DEPARTMENT OF COMPUTER SCIENCE

Sample Efficient Model-free Reinforcement Learning from LTL Specifications with Optimality Guarantees

International Joint Conference on Artificial Intelligence 2023

Daqian Shao and Marta Kwiatkowska
daqian.shao@cs.ox.ac.uk

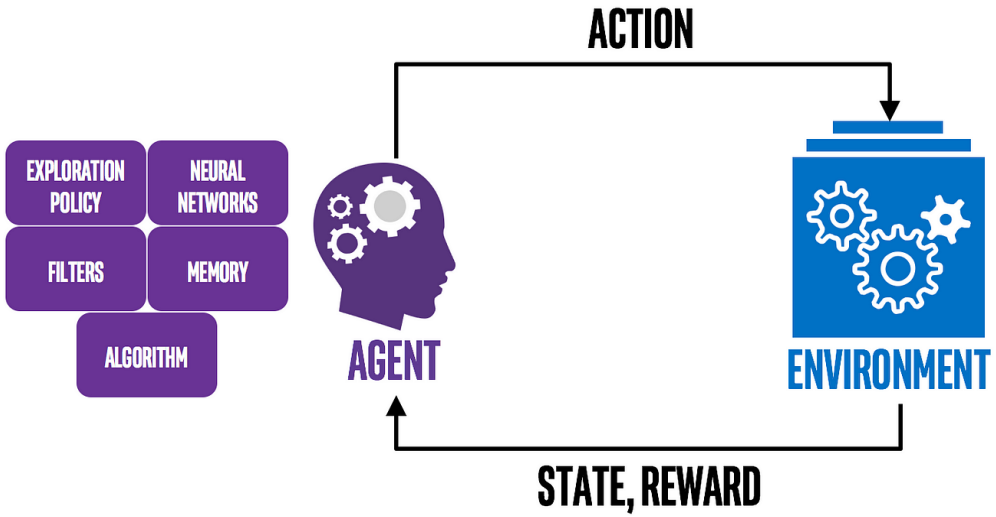
IJCAI/2023 MACAO

Reinforcement Learning

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF OXFORD

- Reinforcement Learning[1] teaches an agent in a particular environment how to choose an action from its action space, in order to maximize rewards over time.
- The environments are normally modeled as Markov Decision Processes (MDPs)



- Model-Free RL: Q learning[2]

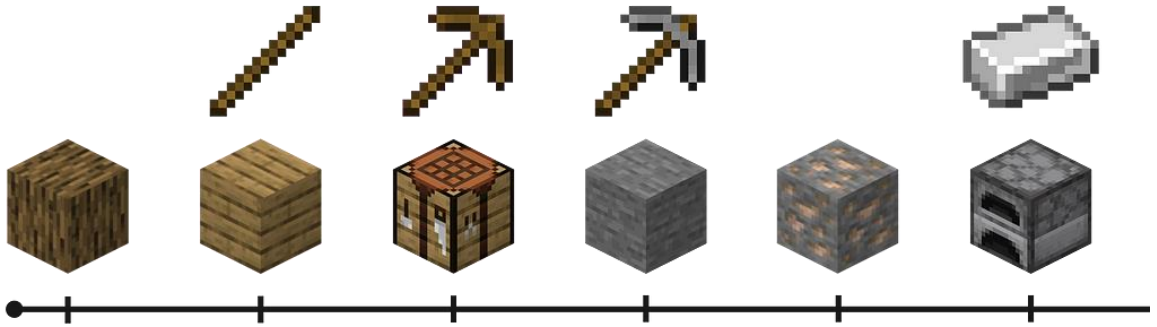
$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\}$$

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\}$$

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An Introduction (2nd edition 2018)*, vol. 3, no. 9. 2018.
[2] Christopher J. C. H et al. Q-learning. Machine Learning 1992

Reinforcement Learning

- Many tasks can be easily specified by some formal language, but how to learn that task with reward-based RL?



- In this paper, we consider the problem of how to efficiently learn optimal policies of tasks specified by a very expressive logic language, Linear Temporal Logic.

