Imperative Action Masking for Safe Exploration in Reinforcement Learning

Sumanta Dey$^{(\boxtimes)}$, Sharat Bhat, Pallab Dasgupta, and Soumyajit Dey

Indian Institute of Technology Kharagpur, Kharagpur, India
{sumanta.dey,sharatbhat}@iitkgp.ac.in, {pallab,soumya}@cse.iitkgp.ac.in

Abstract. Reinforcement Learning (RL) needs sufficient exploration to learn an optimal policy. However, exploratory actions could lead the learning agent to safety hazards, not necessarily in the next state but in the future. Therefore, it is essential to evaluate each action beforehand to ensure safety. The exploratory actions and the actions proposed by the RL agent could also be unsafe during training and in the deployment phase. In this work, we have proposed the Imperative Action Masking Framework, a Graph-Plan-based method considering a finite and small look ahead to assess the safety of actions from the current state. This information is used to construct action masks on the run, filtering out the unsafe actions proposed by the RL agent (including the exploitative ones). The Graph-Plan-based method makes our framework interpretable, while the finite and small look ahead makes the proposed method scalable for larger environments. However, considering the finite and small look ahead comes with a cost of overlooking safety beyond the look ahead. We have done a comparative study against the probabilistic safety shield in Pacman and Warehouse environments approach. Our framework has produced better results in terms of both *safety* and reward.

Keywords: Graph-Plan Algorithm · Explainable/Interpretable Machine Learning · Reinforcement Learning · Exploration considering Safety

1 Introduction

Reinforcement Learning (RL) is a reward-based learning approach where the learning agent learns a state-to-action mapping policy based on the reward [19] accumulated during exploration in the underlying environment. RL, therefore, is suitable for many real-world scenarios where the underlying environment is not known a priori. On that account, the exploration is also necessary to capture the state-to-action-to-reward mapping to learn the highest rewarding policy or optimal policy. Therefore, while training, the RL agent takes a few actions that are not optimal according to the current policy, which is called exploratory actions. However, when RL applies in safety-critical environments, where some

D. Calvaresi et al. (Eds.): EXTRAAMAS 2023, LNAI 14127, pp. 130–142, 2023.
https://doi.org/10.1007/978-3-031-40878-6_8

states are unsafe or hazardous, the goal is to learn an optimal policy while avoiding those unsafe states. Therefore actions that potentially could lead to an unsafe state, not necessarily in the very next state could be in the near future, should be avoided. Also, not only the exploratory actions but the action proposed by the RL agent could be unsafe. Therefore, all actions, whether exploratory or optimal, should be checked before taking for safety-critical environments.

For example, consider the Pacman Game layout as shown in Fig. 1. Based on the current position of the Pacman (yellow colored) and the ghost (blue colored), if the Pacman takes the right action from there, then in the next state, it will be in the green block, which might be safe for now as the ghost still not able to catch the Pacman. However, the Pacman will be trapped within the green-to-red block region, and eventually, the ghost will catch the Pacman. Therefore the right action from the current position is not safe.

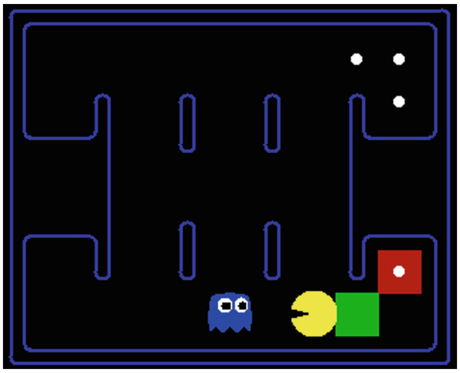


Fig. 1. Pacman Game Layout.

In general, several approaches to reinforcement learning through safe exploration can be broadly classified into Reward Shaping and Action Shaping. Reward Shaping techniques can be found in [2,6,10,18], where safety constraints are included in the reward function as a regularization term. On the other hand, works like [3,7,11,12,14,17] use action-shaping techniques to restrict the RL agent from taking unsafe actions. Apart from these methods, the possibilities of safe exploration using safe baselines/backup policies whenever a safety violation is detected have been demonstrated in [4,13]. On the other hand, in [1], the author proposes a teacher advice-based technique where the RL agent seeks expert or teacher advice whenever it detects any unknown/unsafe situation.

