Learning to Operate in Open Worlds by Adapting Planning Models

Extended Abstract

Wiktor Piotrowski
Palo Alto Research Center
Palo Alto, USA
wiktorpi@parc.com

Roni Stern
Ben-Gurion University
Beersheeba, Israel
sternron@bgu.ac.il

Yoni Sher
Palo Alto Research Center
Palo Alto, USA
yoni.sher@parc.com

Jacob Le
Palo Alto Research Center
Palo Alto, USA
jale@parc.com

Matthew Klenk
Toyota Research Institute
Los Altos, USA
matt.klenk@tri.global

Johan deKleer
Palo Alto Research Center
Palo Alto, USA
dekleer@parc.com

Shiwali Mohan
Palo Alto Research Center
Palo Alto, USA
smohan@parc.com

ABSTRACT

Planning agents are ill-equipped to act in novel situations in which their domain model no longer accurately represents the world. We introduce an approach for such agents operating in open worlds that detects the presence of novelties and effectively adapts their domain models and consequent action selection. It uses observations of action execution and measures their divergence from what is expected, according to the environment model, to infer existence of a novelty. Then, it revises the model through a heuristics-guided search over model changes. We report empirical evaluations on the CartPole problem, a standard Reinforcement Learning (RL) benchmark. The results show that our approach can deal with a class of novelties very quickly and in an interpretable fashion.

KEYWORDS

Open World Learning; Planning; Adaptive Agents; Model Repair

ACM Reference Format:
Wiktor Piotrowski, Roni Stern, Yoni Sher, Jacob Le, Matthew Klenk, Johan deKleer, and Shiwali Mohan. 2023. Learning to Operate in Open Worlds by Adapting Planning Models: Extended Abstract. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 3 pages.

1 INTRODUCTION

Artificial intelligence and machine learning research on sequential decision-making usually relies on the *closed world* assumption. That is, all relevant characteristics of the environment are known ahead of deployment, during agent design time. For a decision-making agent that relies on automated planning techniques, knowledge about environmental characteristics is encoded explicitly as a domain model (description of actions, events, processes) that govern the agent's beliefs about the environment's dynamics. In an *open world*, however, the characteristics of the environment often change while the agent is operational [3, 4]. Such changes — *novelties* — can cause a planning agent to fail catastrophically as its knowledge of the environment may become incomplete or incorrect. We explore how planning agents can robustly handle such novelties in an open world. Agents following our design use the planning domain model to also evaluate if observed outcomes diverge from what is expected in the plans it generated. If the divergence is significant, the novelty is inferred and accomodated through heuristics search. This approach is applicable to planning agents implementing various levels of PDDL. Results in this paper are from a system implemented using PDDL+[2] for CartPole [1], a classic control problem.

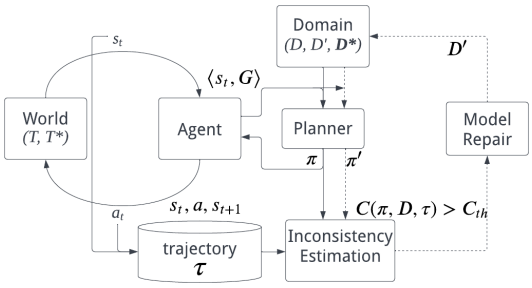

Figure 1: Diagram of novelty reasoning. Solid lines denote the planning process and dotted denote domain model revision.

2 APPROACH

Figure 1 shows the proposed agent design and the novelty reasoning process. The agent interacts with its environment in a sequence of episodes, where each episode is a set of actions taken by the agent to reach a terminal state. At some episode novelty is introduced and the environment changes, the agent is oblivious to the existence, timing, and nature of the introduced novelty.

At an episode's beginning, the agent accepts the current state s_t and creates a corresponding planning problem (s_t, G) which is then paired with the domain model D. Then, it uses a planner to solve the problem to obtain plan π and attempts to execute in the

environment. During execution, it stores the observed trajectory τ as a list of $\langle s_t, a, s_{t+1} \rangle$. At the episode's end, it computes an *inconsistency score* for the current model D by comparing the expected state trajectory with the observed execution trace, τ.

Formally, let $S(\tau)$ be the sequence of states in observations and $S(\pi, D)$ be the expected sequence of states obtained by simulating the generated plan π with the domain model D. Let $S(x)[i]$ denote the i^{th} state in the state sequence $S(x)$. The inconsistency score is computed as $C(\pi, D, \tau) = \sum_i \gamma^i \cdot ||S(\tau)[i] - S(\pi, D)[i]||$ where $0<\gamma<1$ is a discount factor intended to limit the impact of sensing noise. If the inconsistency score exceeds a set threshold C_{th}, the agent infers that its domain model D has become inconsistent with the novel environment characteristics. Then, it initiates the *search-based model repair* process described in Alg. 1 to adjust D accordingly. Alg. 1 searches for a *domain repair* Φ, a sequence of model modifications that, when applied to the agent's internal domain D, returns a domain D' that is consistent with observations. To find such a repair, the algorithm accepts as input a set of basic *Model Manipulation Operators* (MMOs), denoted $\{\varphi\}=\{\varphi_0, \varphi_1, ..., \varphi_n\}$. Each MMO $\varphi_i \in \{\varphi\}$ represents a possible change to the domain. A domain repair Φ is a sequence of one or more basic MMO $\varphi_i \in \{\varphi\}$. An MMO example is to add an amount $\Delta \in \mathbb{R}$ to a numeric domain fluent. After this repair, the agent uses the updated internal domain model D' to solve the subsequent tasks. It may take a few repair steps to find a consistent domain model because a single trajectory may not provide enough information to find the correct repair.