Linear Temporal Logic (LTL)

An LTL formula can be satisfied by an infinite sequence of evaluations of variables in AP (atomic propositions), and it is defined recursively

$$\varphi ::= true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathsf{X}\,\varphi \mid \varphi_1 \mathsf{U}\,\varphi_2, \quad a \in \mathcal{AP}$$

$$\begin{aligned} \sigma \text{ satisfies the LTL formula } \varphi \text{ if } \quad & a \in L(\sigma[0]) && \text{for } a \in \mathcal{AP}; \\ & \sigma[1:] \models \varphi && \text{for } \mathsf{X}\,\varphi; \\ & \exists i, \sigma[i] \models \varphi_2 \text{ and } \forall j < i, \sigma[j] \models \varphi_1 && \text{for } \varphi_1 \mathsf{U}\,\varphi_2 \end{aligned}$$

eventually: $\mathsf{F}\,\varphi = true \mathsf{U}\,\varphi$; *and always:* $\mathsf{G}\,\varphi = \neg(\mathsf{F}\,\neg\varphi)$.

Example: F G b & G !c

Meaning to eventually only visit label ‘b’ in the future while never visit label ‘c’

From LTL to reward-based RL

Learning a policy optimized for LTL specifications are not trivial using reward

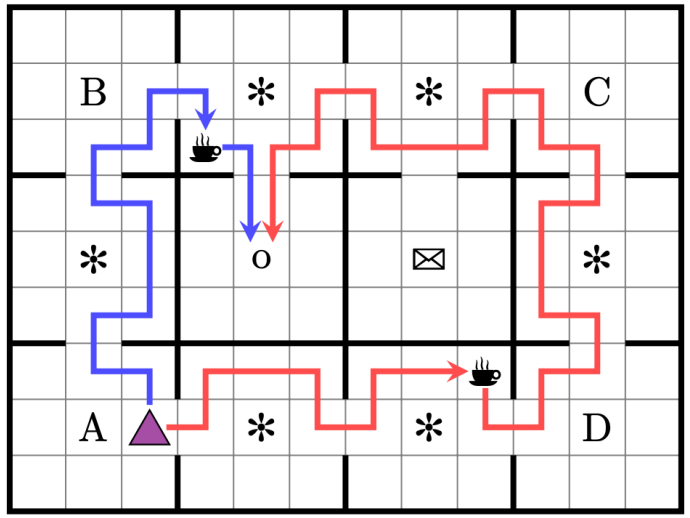
1. When and where to give reward to the agent?
2. How much reward to give?

Need to show that the policy yielding the highest expected (discounted) reward is also the policy with the highest probability of satisfying the LTL specification.

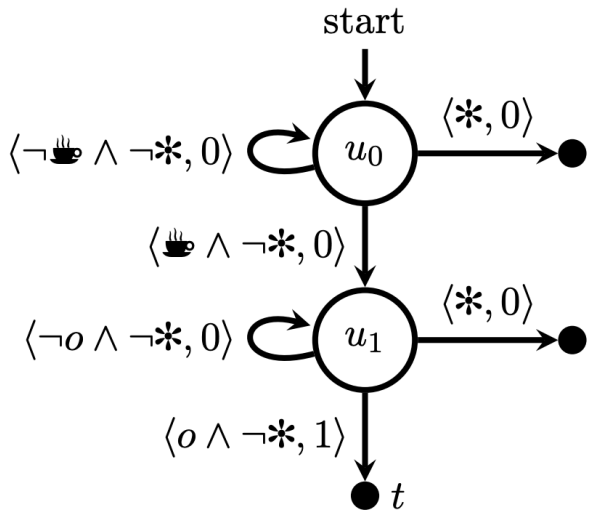
Transforming Specifications

Task specifications can be transformed into Automata

- Reward Machine [3]: get tea and go to office without hitting obstacles



(a) The office gridworld



(b) A simple reward machine

This is equivalent to Deterministic Finite Automata (DFA) or regular expressions

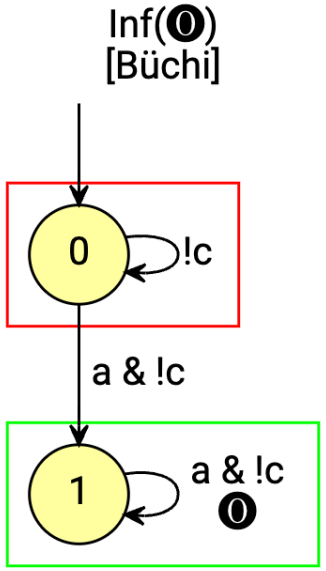
[3] Rodrigo Toro Icarte et al. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. Journal of Artificial Intelligence Research, 2022

Transforming LTL Specifications

LTL language is strictly more powerful than regular expressions because it can specify infinite horizon properties.