Among the action shaping techniques, [14] and [3] are better choices when safety is the major concern, as these provide a formal safety guarantee even though they suffer from scalability issues. A scalable implementation of [3] can be found in [16]. In [16], the authors use a probabilistic model checker to determine the safety probabilities of the actions, which are later used to filter out the actions having unsafe probabilities beyond a threshold. Using a probabilistic model checker helps to improve scalability but with the cost of safety and interpretability.

In this context, we propose the *Imperative Action Masking framework*, a scalable and interpretable action-shaping technique. Action Mask filters out unsafe actions from the current state. Our framework uses the classic Graph Plan algorithm [9] to determine the action mask considering the state space up to a small finite look ahead. The Graph Plan algorithm is used to improve interpretability, as the outcomes of this algorithm are easy to interpret by a human. In contrast, the small finite look ahead helps to increase the scalability, but with the cost of omitting safety hazards that could occur beyond the look ahead. Therefore, our framework is suitable for ensuring safety for environments with safety violations in the near future; however, it can also be used to reduce the chances of safety violations for environments with very far safety consequences.

In summary, our contributions are as follows:

– We develop a classic Graph Plan-based method to determine the membership probabilities of a state in each Action Specific Robust Set.
– These probabilities are later used to compute the Action-Masks. The Action Masks are applied on top of the RL Agent's decision to ensure safety by filtering out potential unsafe actions. Thus our method can be applied for safe exploration in Reinforcement Learning in various discrete state and action environments.
– The use of the Graph Plan-based method makes the Imperative Action Masking framework interpretable.
– We demonstrate the effectiveness of our method in Pacman and Warehouse Environments. We also compare the results of our method over the Probabilistic Shield presented in [16].

The paper is organized as follows. Section 2 outlines the overall problem statement. Section 3 presents the proposed Imperative Action Masking framework. Experimental environment configuration and results are presented in Sect. 4 and Sect. 5, respectively. Concluding remarks are given in Sect. 6.

2 Problem Formalization

Given a Constrained MDP (CMDP) as defined in [15], $<\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, \mathcal{R}, \mu, \mathcal{C}>$, where $\mathcal{S}$ and $\mathcal{A}$ represent the set of states and the set of actions, respectively. $\mathcal{P}$ is the transition probability function, γ is the discount factor, and $\mathcal{R}$ is the reward function. μ is the start state distribution, and $\mathcal{C} = \{(c_i : \mathcal{S} \rightarrow \{0, 1\}) | i \in \mathbb{Z}\}$ is the set of given binary safety constraints; $c_i : s$ can take either 0 (safe) or 1 (unsafe). $\mathcal{C} : s$ is considered unsafe if any of the $c_i : s$ is 1.

We use X to denote the set of safe states where, $\{\forall i,\ c_i : X \rightarrow 0\}$ and X_u to denote the set of unsafe states where, $\{\exists i,\ c_i : X \rightarrow 1\}$. There can be states ($s_k$ in Fig. 2) in the MDP which does not violate the safety constraints; however, all the possible combinations of legitimate actions will eventually lead to unsafe states. We consider this set of states as pseudo-unsafe states X_p and should be avoided just like the unsafe states. We have defined another two sets of safe states, namely Action Specific Robust Safe Sets ($X_I^{a_i}$) and Robust Safe Set (X_I) that is a subset of $\{X \setminus X_p\}$. Figure 2 depicts a schematic relation among these sets.

Definition 1 (Robust Safe Set). Given an MDP having a set of safe states X and action set $\mathcal{A}$, the Robust Safe Set is defined as:

$$X_I = \{s \mid \forall s \in X_I, \exists a \in \mathcal{A},\ \forall s' \in \mathcal{P}(s,a),\ s' \in X_I\}$$

Definition 2 (Action Specific Robust Safe Set). Given an MDP having a set of safe states X and action set $\mathcal{A}$, the Action Specific Robust Safe Set is defined as:

$$X_I^{a_i} = \{s \mid \forall s \in X_I^{a_i},\ s' \in \mathcal{P}(s,a_i),\ s' \in X_I\}$$

Hence, the Robust Safe Set (X_I) is $\bigcup\limits_{\forall a \in \mathcal{A}} X_I^a$.

