Interpretable Transfer for Reinforcement Learning based on Object Similarities

Ramya Ramakrishnan, Karthik Narasimhan, Julie Shah
Massachusetts Institute of Technology
ramyaram@mit.edu, karthikn@mit.edu, julie_a_shah@csail.mit.edu

Abstract

We present Object-Based Transfer (OBT), a cognitively-inspired transfer technique for reinforcement learning that aims to provide transparency about an agent's generalization process. For each new type of object the agent sees, our algorithm adapts existing knowledge by learning to map new object types to previously seen object types while simultaneously refining its policies. We learn both the mapping and new policies in an iterative fashion. We perform computational experiments using two game worlds and demonstrate that OBT is able to learn faster on the target domain than learning from scratch. Additionally, the algorithm provides similarities between objects, lending transparency to the agent's learning process.

1 Introduction

A key challenge in developing intelligent agents that can interact in the real world is that agents must be able to generalize to new situations and adapt to people. A critical component in creating such a system is transfer learning, which involves adapting prior learned knowledge to new tasks. Many current transfer techniques for reinforcement learning (RL) [14; 1] are black boxes that are difficult to understand and cannot directly be used in interactive human-in-the-loop systems.

To develop agents that can interact with people, we need an interpretable representation for the agent that is understandable to people. Prior work [3] has shown that there is a limit on the number of items people can store in their working memories, demonstrating that people learn to focus on a few important features in their environment at any particular instant. Cognitive science research [16] has also shown that object representations are part of the core of human cognition and that infants without any visual experience are able to understand the concept of objects. These studies suggest that instead of representing the world using a large set of high-dimensional features, the agent can use a cognitively-inspired representation based on a small set of important objects in the environment. This has inspired work in RL utilizing object-based representations [4; 2] aimed at speeding up learning in a task.

In this work, we present an approach for performing interpretable Object-Based Transfer (OBT) for reinforcement learning. Our technique uncovers relationships between new and existing objects to speed up learning in a new scenario. Objects of the same class can be treated by the agent in an analogous fashion. The challenge for transfer lies in identifying the optimal mapping between object classes in the source and target tasks to maximize performance on the target task. We simultaneously learn this mapping while updating class-specific policies for the new object classes.

We perform computational experiments between two game domains and compare our algorithm OBT with learning from scratch on the new task. We find that the agent learns more quickly using OBT than learning from scratch and is able to successfully avoid negative transfer. In addition, OBT is able to generate simple explanations for similarities between tasks, based on objects, that can be explained to people.

2 Related Work

In RL problems, transfer learning (TL) [19] is often used to speed up learning in a new target task by using previously learned knowledge learned in one or more source tasks. With the current success of deep learning in RL [12], some works use deep neural networks to facilitate transfer between tasks. In one work [1], a restricted Boltzmann machine (RBM) is used to compute a task similarity measure. Given samples of $< s, a, s' >$ from two MDPs, they train an RBM using the first MDP's data and use the trained model to reconstruct every sample from the second MDP. The reconstruction error is the distance between the original and reconstructed samples from the second MDP, which will be low if the two MDPs are similar. However, this measure of similarity is sensitive to parameters and can take a long time to learn [15]. Another work [14] develops an architecture ADAAPT for transfer between tasks using deep neural networks. While it supports automatic learning of features and representations, it is difficult to provide transparency about the transfer learning process using this approach, and thus it cannot directly be used in interactive settings.

Another work [11] aims to explicitly learn abstract knowledge to transfer between tasks, which can potentially lead to more interpretability. They introduce a technique to learn "options", which are temporally extended actions that can be reused in subsequent tasks. In order to transfer options

across tasks, they consider two representations. *Problem-space* is a Markovian representation and is specific to a particular task, and *agent-space* is non-Markovian and can be shared across tasks. Some works use alternative representations in the RL framework that can more easily be understood, such as Dynamic Bayesian Networks (DBNs) [8] and decision trees [18].

Cognitive science literature [3; 16] indicates that people only keep track of a few items in their short term memory and that objects are important in people's interactions with the environment. Prior work in RL has used representations based on objects. For example, [4] develops a framework in which states are represented using collections of objects and objects belong to specific classes. This work requires a person to manually specify relations about how objects interact with each other. Another work [2] similarly represents the state using objects but automatically learns policies for each object class without requiring hand-designed information. Both of these works focus on single task learning rather than transfer as we do, but their object-based representations are promising base learning algorithms, as objects are critical to human cognition and thus may promote transparency [16].