But it can also be transformed into Automata!

- ω-automata take infinite strings as inputs:
  - Limit-deterministic Büchi Automata (LDBA): an infinite run is accepting if and only if the **accepting states are visited infinitely many times**.
  - The non-determinism is handled by additional epsilon-actions given to the agent



LTL specification: F G a & G ! c

Generalized Product MDP

Definition 3.0.3 (Product MDP). *Given an MDP* $\mathcal{M} = (S, s_0, A, T, \mathcal{AP}, L)$*, an LDBA* $\mathcal{A} = (\mathcal{AP}, \mathcal{Q}, q_0, \Delta, \mathcal{F})$ *and* $K \in \mathbb{N}$*, we construct the product MDP as follows:* $\mathcal{M}^\times = \mathcal{M} \times \mathcal{A} \times [0..K] = (S^\times, s_0^\times, A^\times, T^\times, \mathcal{F}^\times)$*, where the product states* $S^\times = S \times \mathcal{Q} \times [0..K]$*, the initial state* $s_0^\times = (s_0, q_0, 0)$*, the product actions* $A^\times = A \cup \{\epsilon_q \mid q \in \mathcal{Q}\}$*, and the product transitions* $T^\times : S^\times \times A^\times \times S^\times \to [0, 1]$ *are defined as below:*

$$T^\times((s,q,n),a,(\hat{s},\hat{q},n)) = \begin{cases} T(s,a,\hat{s}) & \text{if } a \in A \text{ and } \hat{q} \in \Delta(q, L(s)) \setminus \mathcal{F}; \\ 1 & \text{if } a = \epsilon_{\hat{q}}, \hat{q} \in \Delta(q,\epsilon) \text{ and } \hat{s} = s; \\ 0 & \text{otherwise .} \end{cases}$$

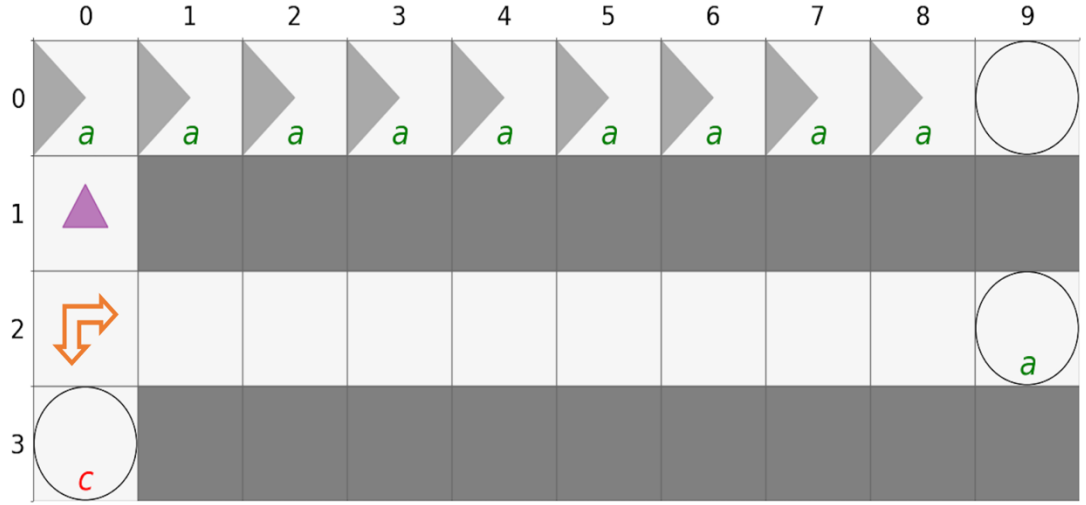
$$T^\times((s,q,n),a,(\hat{s},\hat{q},\min(n+1,K))) = \begin{cases} T(s,a,\hat{s}) & \text{if } a \in A \text{ and } \hat{q} \in \Delta(q, L(s)) \cap \mathcal{F}; \\ 0 & \text{otherwise .} \end{cases}$$

- The agent's action first transitions the environment MDP
- The label of the current MDP state or an epsilon-action transitions the automaton
- Accepting states are states with accepting automaton state
- In order to introduce flexibility on how much reward to give to each visit of the accepting state, we use a counter to count the number of accepting states visited in a run.