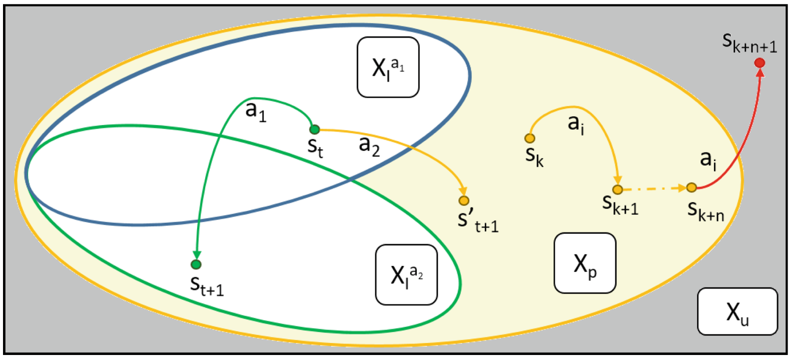


Fig. 2. The yellow bordered region represents the set of safe states(X). The blue and green encircled regions represent the Action Specific Robust Safe Sets for action a_1 and a_2, respectively. The white, yellow, and gray colored regions represent the Robust Safe Set (X_I), Pseudo Unsafe Set X_p, and set of unsafe states X_u, respectively. (Color figure online)

We consider using the action masking method to keep the RL agent within X. The mask value for unsafe actions' will be zero and omitted from the masked action list. Hence, the action is chosen from the masked distribution $\mathcal{M} : \pi$ will always be safe, where M denotes the masking function. Our Imperative Action Masking problem can be formulated as follows:

Problem 1 *(Imperative Action Masking Problem)*
Given a CMDP, a state $s_t \in X$ and, current policy π, the Online Imperative Action Masking Problem is to find an action mask $\mathcal{M}_{s_t}$ for the state s_t such that the next state s_{t+1} for any action a_t sampled from the masked distribution $\mathcal{M}_{s_t} : \pi(s_t)$ belongs to X, i.e., $\mathcal{P}(s_t, a_t) \rightarrow s_{t+1} \in X$.

3 Imperative Action Masking Framework

To construct Action Mask ($\mathcal{M}_{s_t}$) for a state (s_t) requires information about the state's membership in different Action Specific Robust Safe Sets. The masking value for an action is decided as follows.

$$\mathcal{M}_{s_t}[a_t] = \begin{cases} 1, & if\ s_t \in X_I^{a_t}; \\ 0, & Otherwise. \end{cases}$$

However, it is not computationally feasible for MDPs with huge state space to calculate the Robust Safe Set for the entire state space, covering all future time steps. It is also hard to accurately determine future states beyond a particular horizon if the MDP is not fully known. Therefore, we approximate the membership values of a state s_t in an Action Specific Robust Safe Set $X_I^{a_t}$ by checking all the possible states up to a small look ahead l. If the next state s_{t+1} for the action a_t belongs to X and there exist a trace $\tau = s_{t+2}, ..., s_{t+l}$ from s_{t+1}, such that $i = 2..l\ s_{t_i} \in X$ then we consider that the state $s_t \in X_I^{a_t}$.

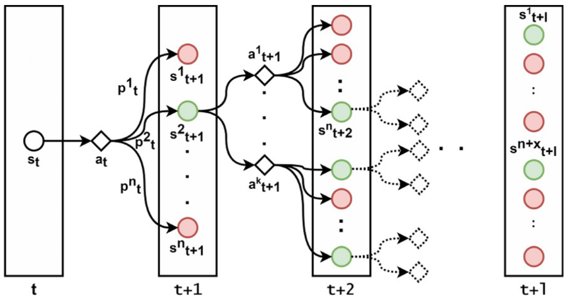


Fig. 3. Membership Check using Graph Plan Algorithm. Each rectangle denotes the state space of Graph-Plan for a time instance. Whereas Circles denote the states and Diamonds represent the action. Red circles denote that the state is unsafe or $\notin X$ and Green circles denote that the state is safe. As shown in the figure, only the safe states are expanded for the next time instances of the Graph-Plan Algorithm. (Color figure online)