Other works generate explanations for MDP decisions. For example, [6] computes the most relevant variable in an MDP by selecting the variable with the highest impact on expected utility given the current state and action, which is then used to generate a verbal explanation. Another work [9] generates a Minimum Sufficient Explanation (MSE) by calculating occupancy frequencies, or the expected number of times the agent reaches each state using a particular policy. These frequencies are then combined with pre-defined templates to provide a mathematically grounded explanation. To generate a more interpretable explanation, one work [5] uses both an MDP-based explainer, which provides information about the predicted future, and a case-based explainer, which provides previous cases similar to the current one.

In addition to interpretability, interaction is critical to human-in-the-loop systems. To develop models that involve interaction, it is important that decisions are made quickly, and feedback is incorporated in real-time. The TAMER framework [10] is one interactive RL work which models a human's reinforcement function. An agent then uses this model to select actions that will result in the most positive human reinforcement. Another interactive technique [7] converts feedback directly into a policy to guide the agent's learning process. These works on improving interpretability and interactivity in MDPs, however, focus on single task learning, while we focus on improving the transparency of the transfer learning process.

3 Background

3.1 Reinforcement Learning

Markov decision processes (MDPs) [13] are often used to represent stochastic tasks. An MDP is a tuple $\{S, A, T, R\}$, where S represents the set of states in the world, A is the set of actions, $T : S \times A \rightarrow \Pi(S)$ is the transition function that specifies the probability of reaching state s' from state s and taking action a, and $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function that specifies the immediate reward received for taking action a in state s and reaching state s'.

Given an MDP for a task, reinforcement learning (RL) [17] can be used to learn to make robust decisions. In the RL framework, the agent receives a state s from the environment, takes an action a, and receives a next state s' and a reward r. The agent continues to take actions until a terminal state is reached, denoting the end of one *episode*. A popular algorithm to learn an optimal policy for the agent is Q-learning [20]. The agent repeatedly executes episodes to learn a Q-value function $Q : S \times A \rightarrow \mathbb{R}$, which represents the expected long-term reward for taking action a from state s. A policy $\pi : S \rightarrow A$ can be obtained by choosing the action with the maximum Q-value for each state.

3.2 Object-Focused Q-learning

Using the standard framework of Q-learning can be time-consuming for large complex problems, and the learned policy is not easily interpretable. An alternative work develops an algorithm, Object-Focused Q-learning (OF-Q) [2], that represents the state as a set of independent objects and decomposes the standard value function $Q(s, a)$ into a set of value functions $\{Q_c\}$, one for each object class $c \in C$ in the task. This decoupling allows for faster learning and provides a grounded interpretation in terms of action policies focused on objects.

An OF MDP is defined as:

$$M_{OF} = \{S, A, \{T_c\}_{c \in C}, \{R_c\}_{c \in C}\},$$

where each state $s \in S$ is a set of objects $s = \{o_a, o_1, ..., o_k\}$ in which the agent object o_a is always present and the other objects can appear or disappear during the course of the task. Each object o is represented by a set of features $\{f_1, ..., f_n\}$, and has a unique id that can be used to track the object at each step, along with an associated object class $c \in C$. We represent the state of an object o_k as a function of the agent object and the object itself $s_{o_k} = g(o_a, o_k)$. Each object class c has a corresponding transition function $T_c : S_o \times A \rightarrow \Pi(S_o)$ and reward function $R_c : S_o \times A \times S_o \rightarrow \mathbb{R}$.

Given an OF MDP, Object-Focused Q-learning (OF-Q) can be used to learn an optimal value function $Q_c(s_o, a)$ for each object class $c \in C$. The optimal policy for the agent is then:

$$\pi(s)_{OF} = \arg\max_{a \in A} \max_{o \in s} Q(s_o, a)$$

4 Object-Based Transfer

In addition to speeding up learning, we hypothesize that the paradigm of OF-Q can be used to transfer policies to new environments. The Q-value function, factorized into object class-specific functions, can be partially or fully adapted to a new task by drawing parallels between objects in the old *source* task and objects in the new *target* task. For instance, if the agent learns to avoid a certain object o_1 which lowers its expected reward in the source task, it can reuse the Q-value function specific to that object's class to avoid a similar object o_2 in the target scenario. The challenge, however, lies in determining the optimal mapping to reuse the value functions while avoiding the problem of negative transfer [19]. In this section, we detail an algorithm to dynamically learn a good mapping using the reward feedback from the target task.

