Hierarchy-Driven Exploration for Reinforcement Learning

Evan Zheran Liu [1] **Ramtin Keramati** [2] **Sudarshan Seshadri** [1] **Kelvin Guu** [3] **Panupong Pasupat** [1]
Emma Brunskill [1] **Percy Liang** [1 3]

Abstract

Strategic exploration methods in tabular reinforcement learning provide guarantees for learning near-optimal policies, but require visiting almost all state-action pairs, which is intractable for large state spaces. However, in many cases, the agent receives reward only for transitions that saliently affect the state, and thus only needs to explore these transitions. We propose an exploration algorithm motivated by tabular strategic exploration methods which explores only these transitions in an abstract Markov Decision Process, featuring a small abstract state space and learned hierarchical macro-actions. For approaches that do not use demonstrations, our algorithm achieves state-of-the-art results in MONTEZUMA'S REVENGE, one of the most challenging games in the Arcade Learning Environment.

1. Introduction

Deep reinforcement learning (RL) algorithms achieve impressive, often superhuman, performance in game playing (Mnih et al., 2015) and robotics (Levine et al., 2016). However, they typically employ simple inefficient exploration methods such as ϵ-greedy, which can take exponential time to find reward. Consequently, these algorithms completely fail on environments with extremely sparse rewards, such as MONTEZUMA'S REVENGE (Bellemare et al., 2013), where ϵ-greedy and similar exploration methods achieve no reward even after millions of frames of training, whereas humans score thousands of points (Hester et al., 2017).

In contrast, tabular exploration algorithms such as R-MAX (Brafman & Tennenholtz, 2002) and MBIE-EB (Strehl & Littman, 2008) achieve provably efficient exploration, guaranteeing near-optimal policies in a bounded number of samples. However, these algorithms do not directly apply to large state Markov Decision Processes (MDPs), because they rely on visiting almost every state-action pair. Recent extensions of tabular count-based exploration methods to large state MDPs (Bellemare et al., 2016; Ostrovski et al., 2017) achieve significant improvements for sparse-reward tasks. However, these approaches can suffer from long-term credit assignment and still significantly trail human performance. We also draw motivation from tabular exploration algorithms, but propose a different exploration strategy, guided by the observation below.

Tabular setting exploration algorithms require visiting almost every state-action pair because they implicitly assume that reward can be high at any state-action pair. However, this assumption generally does not hold for tasks with large state-spaces such as Atari games. Instead, reward is typically given when some salient part of the state changes (e.g. the agent reaches the next level or the agent obtains a new item), not for trivial changes in the state (e.g. a pixel flashes in the corner). This implies that efficient exploration methods in the non-tabular setting need not explore *all* state-action pairs, only those that lead to salient changes in the state.

To explore just these state-action pairs, instead of directly exploring the high-dimensional state space with the provided low-level actions, we propose to explore over the *abstract states* that capture the salient part of the state (Figure 1a). Notably, a single low-level action usually changes minute details of the state without altering the abstract state. Since we desire to explore the transitions between abstract states (the transitions that saliently change the state), we learn high-level *macro-actions* composed of many low-level actions to transition between abstract states. Effectively, we operate in a much smaller abstract MDP, which summarizes the original MDP via an abstraction function.

We evaluate our approach on MONTEZUMA'S REVENGE,[1] one of the most challenging exploration games in the Arcade Learning Environment (Bellemare et al., 2013), and

[*]Equal contribution [1]Department of Computer Science, Stanford University, Stanford, USA [2]Institute of Computational and Mathematical Engineering, Stanford University, Stanford, USA [3]Department of Statistics, Stanford University, Stanford, USA. Correspondence to: Evan Zheran Liu <evanliu@cs.stanford.edu>.

Published at the Exploration in Reinforcement Learning Workshop at the 35^{th} International Conference on Machine Learning, Stockholm, Sweden.

[1]Video highlights at https://sites.google.com/view/hierarchy-driven-exploration

achieve state-of-the-art performance for approaches without demonstrations. We detail theoretical guarantees for efficient exploration in an extended version of this paper.