We consider the well-established Graph Plan algorithm [9] to efficiently check for such traces τ. The actions of the MDP are considered the actions for the Graph Plan algorithm, and the effect of the actions are the next states as shown in Fig. 3. Two preconditions ($\neg explored$, $s \in X$) for all the actions are also considered to improve efficiency. To determine the membership of a state s_t in the Action Specific Robust Safe Set of action a_t in the first stage of the Graph Plan Algorithm, we apply action a_t only. All the actions that satisfy the preconditions are applied in the later stages. For non-deterministic MDP, the membership function $\mathcal{F}$ also becomes probabilistic $\mathcal{F}(X_I^a, s) \rightarrow [0, 1]$. In this case, the membership probability of only the maximum reachability probability

Algorithm 1: Determining Membership using GraphPlan

```
    Input: Action List $\mathcal{A}$, Safe States $X$, Current State $s_t$, Given Action $a_t$, Look
           Ahead $l$
 1  Function Membership($A, X, s_t, a_t, l$):
 2      if $s_t \notin X$ then
 3          return 0
 4      $\mathcal{F}_0(s_t) = 1$
 5      $GPS = \{s_t\}$
 6      $GPA = \{a_t\}$
 7      for $i = 1$ to $l$ do
 8          if $\forall s \in GPS : s \notin X$ then
 9              return 0
10          $GPS \leftarrow ExpandGraph(GPS, GPA)$
11          for $s' \in GPS$ do
12              if $s' \notin X$ then
13                  $\mathcal{F}_i(s') = 0$
14              else
15                  $\mathcal{F}_i(s') = max\Big[\forall s, a \in Parent(s'),\ \mathcal{F}_i(s) \times \mathcal{P}(s, a, s')\Big]$
16          end
17          $GPA \leftarrow \mathcal{A}$
18      end
19      return $max[\mathcal{F}_l]$
20  Comments:
21  $GPS$: Current State Set of the Graph-Plan Algorithms
22  $GPA$: Current Action set for the Graph-Plan Algorithm
23  $ExpandGraph$: Returns the set of next states from the states in GPS by taking
    actions in GAP while satisfying the preconditions
24  $Parent(s')$: Return all the state, action combination used in $GPS_{t-1}$ to reach $s'$
    in $GPS_t$.
```

to all safe states presented in the $(t+l)$ time step of the Graph Plan algorithm is considered the membership probability. The membership probability $\mathcal{F}(X_I^{a_t}, s_t)$ is determined as follows:

$$\mathcal{F}(X_I^{a_t}, s_t) = max\Bigg[\forall s_{t+l} \in X, max\Bigg[\forall \tau,\ \tau[-1] = s_{t+l}, \prod_{(s,a,s') \in \tau} \mathcal{P}(s, a, s')\Bigg]\Bigg]$$

Here, $\tau[-1]$ returns the last state of the trajectory τ. In the end, it returns maximum $\mathcal{F}$ values among the states of Graph-Plan l-th time step. This membership value is used to decide the action mask for the state s_t.

$$\mathcal{M}_{s_t}[a_t] = \begin{cases} 1, & if\ \mathcal{F}(X_I^{a_t}, s_t) \geq \mathcal{H}; \\ 0, & Otherwise. \end{cases}$$

Here, $\mathcal{H}$ is the safety threshold range between [0,1]. The $\mathcal{H}$ value is a design choice. However, with the higher value of $\mathcal{H}$ exploration process becomes more conservative. Algorithm 1 outlines the overall process flow to find the membership values. Line 2 of the algorithm checks if the state s_t is in X or else it returns the membership value as zero (Line 3). If the state s_t is in X, the membership value is initialized with one (Line 4). The Graph Plan State list (GPS) and Graph Plan Action list (GPA) are also initialized with the current state and action. Then the graph is expanded (Line 10) up to depth l (look ahead value) (Line 7–18). However, after each level expansion, the new states of that level are checked if those states are in X, or else the membership for those states is assigned to zero (Line 13). If the states are in X, then their temporary membership values up to layer i are updated with the maximum value of their parent states ($i-1$ layer) value times corresponding transition probability, as shown in Line 15. Finally, in Line 19, the maximum temporary membership value among the last graph plan layer or $l-th$ layer states are returned as the membership value of state s_t in the corresponding Robust Set of Action a_t.