4.1 Problem Statement

We formulate our object-based transfer learning problem as follows: an agent is first given a source Object-Focused MDP M_1 with $|C_1|$ object classes and learns a set of Q-value functions $\{Q_c\}, c \in C_1$ using the OF-Q algorithm. The agent is then given a target MDP M_2 with $|C_2|$ object classes and must quickly learn a set of Q-value functions $\{Q_c\}, c \in C_2$ to operate effectively in the new environment.

The first objective is to learn $\{Q_c\}, c \in C_2$ for M_2 faster than learning from scratch using standard OF-Q. This is already a challenging problem, as the agent must avoid negative transfer, which is when transferring knowledge actually hurts performance in the new task.

Our second objective is to provide intuitive information about similarities between the two tasks. We hypothesize that by representing the state as a set of objects and finding similarities between these objects, humans can better understand the agent's transfer learning process [16].

4.2 Algorithm

Our algorithm, Object-Based Transfer (OBT), uses an iterative approach to simultaneously (1) learn mappings between new object classes in the target task and previously seen object classes in the source task and (2) adapt previously learned value functions for the target task. We alternate between two phases: a *mapping* phase, in which mappings are learned and Q-value functions are kept constant, and a *Q-values* phase, in which value functions are updated and mappings are kept constant.

We define a mapping between the object classes as $\chi : C_2 \rightarrow C_1 \cup \{c_{new}\}$, where c_{new} is a new object class, non-existent in the source task. We wish to uncover the best possible mapping χ that can speed up learning in the target task. To achieve this, we learn an optimal value function over mappings Q^*_χ to maximize the expected per episode reward $\mathcal{R}$:

$$Q^*_\chi(i,j) = \max_\chi \mathrm{E}[\mathcal{R} \mid \chi_i = j] \quad (1)$$

where i is the target object class mapped to source object class j.

The input to OBT is the target MDP M_2; a set of value functions $\{Q\} = \{Q_1, ..., Q_{C_1}, Q_{new}\}$ which include $|C_1|$ previously learned value functions from the source task T_1 as well as a newly initialized value function Q_{new}; the number of epochs E, where each epoch consists of one mapping phase and one Q-values phase; the number of episodes per mapping phase K_χ; the number of episodes per Q-values phase K_Q; and the maximum number of time steps T per episode. The algorithm also uses the following parameters: the learning rate α_χ for the mapping phase; a parameter ϵ_χ for the exploration rate in the mapping phase; a discount factor γ for future rewards; the learning rate α; and the exploration rate ϵ for the Q-values phase.

Figure 1 provides the pseudocode for OBT. We first initialize a value function Q_χ over all possible mappings from target task object classes C_2 to source task object classes C_1. The algorithm then runs E epochs, with the mapping phase running for K_χ episodes and the Q-values phase running for K_Q episodes.

Algorithm: Object-Based Transfer (M_2, $\{Q\}$, E, K_χ, K_Q, T)

1. Initialize $Q_\chi(i,j)$ randomly, $\forall i = 1, ..., C_2, \forall j = 1, ..., C_1 + 1$
2. **for** $epoch = 1$ **to** E
3. **Mapping phase:**
4. **for** $k = 1$ **to** K_χ
5. χ = ϵ_χ-greedy(Q_χ) (Equation 2)
6. Run-Episode(χ, Q_χ, $\{Q\}$, T, false)
7. Update Q_χ (Equation 3)
8. **end for**
9. **Q-values phase:**
10. χ = greedy(Q_χ) (Equation 2)
11. **for** $k = 1$ **to** K_Q
12. Run-Episode(χ, Q_χ, $\{Q\}$, T, true)
13. **end for**
14. **end for**
15. χ = greedy(Q_χ) (Equation 2)
16. **Return** $< \chi, \{Q\} >$

Figure 1: Object-Based Transfer is a transfer technique that simultaneously learns mappings between previously seen object classes and new object classes while updating its policies for the new task.