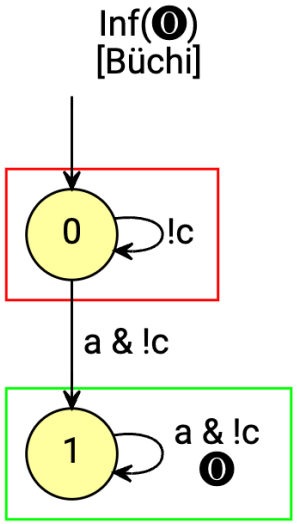
Generalized Reward structure

Why count the number of visits to accepting states?

Motivating example:
Consider this environment MDP
LTL specification: F G a & G ! c



The purple triangle at (1,0) is the starting point and the bidirectional arrow at (2,0) is a probability gate



- We can allow more exploration by assigning very small rewards to initially visited accepting states and gradually increase the reward as more accepting states are visited.

- Exploration and exploitation trade-off

Generalized Reward structure

Definition 3.1.1 (Reward Structure). *Given a product MDP $\mathcal{M}^\times$ and a policy π, the product reward function $r^\times : S^\times \times A^\times \times S^\times \to \mathbb{R}$ is suitable for LTL learning if*

$$r^\times((s,q,n),a^\times,(s',q',n')) = \begin{cases} R_n & \text{if } q' \in \mathcal{F}; \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

where $R_n \in (0,U]$ are constants for $n \in [0..K]$ and $U \in (0,1]$ is an upper bound on the rewards. The rewards are non-zero only for accepting automaton states, and depend on the value of the K counter.

Then, given a discount factor $\gamma \in (0,1]$, we define the product discount function $\gamma^\times : S^\times \to (0,1]$ as

$$\gamma^\times(s_j^\times) = \begin{cases} 1 - r_j^\times & \text{if } r_j^\times > 0; \\ \gamma & \text{otherwise,} \end{cases}$$

and the expected discounted reward following policy π starting at $s^\times$ and time step t is

$$G_t^\pi(s^\times) = \mathbb{E}_\pi[\sum_{i=t}^{\infty}(\prod_{j=t}^{i-1}\gamma^\times(s_j^\times)) \cdot r^\times(s_i^\times, a_i^\times, s_{i+1}^\times) \mid s_t^\times = s^\times]. \tag{8}$$

Optimality Guarantees

Theorem: Given an LTL formula and a product MDP, there exists a discount factor and a reward upper bound such that the policy maximises the discounted reward also maximises the probability of satisfying the LTL formula.

Proof sketch: We need to make sure the amount of reward received by paths that eventually doesn't satisfy the LTL specification is lower than the reward received by paths that satisfies the LTL specification after discounting.

Efficiency Techniques

Collapsing the Q function

To combat the state-space expansion caused by taking product with the K counter, we only define the Q-function on the environment MDP and Automaton. The Q value is updated for rewards received by all K counter values.

Counterfactual Imagining

Since the automaton is given, at every current state in the environment MDP, imagine the agent is at different automaton states while taking the same action. These imaginations can be used in conjunction with any off-policy RL algorithm and do not affect optimality.

IJCAI/2023 MACAO

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF OXFORD

Experiments Setup

- We adopt model checker PRISM[5] to evaluate the actual probability of satisfaction the LTL formula for direct evaluation and better comparison between methods.
- Comparing against three SOTA methods of learning LTL specifications in RL
- All experiments use rewards that increase linearly with the number of accepting states visited
- Optimistic initialization of Q value to (2*max reward) for all feasible state-action pairs
- All experiments use learning rate 0.1, epsilon 0.1 and discount factor 0.99
- All experiments are run 100 times, the average and half standard deviation is plotted

[5] Marta Kwiatkowska et al. PRISM 4.0: Verification of Probabilistic Real-time Systems. CAV 2011