Graph Plan [9] is a simple and intuitive graph-based planning algorithm. The use of the Graph Plan-based algorithm for determining whether the action is safe or unsafe in our framework also makes the process intuitive and interpretable. For example, consider the Pacman example shown in Fig. 1. Here based on the current situation, our Imperative Action Masking Framework is able to identify the Left action as Safe even though the Left action may lead the agent to a trap or dead-end (colored in red) from which it cannot run away from an optimal ghost, but it also has a safe state (colored in green) from which it can run away from the ghost. Determining all these action labels (safe/unsafe) is quite intuitive and interpretable.

4 Environment Setup

In order to demonstrate the working of our method, we have used two different environments: (1) Pacman and (2) Warehouse. We have also presented a performance comparison analysis of our method with the Probabilistic Safety Shield for both environments. Table 1 describes the values of the hyperparameters used in the experiments, and we consider the default reward settings for all the environments. All the hyperparameter values used here are chosen empirically while keeping the environment settings or parameters like reward settings as default. Our implementation is uploaded as a public GitHub repository[1].

4.1 Pacman

In this environment, [5], the Pacman (Learning Agent) navigates through the grid as shown in Fig. 4 aims to collect all the pellets/food in the maze in the least possible steps without getting caught by the chasing ghosts (Adversarial Agents). The Pacman receives a positive reward of +10 for collecting a pellet and a negative reward of -1 at each time step. A large positive reward of +500

[1] https://github.com/sumantasunny/ImperativeActionMasking.git.

Table 1. Hyper-parameter values used in the experiments.

Hyper-Parameters	Values
Discount Factor γ	0.8
Learning Rate α	0.2
Initial/Min ϵ	0.90/0.05
ϵ-Decay	0.90
Max Steps/Episode	10000
Safety Threshold $\mathcal{H}$	1

is received on collecting all the dots, and on getting caught by the ghosts, a large negative reward of -500 is received. Then the game restarts. We have used directional ghosts (Probability of following the Pacman = 0.9), thus enabling a more severe safety-critical environment. We have chosen two types of layouts - Grid Trap (Fig. 4b) is a layout with dead-ends (traps), while Original Classic (Fig. 4a) and Medium Classic (Fig. 5a) layouts are without dead-end (no traps) for better comparison.

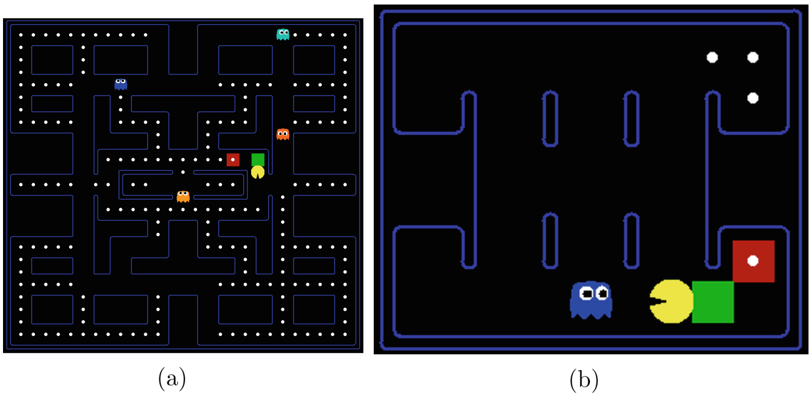


Fig. 4. (a) Original Classic Pacman Layout. (b) Grid Trap Pacman Layout. The yellow bot is the Pacman, controlled by the RL Agent. (Color figure online)

4.2 Warehouse

This environment, as shown in Fig. 5b, consists of a warehouse floor plan containing packages on the shelves and an exit. The yellow fork-lifter (Learning Agent) has to collect all the packages from the respective shelves and deliver them at the exit by navigating through the narrow corridors. It has to make sure that it does not collide with other fork-lifters, similar to the setup in [8].