During the mapping phase (line 3), we select a mapping χ for each episode using an ϵ_χ-greedy approach. With ϵ_χ probability, we choose a random mapping, and with $1 - \epsilon_\chi$ probability, we select a greedy mapping according to the equation below:

$$\chi_i = \arg\max_j Q_\chi(i,j) \quad (2)$$

Given this mapping, we run one episode using the Run-Episode function in Figure 2. During this phase of OBT, we do not update the Q-values, but rather use it purely to explore different mappings and learn values for them. The function returns the accumulated reward $\mathcal{R}$ in each episode, which is then used to update the value function over mappings $Q_\chi, \forall i \in C_2$, using the following equation:

$$Q_\chi(i,\chi_i) = (1 - \alpha_\chi) * Q_\chi(i,\chi_i) + \alpha_\chi * \mathcal{R} \quad (3)$$

After K_χ episodes, we switch to the Q-values phase (line 9). We calculate the best mapping χ using Equation 2 and keep it constant over all K_Q episodes during this phase, while only updating the Q-value functions. At the end of the Q-values phase, one epoch has been completed, and the two phases are again repeated. After running all epochs, we calculate the final mapping χ (line 15) based on the learned Q_χ. OBT returns this final best mapping χ and the updated value functions $\{Q\}$. As we demonstrate in Section 5, this χ can be used to provide some transparency into the agent's transfer learning process.

The Run-Episode function is provided in Figure 2. At each time step, an action a_t is selected (line 3) using an ϵ-greedy approach. For greedy action selection, we use the equation

Algorithm: Run-Episode (χ, Q_χ, $\{Q\}$, T, $updateQ$)

1. Set the initial state s_1 randomly
2. **for** $t = 1$ **to** T
3. a_t = ϵ-greedy(s_t) (Equation 4)
4. Receive reward r_t and next state s_{t+1}
5. if $updateQ$
6. Update $\{Q\}$ (Equation 5)
7. Set $s_t \leftarrow s_{t+1}$
8. **end for**
9. $\mathcal{R} = \sum_{t=1}^{T} r_t$
10. **Return** $< \mathcal{R}, Q_\chi, \{Q\} >$

Figure 2: Run-Episode is a subfunction of Object-Based Transfer that executes one episode of the task (from initial to goal state).

below:

$$a = \arg\max_{a' \in A} \max_{o \in s} Q(s_o, a') \quad (4)$$

In line 4, the reward r_t and the next state s_{t+1} is received from the environment. The reward is used to update the Q-value functions $\{Q\}$, if the $updateQ$ parameter is set to true. We make a Q-value update for each object o in state s using the following equation:

$$Q_c(s_o, a) = (1 - \alpha) Q_c(s_o, a) + \alpha \Big(r + \gamma \max_{a' \in A} Q_c(s'_o, a') \Big) \quad (5)$$

After T steps, the episode is completed. The function returns the total accumulated reward from the episode (line 9), along with the mapping value function Q_χ and the updated Q-value functions $\{Q\}$.

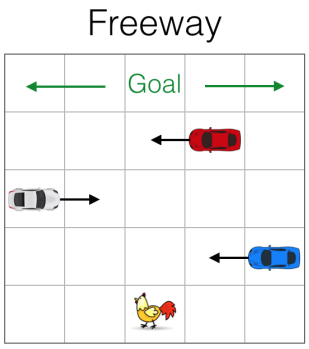
5 Experiments

5.1 Domains

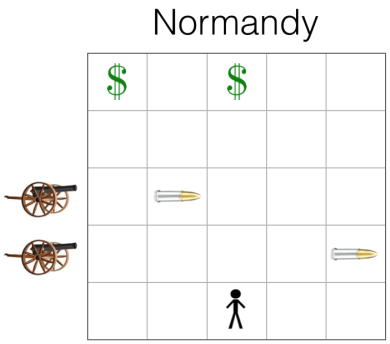
We run experiments using two domains. The source task is a variant of the Freeway Atari game on a 10x10 grid, and the target task is the Normandy domain used in [2], also on a 10x10 grid. Figure 3 shows a simplified 5x5 version of the domains we use.

In the Freeway game, the agent (a chicken) aims to cross the road from the bottom row to the top row while avoiding cars. We have two object classes: cars moving to the right and cars moving to the left, as they have different transition functions. The agent starts randomly in one of the locations on the bottom row, and can move left, right, up, down, or wait. Initially, there are no cars on the screen, and cars appear with 0.5 probability, up to a maximum of four cars. Each car is randomly assigned to a class (either right-moving or left-moving). Once on the screen, cars move one cell to the right or left at each time step. The agent receives -100 reward if it hits a car and +100 if it reaches the top row. The episode ends if the agent successfully crosses the road or if it hits a car.

In the Normandy game, the agent also starts randomly at one of the locations on the bottom row and has the same set of actions as in Freeway. The agent's objective is to collect



(a) Freeway. The agent starts at a random location on the bottom row and must avoid the cars to get to the top row.



(b) Normandy. The agent starts at a random location on the bottom row, must avoid the bullets shot by the cannons, and collect both prizes randomly placed in the top row.