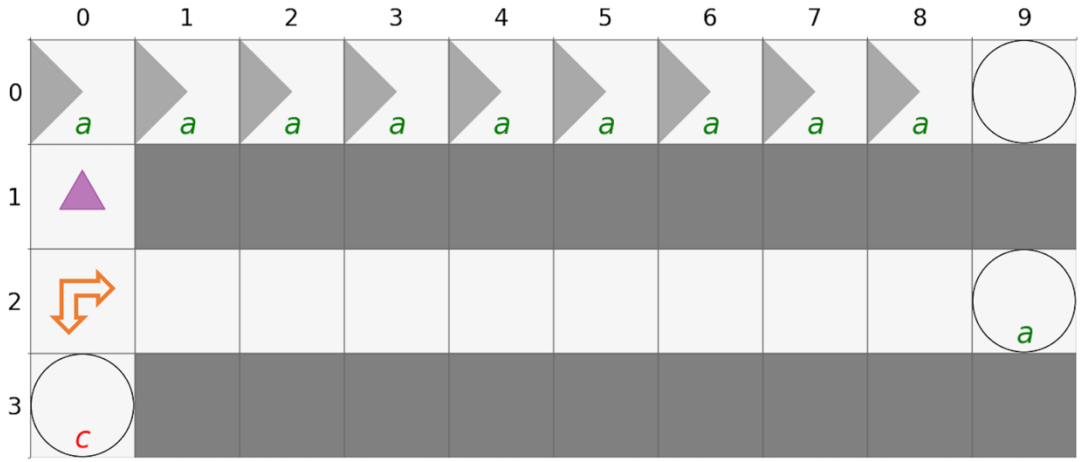
IJCAI/2023 MACAO

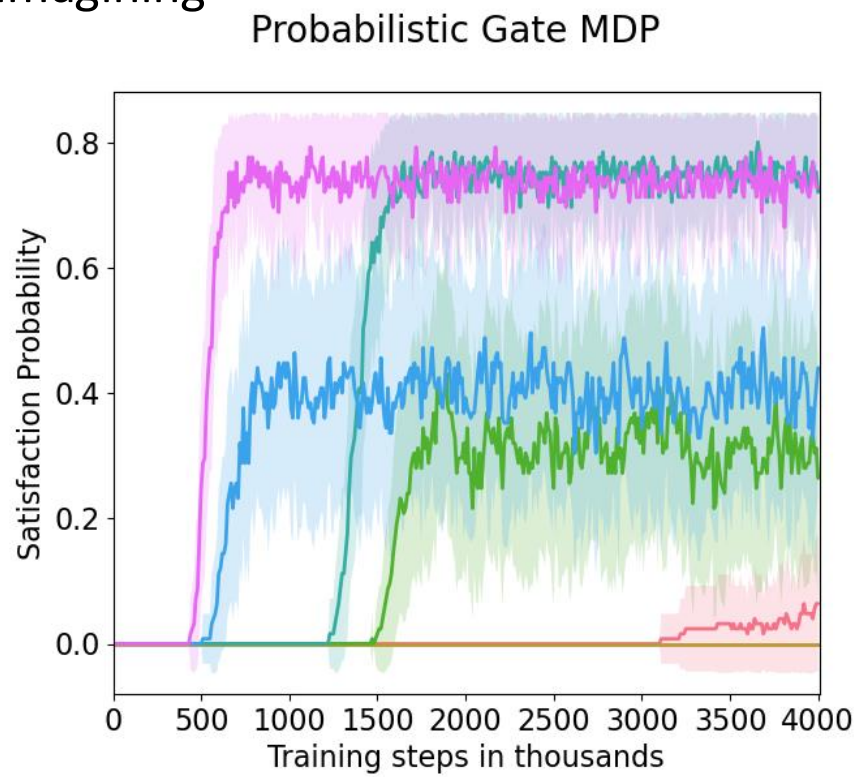
DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF OXFORD

Experimental Results

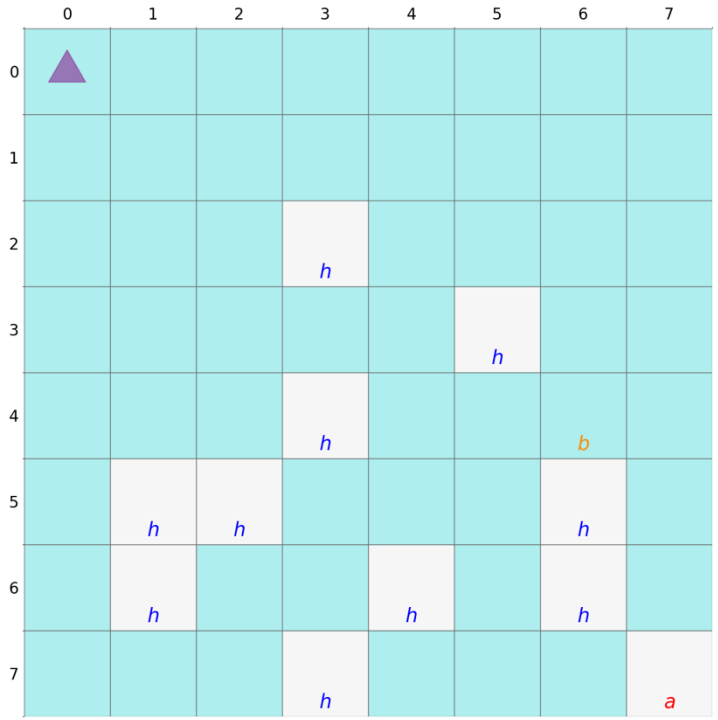
- KC represents using K counter reward with K=10
- CF represents setting K=0 and applying counterfactual imagining
- CF+KC represents setting K=10 and applying CF