2. Related Work

Exploration in Tabular Reinforcement Learning Exploration in tabular settings is well-understood through the frameworks of optimism in the face of uncertainty (OFU) (Brafman & Tennenholtz, 2002; Strehl et al., 2009) and posterior sampling (Osband et al., 2013; Osband & Van Roy, 2016). OFU methods such as R-MAX (Brafman & Tennenholtz, 2002), MBIE (Strehl & Littman, 2005) and UCRL (Jaksch et al., 2010) efficiently explore by providing bonuses for exploring regions of uncertainty and provably yield near-optimal policies in time polynomial in the size of the MDP. Despite recent progress that has yielded much stronger bounds(Azar et al., 2017; Dann et al., 2017), these methods don't scale well to the deep RL setting, where the size of the MDP can be enormous.

Exploration in Deep Reinforcement Learning Recent works (Bellemare et al., 2016; Tang et al., 2017; Ostrovski et al., 2017) generalize optimism exploration approaches from the tabular setting to the function approximation setting. The exploration bonuses provide substantial improvements over naive exploration methods, especially in sparse reward environments. However, these methods can suffer from long-term credit assignment, which we propose to address with hierarchy. Other exploration methods (Osband et al., 2016; Azizzadenesheli et al., 2018; Fortunato et al., 2017) extend posterior sampling to deep RL, but do not show substantial improvements in hard exploration environments.

Learning from Demonstrations Imitation learning methods (Abbeel & Ng, 2004) attempt to solve sparse reward tasks by leveraging prior knowledge in the form of demonstrations. Recent works leveraging demonstrations (Aytar et al., 2018; Hester et al., 2017; Pohlen et al., 2018) achieve impressive results on challenging domains such as MONTEZUMA'S REVENGE. However, they require extensive prior knowledge as expert demonstrations for the exact task at hand, which can be expensive or impossible to collect. Additionally, these methods do not directly address the exploration problem, as the agent follows the demonstrated trajectory instead of exploring for reward.

Hierarchical Reinforcement Learning Our work also draws from hierarchical reinforcement learning (HRL) ideas of state abstraction (Li et al.; Singh et al., 1995), action abstraction (Sutton, 1995; Sutton et al., 1999), and decomposing tasks into subgoals (Dietterich, 2000). While HRL has received a lot of attention over the years, learning the appropriate abstractions remains an open challenge. For example, the Option-Critic architecture (Bacon et al., 2017) provides an algorithm for automatically learning high-level actions from scratch. However, the algorithm often degenerates to trivial solutions. Feudal networks (Vezhnevets et al., 2017) attempt to sidestep this issue by formulating sub-goals in a latent space. Our work generally differs from prior work on hierarchy with our focus on leveraging hierarchy for strategic exploration by systematically producing subgoals to explore different parts of the state space.

3. Setup

We assume an episodic finite-horizon MDP, $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, R \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P}$ the transition function and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function. The goal of the agent is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the total expected reward $\mathbb{E}_\pi[\sum_{t=0}^{T} R(s_t, a_t)]$. To distinguish from our learned macro-actions, we refer to actions in the MDP as concrete actions.

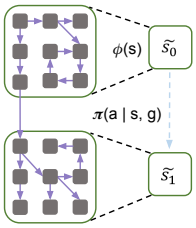
We further assume existence of an abstraction function $\phi : \mathcal{S} \to \tilde{\mathcal{S}}$, which captures the salient parts of concrete states $s \in \mathcal{S}$ as abstract states $\tilde{s} \in \tilde{\mathcal{S}}$. We assume reward is 0 for all transitions within the same abstract state and that all paths to the same abstract state yield approximately the same reward. This can be trivially achieved by augmenting the abstract state with reward. In this work we consider pre-specified abstraction functions, but future work could learn the abstraction.

4. Approach

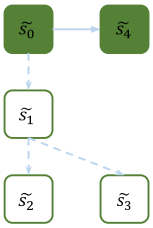
Instead of directly exploring in the original MDP, we explore in a small abstract MDP which consists of abstract states and learn macro-actions that transition between the abstract states. This enables us to ignore exploring trivial transitions and focus our exploration on transitions that saliently change the state and may contain reward.