Figure 3: The two domains used in our experiments. We used a 10x10 grid for both games. The agent first learned on the source task Freeway and then used the learned value functions to learn more quickly in the target task Normandy.

two prizes located randomly at two unique locations on the top row. There are also two cannons located on the third and fourth rows of the grid that can shoot bullets towards the right. Bullets move one cell to the right at each time step until it moves off the grid. If no bullet exists in that row, the cannon will shoot a new bullet with probability 0.5. The agent receives +100 reward for each prize it collects and -100 reward if it collides with a bullet. An episode ends if the agent collects both prizes or if the agent collides with a bullet.

5.2 Baselines

We compare our algorithm OBT with Object-Focused Q-learning (OF-Q) from scratch on the target task. In our setup, we represent each object o using two features: the x and y locations of the object on the grid. The state of each object s_o is based on the Euclidean distance between the object's location and the agent's location in the x and y axes.

All algorithms use the following parameters: learning parameter α = 0.001, discount factor γ = 0.99, and exploration

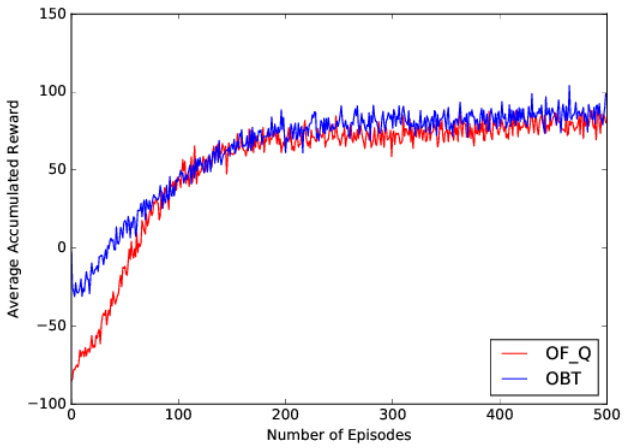


Figure 4: This chart shows the performance of OBT compared to OF-Q from scratch on the target domain.

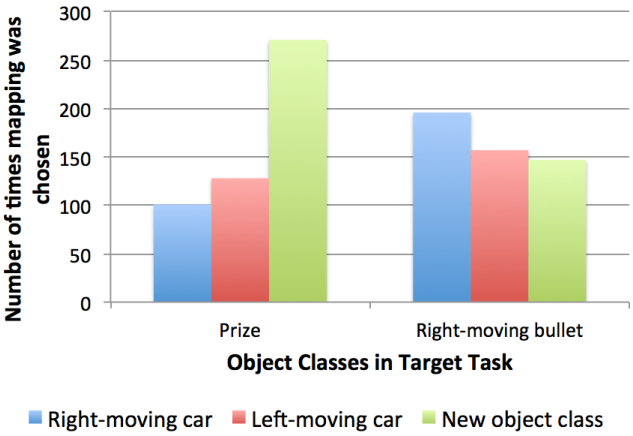


Figure 5: This chart shows the number of times each of the target task classes were mapped to each of the source task classes.

parameter ϵ = 0.05. We also use number of epochs E = 50, number of episodes per mapping phase K_χ = 1, number of episodes per Q-values phase K_Q = 9, number of steps per episode T = 500, and mapping learning rate α_χ = 0.1. The mapping exploration rate ϵ_χ starts at 0.1 and is decreased by 0.001 every time step until it reaches a constant value of 0.01. We execute 500 simulation runs, each for E epochs, and average over all of the runs to get a smoother learning curve.

5.3 Results

Figure 4 shows the performance of OBT and OF-Q from scratch on the target task. We see that early in learning, OBT obtains higher average accumulated reward starting at around -30, compared to about -80 for OF-Q. This improvement shows that OBT is using prior learned knowledge from the source task to achieve a jumpstart in initial performance. Both approaches ultimately converge to the same average accumulated reward. These preliminary results show that on the domains tested, OBT is able to achieve a jumpstart in performance and learn more quickly than learning from scratch.

We also evaluate the transparency of the approach to see if OBT is able to provide some intuition into the agent's transfer learning process. Freeway has two object classes: cars that moved to the left and cars that moved to the right, both resulting in -100 if the agent hits them. Normandy has two object classes: prizes, which provide +100 reward if collected, and right-moving bullets from cannons, which result in -100 reward if hit. When determining mappings between these two tasks, we expect to see the prize class mapped to a new object class, and the right-moving bullet most likely mapped to the right-moving car, but possibly also to the left-moving car.