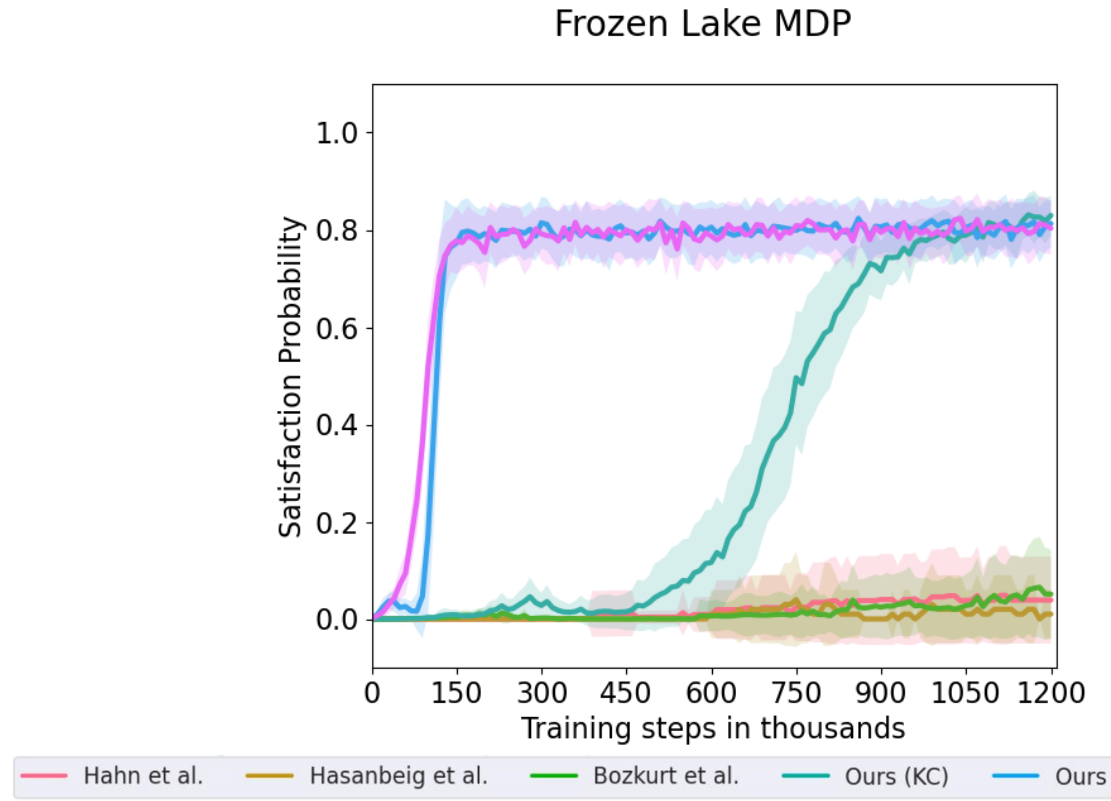


Frozen Lake OpenAI Gym

(G F a | G F b) & (G !h)