To explore in the abstract MDP, we maintain the *feasible set*: the set of abstract states that can be reached from the starting abstract states with learned macro-actions, which initially only includes the starting abstract states. When the feasible set contains all reachable abstract states, we can recover a near-optimal policy on the abstract MDP by navigating to the game-winning state.

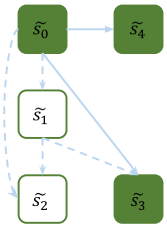
Consequently, our main objective is to add all reachable abstract states to the feasible set. We achieve this by (1) discovering potential transitions to new abstract states (Section 4.1) and (2) learning macro-actions to perform these transitions (Section 4.2). To enable planning, we maintain estimates of the transition dynamics of the learned macro-actions transitioning between abstract states, and the reward dynamics for transition between two abstract states. Finally,



(a) Each abstract state consists of many concrete states. $\tilde{s_1}$ is a degree-1 neighbor of $\tilde{s_0}$ because there is a transition from a concrete state in $\tilde{s_0}$ to a concrete state in $\tilde{s_1}$.



(b) The agent attempts to expand the feasible set by learning a macro-action for the transition $(\tilde{s_0}, \tilde{s_1})$.



(c) Degree-i transition $(\tilde{s_0}, \tilde{s_1})$ could not be learned, so $\tilde{s_2}$ and $\tilde{s_3}$ were added as degree-$(i+j)$ neighbors of $\tilde{s_0}$ (assuming $\tilde{s_2}$ and $\tilde{s_3}$ are degree-j neighbors of $\tilde{s_1}$.

Figure 1. Green nodes represent abstract states. Shaded green nodes are in the feasible set. Dotted arrows represent abstract state neighbors. Solid arrows represent a learned macro-action.

after computing a near-optimal policy on the abstract MDP, we then map the abstract MDP policy to a policy on the original MDP (Section 4.3).

4.1. Discovering Transitions

To extend the feasible set, we must first discover candidate transitions to new abstract states to learn. To do this, for each observed abstract state $\tilde{s}$, we maintain a count of the number of times we have visited that abstract state: $n(\tilde{s})$, and a set of degree-n neighboring abtract states: $\text{neighbors}_n(\tilde{s})$, initialized as empty, for each $n = 1, 2, \ldots, d_{max}$.

We learn macro-actions to transition between abstract states and their neighbors, extending the feasible set. We assume that we discover every immediate neighbor of an abstract state $\tilde{s}$ after visiting it enough times ($n(\tilde{s}) \geq T$ for some threshold T). We refer to such abstract states as *exhausted*, and abstract states which have not been visited enough times as *active*. In our experiments, we use $T = 100$.

Degree-1 neighbors. If an active abstract state in the feasible set exists, we plan to reach this active abstract state from the current abstract state using our transition dynamics model over the abstract states and learned macro-actions, replanning whenever the current plan fails. Upon reaching the active abstract state, we take random actions for T_e timesteps (in our experiments, we use $T_e = 5$), to discover new nearby abstract states. During these T_e timesteps, for each observed transition $(\tilde{s}, \tilde{s}')$, we add $\tilde{s}'$ as a degree-1 neighbor of $\tilde{s}$, illustrated in Figure 1b.

Higher-degree neighbors. Sometimes, since the abstract state abstracts away certain details of the state, it may be preferable to skip learning transitions to immediate degree-1 neighbors by directly learning transitions to farther away abstract states. For example, the agent can successfully learn to transition from one side of a gap to the other. But learning to transition from one side of the gap to the middle of the gap can cause the agent navigate to the middle of the gap and immediately fall to its death afterward. To handle these cases, if we fail to learn a transition $(\tilde{s}, \tilde{s}')$ after many attempts, we consider learning longer transitions by adding the neighbors of $\tilde{s}'$ as higher-degree neighbors of $\tilde{s}$. Formally, if $\tilde{s}'$ is a degree-i neighbor of $\tilde{s}$ and $\tilde{s}''$ is a degree-j neighbor of $\tilde{s}''$, we add $\tilde{s}''$ as a degree-$(i+j)$ neighbor of $\tilde{s}$ (Figure 1c). We only add neighbors up to a maximum degree of d_{max}, which should be large enough to handle the longest obstacle. In our experiments, we use $d_{max} = 15$.