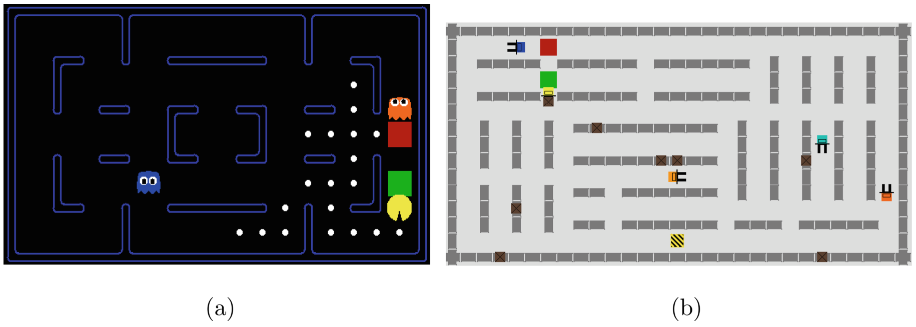


Fig. 5. (a)Medium Classic Pacman Layout. (b) Warehouse Grid Layout. The RL Agent controls the yellow robot. (Color figure online)

The yellow fork-lifter receives a positive reward of +20 on respectively loading and delivering each package and a negative reward of -1 at each time step. On delivering all the packages, a large positive reward of +500 is received, and on collision with other fork lifters, it gets a large negative reward of -500. All the fork lifters have only one exit in the entire floor plan, creating a safety-critical condition at the exit.

5 Results

The experimental results were produced in a Ubuntu 20.04 OS with 16 GB physical memory and Intel Core-i7 8th Gen processor. As a programming language, we use Python 3.7. As both environments are discrete, we use Tabular Q-Learning-based RL Agents. The used hyper-parameters are given in Table 1.

The RL algorithm has been trained for 300 episodes in each environmental setup, and each episode runs till termination. The discount factor $\gamma = 0.8$, learning rate $\alpha = 0.2$, and $\epsilon = 0.05$ in the ϵ-greedy exploration policy are used as hyper-parameters. The safety threshold $\mathcal{H} = 1$ is used throughout the experiment, i.e., if the action is not completely safe, it is masked. We have taken the plots of the respective agents' average scores and winning rates in their respective environmental setups during training over different look ahead values, which can be seen in Fig. 6, 8. The comparison of the training scores over several episodes in each environmental setup using various action-masking methods can be seen in Fig. 7, 9.

5.1 Pacman

We can observe from Fig. 6a and Fig. 6b that for all grids, the training scores and winning rates are very low when the look ahead is either very small or very large. This is because, for a smaller look ahead, the action masker fails to mask those actions that could be potentially hazardous in the future, if not immediately.

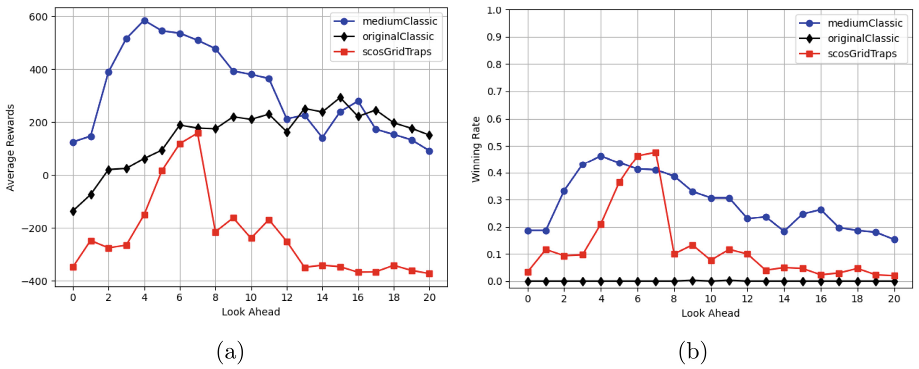


Fig. 6. Training Results on different look ahead for different Pacman grids. (a) Pacman's average training score over different look ahead. (b) Pacman's average winning rate over different look ahead.