3 RESULTS

We evaluated our approach using a standard implementation of CartPole [1], where the task is to balance the pole in the upright position for n=200 steps by pushing the cart either left or right. The environment reports the velocity and position of the cart and pole. System dynamics are defined by several parameters: cart mass, pole mass, pole length, gravity, pole angle and cart limits, push force.

Fig. 2 summarizes the performance of various agents. We studied two novelties: changing pole length to 1.1 and gravity to 12; and changing pole length to 1.1 and cart mass to 0.9. As baselines, we implemented RL agents that use dynamic q-networks with experience replay[5]. DQN-static uses policy learned in non-novel settings in the novel environment, DQN-dynamic learns a new policy.

Algorithm 1: PDDL+ model repair algorithm.

Input : $\{\varphi\}$: a set of basic MMOs; D: the original PDDL+ domain; π: plan generated using D; τ: a trajectory; C_{th}: consistency threshold
Output: Φ_{best}, a domain repair for D

```
OPEN ← {∅}; C_best ← ∞; φ_best ← ∅
while C_best ≥ C_th do
    Φ ← pop from OPEN
    foreach φ_i ∈ {φ} do
        Φ' ← Φ ∪ φ_i ;                          /* Compose a domain repair */
        DoRepair(Φ', D)
        C_Φ' ← InconsistencyEstimator(π, D, τ)
        if C_Φ ≤ C_best then
            C_best ← C_Φ'
            Φ_best ← Φ'
        Insert Φ' to OPEN with key f(Φ', C_Φ')
        UndoRepair(Φ', D)
```

Results show that the planning agents are, first, *resilient*; the impact of novelty on their performance is not as drastic as on the DQN agents. It is because planning agents use general models that are modular and composable. In the novelty setting, a subset of model elements are still relevant. Second, our approach, the planning-adaptive agent learns *quickly* and recovers optimal performance in ≈20 episodes. This observation supports our central thesis: model-space search enables quick adaptation in dynamic environments because it can localize the learning to specific parts of the explicit model and other parts are *transferred*. In contrast, a DQN agent has to learn new network parameters afresh. Finally, the adaptations are *interpretable*; they are expressed in the same language as the original model, enabling a model designer to inspect what the system has learned. Our method found the following example repairs for CartPole. Each element in the repair is a numeric domain fluent and the reported value is a change from its nominal value.

Repair 1: `mass_cart:0,length_pole:0.3,mass_pole:0, force_mag:0,gravity:0,angle_limit:0,x_limit:0`

Repair 2: `mass_cart:0,length_pole:0,mass_pole:0,force_mag:0, gravity:1.0,angle_limit:0,x_limit:0`

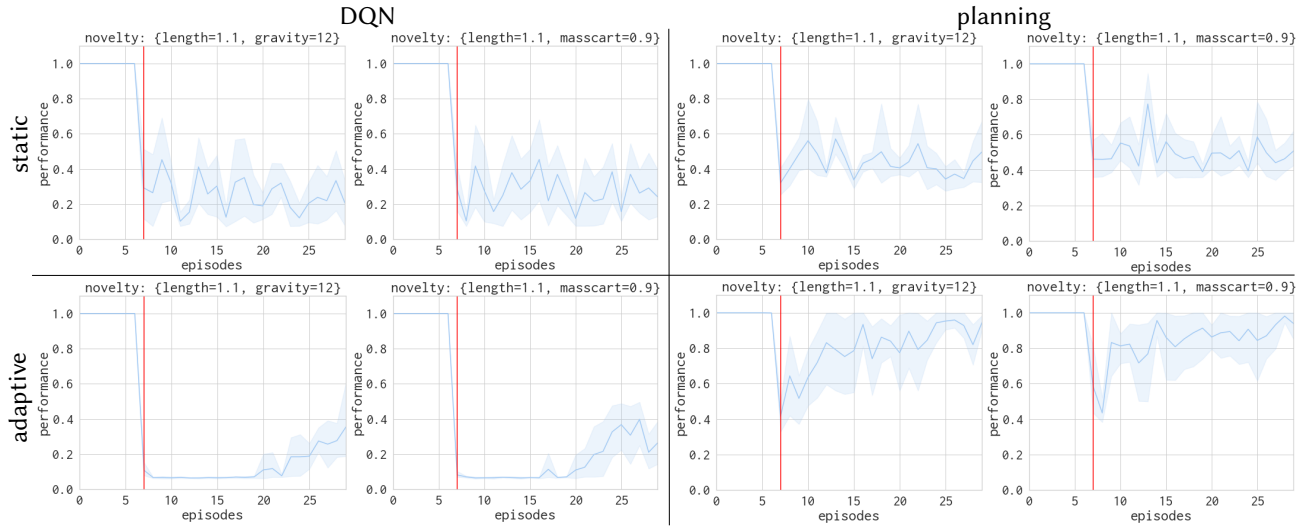

Figure 2: Graphs showing performance of DQN-static/adaptive and planning-static/repairing agents. Episodes are on the x-axis and reward on the y-axis. The results are averaged over 5 trials. Red line indicates the episode 7 when novelty was introduced.

ACKNOWLEDGEMENTS

The work presented in this paper was supported in part by the DARPA SAIL-ON program under award number HR001120C0040. The views, opinions and/or findings expressed are those of the authors' and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

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

[3] Mayank Kejriwal, Abhinav Shrivastava, Eric Kildebeck, Bharat Bhargava, and Carl Vondrick. 2022. Designing Artificial Intelligence for Open Worlds. (2022). https://usc-isi-i2.github.io/AAAI2022SS/.

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

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