4.2. Macro-Action Learning

Once we have discovered neighboring transitions, we then learn macro-actions for these transitions to extend the feasible set. We place all learned macro-actions in the *macro-action repository*, and reuse these macro-actions instead of learning new macro-actions whenever possible.

Learning macro-actions for the neighbor transitions consist of three stages: 1) First, we compute a priority for each transition and select the transition with the highest priority 2) Then, we attempt to reuse macro-actions from the macro-action repository to perform the transition. 3) Finally, if no macro-action from the repository can perform the transition, we learn a new macro-action. We begin with a description of the representation of macro-actions and then describe the three stages in order.

Macro-action representation. We represent a macro-action $\pi_M(a|s, g)$ for the transition $(\tilde{s}, \tilde{s}')$ as a sub-policy, which conditions on the concrete state s and goal $g = \tilde{s}' - \phi(s)$ to produce a concrete action a. At each timestep, the macro-action receives sparse binary reward equal to 1 if $\phi(s) = \tilde{s}$ and 0 otherwise. For each transition $(\tilde{s}, \tilde{s}')$ and macro-action π_M, we maintain an estimate of the reward for successfully traversing the transition $R(\tilde{s}, \tilde{s}')$ (averaged over all observations of the transition), the transition dynamics $P(\tilde{s}'|\pi_M, \tilde{s})$ (empirically computed as the number

of successes divided by the number of attempts), and the number of training attempts $n(\tilde{s}, \tilde{s}', \pi_M)$. A macro-action episode consists of executing the macro-action for a fixed T_M timesteps times the degree of the transition, starting from the beginning of the transition $\tilde{s}$. Motivated by the idea of controlling the environment from the intrinsic motivation literature (Barto, 2013; Mohamed & Rezende, 2015; Bellemare et al., 2016), we consider an episode a success in the computation of the transition dynamics $P(\tilde{s}'|\pi_M, \tilde{s})$ if the macro-action stays in the goal abstract state $\tilde{s}'$ for at least T_{hold} timesteps. Failing to hold $\tilde{s}'$ for at least T_{hold} timesteps indicates a lack of control (e.g. falling down a gap). We use $T_{hold} = 3$ in our experiments.

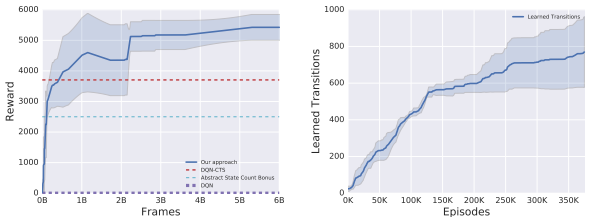
Prioritizing transitions. To select the next transition to learn, we maintain a priority queue of all of the discovered neighboring transitions. The priority of a transition $(\tilde{s}, \tilde{s}')$ is a linear combination of: 1) The reward of the transition $R(\tilde{s}', \tilde{s})$ plus the reward of the highest reward path from the starting abstract states to $\tilde{s}$, according to our model estimates, prioritizing higher rewards. 2) The number of training attempts so far $n(\tilde{s}, \tilde{s}', \pi_A)$, prioritizing fewer training attempts. 3) The length of the shortest path from the starting abstract states to $\tilde{s}$, prioritizing longer paths. We select the transition with the greatest priority.

Reusing the macro-action repository. Once we have selected a transition $(\tilde{s}, \tilde{s}')$ to learn, we repeatedly navigate to the beginning of the transition $\tilde{s}$ via planning with our transition dynamics model, replanning as necessary, and evaluate each macro-action in the repository on M episodes (we use $M = 30$). If the empirical success rate $P(\tilde{s}'|\pi_M, \tilde{s})$ exceeds a threshold p for one of the macro-actions in the repository, we associate π_M with the transition and add $\tilde{s}'$ to the feasible set. In our experiments, we use $p = 0.9$.