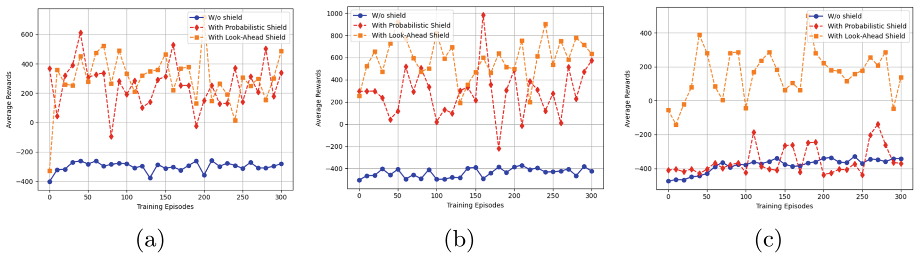
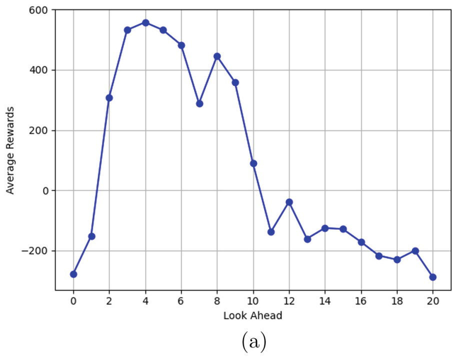


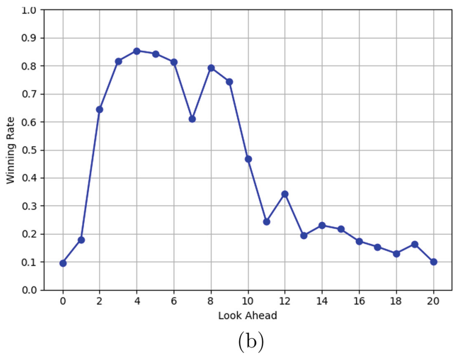
Fig. 7. Training Results on different Pacman grids. (a) Training Scores on original classic Pacman. (b) Training Scores on medium classic Pacman. (c) Training Scores on Pacman grid containing traps.

On the other hand, for a larger look ahead, the action masker becomes over-conservative and thus may end up masking even those actions which practically may not be unsafe. As a result, we see an optimal look ahead, leading to the optimal performance of the Pacman in each of the grids.

Figure 7 compares the scores of RL approaches using our method of GraphPlan-based look ahead shield to the unshielded RL approaches and RL approaches using probabilistic shield constructed via model checker as in [16]. Figure 7a and Fig. 7b correspond to the grid without traps (dead ends), while Fig. 7c corresponds to the grid with traps (dead ends). While outperforming the unshielded RL method in the first two scenarios, our method performed better in the third scenario, whereas the model checker-based probabilistic shield method could not. Where our method is able to accumulate an average of +200 reward, the model checker-based probabilistic shield performs similarly to the unshielded RL method with an average of -400 reward.



(a)



(b)

Fig. 8. Training Results on different look ahead for the warehouse floor plan (a) Fork-lifter's average training score over different look ahead. (b) Fork-lifter's average winning rate over different look aheads.

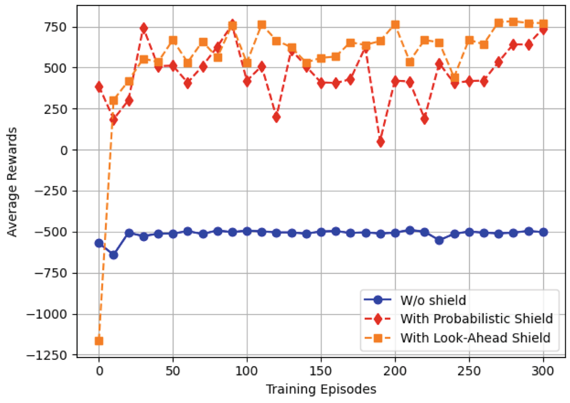


Fig. 9. Training Scores on warehouse environment.

5.2 Warehouse

Just like in the Pacman environment, as we can see from the overall trend of the graphs in Figures Fig. 8a and Fig. 8b, the reward and winning rate increases with the look ahead. However, after reaching a peak, the reward and winning rate start decreasing with the further increase of look ahead length. A large look ahead leads to overprotective scenarios and may end up masking not-so-unsafe action, as shown in Fig. 8a and Fig. 8b.

