] Wiktor Mateusz Piotrowski. 2018. *Heuristics for AI Planning in Hybrid Systems*. Ph.D. Dissertation. King's College London.

[15] Jochen Renz, XiaoYu Ge, Matthew Stephenson, and Peng Zhang. 2019. AI meets Angry Birds. *Nature Machine Intelligence* 1, 7 (2019), 328–328.

[16] Ted Senator. 2019. *Science of AI and Learning for Openworld Novelty (SAIL-ON)*. Technical Report. DARPA.

[17] Sebastian Thrun. 1998. Lifelong learning algorithms. In *Learning to learn*. Springer, 181–209.

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

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