Training a new macro-action. If none of the macro-actions in the repository successfully perform the transition with high probability, we begin training a new macro-action. To train a new macro-action for a transition $(\tilde{s}, \tilde{s}')$, we again repeatedly navigate to the beginning abstract state $\tilde{s}$. Then, we alternate between rolling out macro-action episodes and updating the macro-action. If at any point, the new macro-action's success rate exceeds the success threshold p, we freeze the macro-action's parameters and add it to the macro-action repository.

4.3. Recovering an Original MDP Policy

Once we've learned a near-optimal policy over the abstract MDP, we wish to recover a near-optimal policy over the original MDP. Since the macro-actions in the abstract MDP are implemented as sub-policies that produce concrete actions in the original MDP, we have the following natural mapping from abstract MDP policies to original MDP policies. Given an abstract MDP policy $\pi_{abstract}$, and a concrete state s, executing $\pi_{abstract}(\phi(s))$ produces a macro-action π_A. Executing $\pi_A(s)$ then produces a concrete action in the original MDP.



(a) (b)

Figure 2. (a) Training curves for our approach on the hard exploration game MONTEZUMA'S REVENGE compared with the asymptotic performance of DQN, DQN-CTS, and Abstract State Count Bonus. (b) Whereas other approaches plateau, our approach continues to learn more transitions in the Abstract MDP.

5. Experiments

We present our preliminary asymptotic performance results on MONTEZUMA'S REVENGE, one of the most challenging games in the Arcade Learning Environment (Bellemare et al., 2013) below, with additional results in Appendix B. We describe full experimental details in Appendix A.

5.1. Asymptotic Performance

We compare the asymptotic performance of our approach with the best approach that does not leverage demonstrations, DQN-CTS (Ostrovski et al., 2017). We note that two works that incorporate demonstrations (Pohlen et al., 2018; Aytar et al., 2018) achieve higher results, but we do not compare with them because using demonstrations solves a different problem. In order to evaluate the contribution of the abstraction function ϕ, we additionally compare against Abstract State Count Bonus, a modification of the source code of (Tang et al., 2017), which maintains visit counts for each abstract state and provides exploration bonuses inversely proportional to the square root of the visit counts. We include the performance of DQN (Mnih et al., 2015) for reference. We report results from our approach averaged over 4 seeds up to 6 billion frames in Figure 2a.

Our approach achieves state-of-the-art results for approaches that do not use demonstrations. We report on significantly more frames than previous works, but our approach surpasses the previous state-of-the-art in under 1B frames. Importantly, our approach continues to learn even after billions of frames (Figure 2b), whereas other approaches completely plateau. The performance of Abstract State Count Bonus suggests that our improved performance is not just due to the abstraction function.

Acknowledgements This work was supported by a Google Research Award and a SAIL-Toyota seed grant.

References

Abbeel, Pieter and Ng, Andrew Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1. ACM, 2004.

Aytar, Yusuf, Pfaff, Tobias, Budden, David, Paine, Tom Le, Wang, Ziyu, and de Freitas, Nando. Playing hard exploration games by watching youtube. *arXiv preprint arXiv:1805.11592*, 2018.

Azar, Mohammad Gheshlaghi, Osband, Ian, and Munos, Rémi. Minimax regret bounds for reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2017.

Azizzadenesheli, Kamyar, Brunskill, Emma, and Anandkumar, Animashree. Efficient exploration through bayesian deep q-networks. *arXiv preprint arXiv:1802.04412*, 2018.

Bacon, Pierre-Luc, Harb, Jean, and Precup, Doina. The option-critic architecture. In *AAAI*, pp. 1726–1734, 2017.

Barto, Andrew G. Intrinsic motivation and reinforcement learning. In *Intrinsically motivated learning in natural and artificial systems*, pp. 17–47. Springer, 2013.