For each of the 500 simulation runs, we calculate the final greedy mapping χ based on Q_χ at the end of learning (line 15 in Algorithm 1). We plot a bar graph (Figure 5) that shows the number of times each target class is mapped to each of the source classes. We see that the prize object class is mapped to a new object class for the majority of the runs, which is intuitive because the prize is not very similar to either of the car classes from the source task. For the bullet class in Normandy, the right-moving car is the best mapping, followed by the left-moving car and the new object class, which matches our hypothesis as well.

6 Discussion

Preliminary results in this work show that our approach Object-Based Transfer can speed up learning in a new task and provide some information about the agent's transfer learning process on the domains tested. This is an initial step towards developing learning approaches that are inspired by the way people represent knowledge and learn.

However, there are many limitations and areas for future work. To scale up the current framework to larger problems, we can use function approximation techniques, such as neural networks. We plan to investigate whether these approximation methods can allow for more generalization, while still maintaining interpretability.

One limitation in the current approach is that OBT does not differentiate between value functions that provide equally high reward, even when there is an intuitive difference in the classes. This phenomenon can be detrimental in a human interaction scenario as providing unintuitive mappings can lead to confusion. One potential solution is to consider similarities between objects based on other features, not just based on the reward received. Additional features about the objects can help the agent disambiguate between mappings with similar reward.

Another way to learn intuitive mappings while robustly avoiding negative transfer is to integrate human feedback into the approach. The agent can use this to speed up generalization and to personalize its learning to different types of people, which can be especially helpful in a home assistance scenario. In order to assess the interpretability of our method, we also plan to run human subject experiments comparing OBT

to other state-of-the-art transfer learning approaches.

7 Conclusion

In this work, we develop a transfer learning algorithm Object-Based Transfer (OBT), inspired by cognitive science, that can transfer knowledge between tasks and provide similarities between objects. Our algorithm uses an iterative approach to learn the similarity between previously-seen object types and new object types. Results from computational experiments show that OBT is able to transfer knowledge and learn more quickly than learning from scratch. Additionally, it provides simple explanations for how the agent transferred knowledge, by indicating similarity to previous objects.

References

[1] Haitham Bou Ammar, Eric Eaton, Matthew E Taylor, Decebal Constantin Mocanu, Kurt Driessens, Gerhard Weiss, and Karl Tuyls. An automated measure of MDP similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[2] Luis C Cobo, Charles L Isbell, and Andrea L Thomaz. Object focused q-learning for autonomous agents. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1061–1068. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[3] Nelson Cowan. The magical mystery four how is working memory capacity limited, and why? *Current directions in psychological science*, 19(1):51–57, 2010.

[4] Carlos Diuk, Andre Cohen, and Michael L Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 240–247. ACM, 2008.

[5] Thomas Dodson, Nicholas Mattei, Joshua T Guerin, and Judy Goldsmith. An english-language argumentation interface for explanation generation with Markov decision processes in the domain of academic advising. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 3(3):18, 2013.

[6] Francisco Elizalde, Enrique Sucar, Julieta Noguez, and Alberto Reyes. Generating explanations based on Markov decision processes. In *MICAI: Advances in Artificial Intelligence*, pages 51–62. Springer, 2009.

[7] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L. Isbell, and Andrea Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *NIPS*, 2013.

[8] Anders Jonsson and Andrew Barto. Active learning of dynamic Bayesian networks in Markov decision processes. In *Abstraction, Reformulation, and Approximation*, pages 273–284. Springer, 2007.

[9] O Khan, Pascal Poupart, and J Black. Automatically generated explanations for Markov decision processes. *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, pages 144–163, 2011.

[10] W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The TAMER framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16. ACM, 2009.

[11] George Konidaris and Andrew G Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pages 895–900, 2007.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[13] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*, volume 414. John Wiley & Sons, 2009.

[14] Janarthanan Rajendran, P Prasanna, Balaraman Ravindran, and Mitesh M Khapra. Adaapt: A deep architecture for adaptive policy transfer from multiple sources. *arXiv preprint arXiv:1510.02879*, 2015.

[15] Ramya Ramakrishnan. Perturbation training for human-robot teams. Master's thesis, MIT, 2015.

[16] Elizabeth S Spelke and Katherine D Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

[17] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[18] Ming Tan. Learning a cost-sensitive internal representation for reinforcement learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 358–362, 2014.

[19] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *JMLR*, 10:1633–1685, 2009.

[20] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.