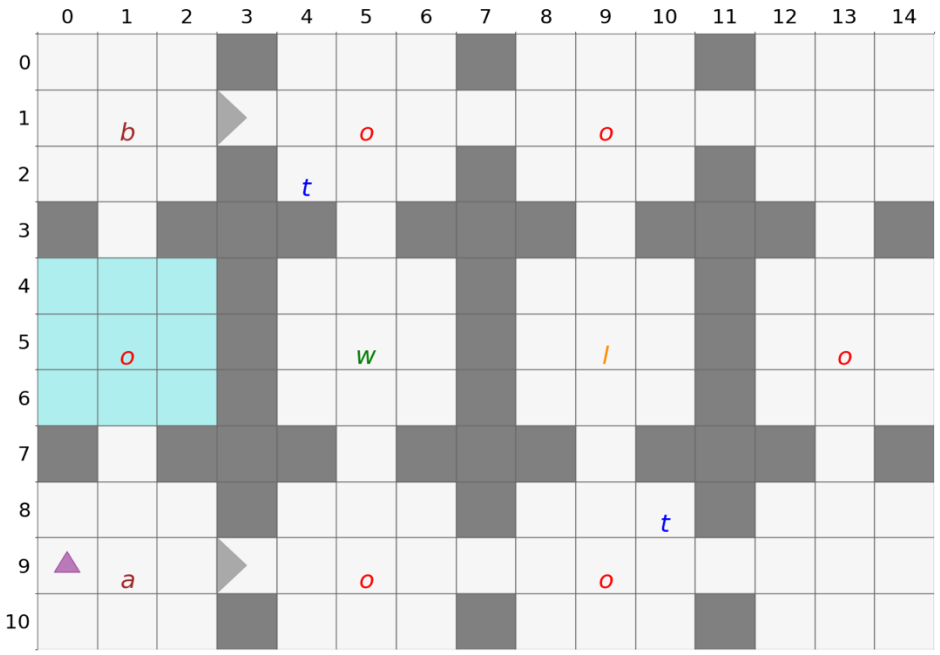
Adopted from Frozen Lake 8*8 environment OpenAI Gym
Blue represent frozen lake, where the agent has 1/3 probability of moving to the intended direction and 1/3 each of going sideways



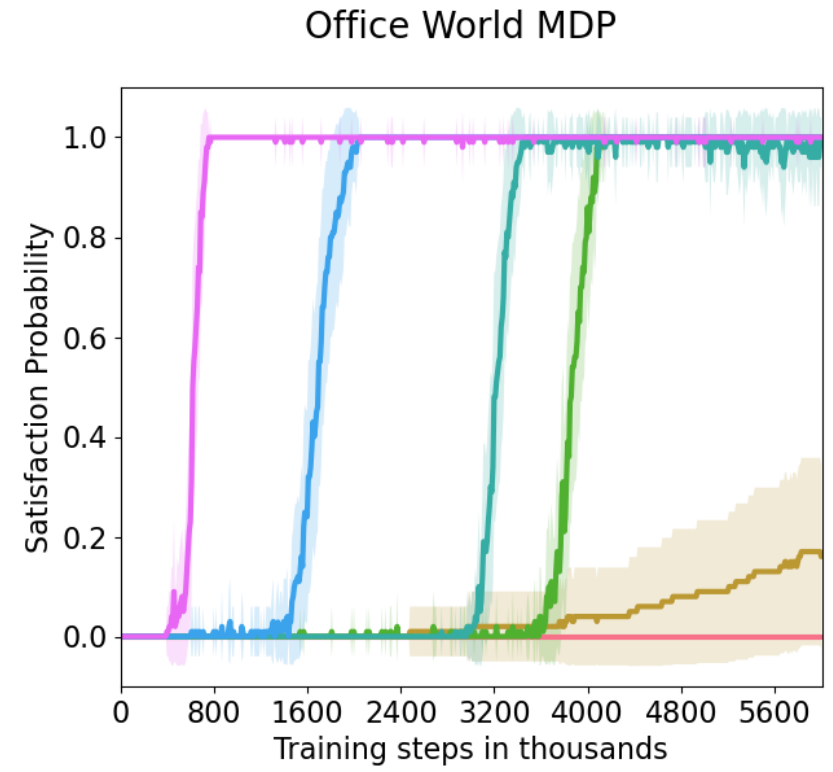
The Office Environment

- We set K=5 for this task

(F (l & X (G F t & G F w))|(G F a & G F b) & (G !o)



The office environment adopted from the Reward Machine paper[3]
I added frozen surfaces and one directional gate



[3] Rodrigo Toro Icarte et al. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. Journal of Artificial Intelligence Research, 2022

IJCAI/2023 MACAO

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF OXFORD

Thank you for listening!

Q&A

daqian.shao@cs.ox.ac.uk