Bellemare, Marc, Srinivasan, Sriram, Ostrovski, Georg, Schaul, Tom, Saxton, David, and Munos, Remi. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 1471–1479, 2016.

Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Brafman, Ronen I and Tennenholtz, Moshe. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3 (Oct):213–231, 2002.

Dann, Christoph, Lattimore, Tor, and Brunskill, Emma. Unifying pac and regret: Uniform pac bounds for episodic reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 5713–5723, 2017.

Dietterich, Thomas G. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Fortunato, Meire, Azar, Mohammad Gheshlaghi, Piot, Bilal, Menick, Jacob, Osband, Ian, Graves, Alex, Mnih, Vlad, Munos, Remi, Hassabis, Demis, Pietquin, Olivier, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Hester, Todd, Vecerik, Matej, Pietquin, Olivier, Lanctot, Marc, Schaul, Tom, Piot, Bilal, Horgan, Dan, Quan, John, Sendonaris, Andrew, Dulac-Arnold, Gabriel, et al. Deep q-learning from demonstrations. *arXiv preprint arXiv:1704.03732*, 2017.

Jaksch, Thomas, Ortner, Ronald, and Auer, Peter. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010.

Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Levine, Sergey, Finn, Chelsea, Darrell, Trevor, and Abbeel, Pieter. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Li, Lihong, Walsh, Thomas J, and Littman, Michael L. Towards a unified theory of state abstraction for mdps.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Mohamed, Shakir and Rezende, Danilo Jimenez. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pp. 2125–2133, 2015.

Osband, Ian and Van Roy, Benjamin. Why is posterior sampling better than optimism for reinforcement learning. *arXiv preprint arXiv:1607.00215*, 2016.

Osband, Ian, Russo, Daniel, and Van Roy, Benjamin. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, pp. 3003–3011, 2013.

Osband, Ian, Blundell, Charles, Pritzel, Alexander, and Van Roy, Benjamin. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pp. 4026–4034, 2016.

Ostrovski, Georg, Bellemare, Marc G, Oord, Aaron van den, and Munos, Rémi. Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*, 2017.

Pohlen, Tobias, Piot, Bilal, Hester, Todd, Azar, Mohammad Gheshlaghi, Horgan, Dan, Budden, David, Barth-Maron, Gabriel, van Hasselt, Hado, Quan, John, Večerík,

Mel, et al. Observe and look further: Achieving consistent performance on atari. *arXiv preprint arXiv:1805.11593*, 2018.

Singh, Satinder P, Jaakkola, Tommi, and Jordan, Michael I. Reinforcement learning with soft state aggregation. In *Advances in neural information processing systems*, pp. 361–368, 1995.

Strehl, Alexander L and Littman, Michael L. A theoretical analysis of model-based interval estimation. In *Proceedings of the 22nd international conference on Machine learning*, pp. 856–863. ACM, 2005.

Strehl, Alexander L and Littman, Michael L. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74 (8):1309–1331, 2008.

Strehl, Alexander L, Li, Lihong, and Littman, Michael L. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(Nov):2413–2444, 2009.

Sutton, Richard S. Td models: Modeling the world at a mixture of time scales. In *Machine Learning Proceedings 1995*, pp. 531–539. Elsevier, 1995.

Sutton, Richard S, Precup, Doina, and Singh, Satinder. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Tang, Haoran, Houthooft, Rein, Foote, Davis, Stooke, Adam, Chen, OpenAI Xi, Duan, Yan, Schulman, John, DeTurck, Filip, and Abbeel, Pieter. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2750–2759, 2017.

Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pp. 2094–2100, 2016.