The comparison of the scores of the RL approach using Graph Plan-based look ahead shield of look ahead (l) 4 to the unshielded RL approach and RL approach using probabilistic shield constructed via model checker as in [16] has been shown in Fig. 9 where it is clearly visible that both the shield approach performs much better than the unshielded RL method. Also, our RL with the look ahead-based shielding approach performs slightly better in terms of rewards than the RL with the probabilistic shield.

6 Conclusion

We have presented an interpretable safe exploration method in safety-critical environments with discrete state action in RL. We use the classic Graph Plan algorithm to calculate the membership probabilities in each Action Specific Robust Safe Set for all states up to a small finite look ahead. The Graph Plan algorithm helps to improve the interpretability where the small finite look ahead helps to increase the scalability. Using membership probabilities to construct Action Masks in our proposed framework helps ensure safety with a certain probability. We have also presented various experiments that empirically validate that our method incurs fewer safety incidents while achieving higher rewards than the Probabilistic Safety Shield technique.

References

1. Abbeel, P., Coates, A., Ng, A.: Autonomous helicopter aerobatics through apprenticeship learning. Int. J. Robot. Res. **29**, 1608–1639 (2010)
2. Achiam, J., Held, D., Tamar, A., Abbeel, P.: Constrained policy optimization. In: International Conference on Machine Learning (2017)
3. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. In: AAAI Conference on Artificial Intelligence (2017)
4. Amodei, D., Olah, C., Steinhardt, J., Christiano, P.F., Schulman, J., Mané, D.: Concrete problems in AI safety. arXiv abs/1606.06565 (2016)
5. Berkeley, U.: UC Berkeley CS188 Intro to AI reinforcement learning. http://ai.berkeley.edu/reinforcement.html Accessed 14 Jun 2023
6. Berkenkamp, F., Moriconi, R., Schoellig, A.P., Krause, A.: Safe learning of regions of attraction for uncertain, nonlinear systems with gaussian processes. In: 2016 IEEE 55th Conference on Decision and Control (CDC), pp. 4661–4666 (2016)
7. Bharadhwaj, H., Kumar, A., Rhinehart, N., Levine, S., Shkurti, F., Garg, A.: Conservative safety critics for exploration. arXiv abs/2010.14497 (2020)
8. Bit-Monnot, A., Leofante, F., Pulina, L., Ábrahám, E., Tacchella, A.: Smartplan: a task planner for smart factories. arXiv abs/1806.07135 (2018)
9. Blum, A., Furst, M.L.: Fast planning through planning graph analysis. In: International Joint Conference on Artificial Intelligence (1995)
10. Chow, Y., Nachum, O., Duéñez-Guzmán, E.A., Ghavamzadeh, M.: A lyapunov-based approach to safe reinforcement learning. In: Neural Information Processing Systems (2018)
11. Dey, S., Dasgupta, P., Dey, S.: Safe reinforcement learning through phasic safety oriented policy optimization. In: SafeAI@AAAI Conference on Artificial Intelligence (2023)
12. Dey, S., Mujumdar, A., Dasgupta, P., Dey, S.: Adaptive safety shields for reinforcement learning-based cell shaping. IEEE Trans. Netw. Serv. Manage. **19**, 5034–5043 (2022)

13. Feghhi, S., Aumayr, E., Vannella, F., Hakim, E.A., Iakovidis, G.: Safe reinforcement learning for antenna tilt optimisation using shielding and multiple baselines. arXiv abs/2012.01296 (2020)
14. Fulton, N., Platzer, A.: Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In: AAAI Conference on Artificial Intelligence (2018)
15. García, J., Fernández, F.: A comprehensive survey on safe reinforcement learning. J. Mach. Learn. Res. **16**, 1437–1480 (2015)
16. Jansen, N., Könighofer, B., Junges, S., Serban, A.C., Bloem, R.: Safe reinforcement learning using probabilistic shields (invited paper). In: International Conference on Concurrency Theory (2020)
17. Nikou, A., Mujumdar, A., Orlic, M., Feljan, A.V.: Symbolic reinforcement learning for safe ran control. In: Adaptive Agents and Multi-Agent Systems (2021)
18. Perkins, T.J., Barto, A.G.: Lyapunov design for safe reinforcement learning. J. Mach. Learn. Res. 3(null), 803–832 (mar 2003)
19. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)