Vezhnevets, Alexander Sasha, Osindero, Simon, Schaul, Tom, Heess, Nicolas, Jaderberg, Max, Silver, David, and Kavukcuoglu, Koray. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

Wang, Ziyu, Schaul, Tom, Hessel, Matteo, Van Hasselt, Hado, Lanctot, Marc, and De Freitas, Nando. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

A. Experiment Details

We use a 5-tuple of the (agent's x-coordinate, agent's y-coordinate, room number, agent's inventory, objects in the current room) as the abstract state representation. These correspond to the RAM observations at locations 42, 43, 3, 65, and 66 respectively. We represent our macro-actions as combining deep double q-networks (Van Hasselt et al., 2016) with dueling networks (Wang et al., 2015), optimized with Adam (Kingma & Ba, 2014). Additionally, we train the macro-actions with count-based reward bonuses (Bellemare et al., 2016; Tang et al., 2017) inversely proportional to the square root of the current abstract state visitation count. The macro-actions receive the concrete state concatenated with the goal as input. Following (Mnih et al., 2015), we represent the concrete state inputs as the past four pixel frames downsamples, cropped to size 84 by 84, and converted to grayscale. Additionally, the macro-actions receive a negative reward penalty for losing lives. We employ standard frame-skipping so that each concrete action is repeated four times. For simplicity, we do not employ any forms of stochasticity.

Many simple tasks, such as moving to the left, do not even require pixel inputs to learn. To capitalize on this, we first try to learn macro-actions that ignore the pixel inputs and only condition on the goal. If the pixel-blind macro-action fails to learn, we then train a macro-action that conditions on the pixel inputs.

B. Additional Results

B.1. Reusing the Macro-Action Repository

Figure 3 and Figure 5 illustrate the reuse of macro-actions from the macro-action repository. Most learned transitions reuse common macro-actions (e.g. moving up or down a ladder), while there is a long tail of uncommon macro-actions (e.g. jumping over a specific type of monster). Figure 4 depicts two common macro-actions. The agent successfully reuses the same macro-action for moving up a ladder and up a rope.

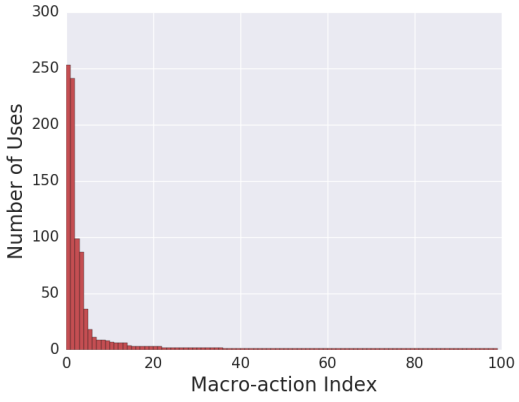


Figure 3. Number of reuses of each macro-action in the macro-action repository.

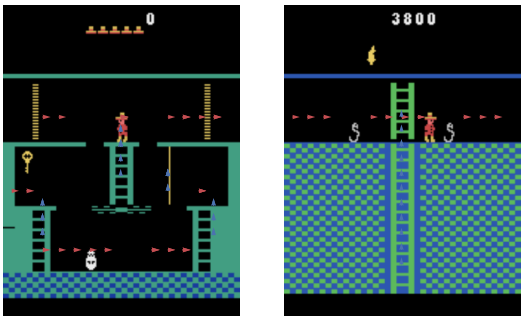


Figure 4. The usages of two macro-actions in two different rooms. The same blue macro-action moves up ladders and ropes. The same red macro-action moves right in many abstract states. The moving right macro-action notably does not apply when there is an obstacle in its way.

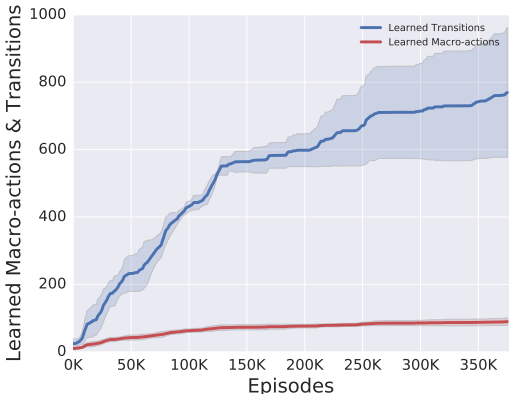


Figure 5. Number of learned transitions in abstract MDP compared to the number of learned macro-actions.