ModelDiff: Leveraging Models for Policy Transfer with Value Lower Bounds

Xiaotian Liu[1], Jihwan Jeong[1], Ayal Taitler[1], Michael Gimelfarb[1], Scott Sanner[1]

[1]University of Toronto, Toronto, ON, Canada

Abstract

Despite significant recent advances in the field of Deep Reinforcement Learning (DRL), such methods typically incur high cost of training to learn effective policies, thus posing cost and safety challenges in many practical applications. To improve the learning efficiency of (D)RL methods, transfer learning (TL) has emerged as a promising approach to leverage prior experience on a source domain to speed learning on a new, but related, target domain. In this paper, we take a novel model-informed approach to TL in DRL by assuming that we have knowledge of both the source and target domain models (which would be the case in the prevalent setting of DRL with simulators). While directly solving either the source or target MDP via solution methods like value iteration is computationally prohibitive, we exploit the fact that if the target and source MDPs differ only due to a small structural change in their rewards, we can apply structured value iteration methods in a procedure we term ModelDiff to solve the much smaller target−source "Diff" MDP for a reasonable horizon. This ModelDiff approach can then be integrated into extensions of standard DRL algorithms like lower bound (LB) DQN where it provides enhanced provable LB guidance to DQN that speeds convergence. Experiments show that our ModelDiff LB-DQN matches or outperforms existing TL methods and baselines in both positive and negative transfer settings.

Introduction

Deep Reinforcement Learning (DRL) has emerged as the go-to approach for solving complex sequential decision-making problems (Arulkumaran et al. 2017; Silver et al. 2016). Despite recent advances in DRL, a critical limitation persists: DRL algorithms typically require extensive agent experience to develop effective policies (Mnih et al. 2015; Zhu, Lin, and Zhou 2020), thus posing cost and safety challenges in many practical applications (Gu et al. 2022).

To tackle the efficiency problem inherent in (D)RL, transfer learning has (Taylor and Stone 2009; Zhu, Lin, and Zhou 2020) emerged as a promising approach. TL capitalizes on the idea that an agent equipped with prior knowledge from a previously learned task can rapidly adapt and learn on a new, but similar, task. Thus the agent can significantly reduce the time and data required for learning a performant policy.

Existing model-free TL methods leverage multiple source policies or data as done in Probabilistic Policy Reuse (PPR) (Fernández and Veloso 2006) and in techniques involving reward shaping (Brys et al. 2015) or successor features (Barreto et al. 2018). In model-based transfer learning approaches, the focus shifts to using *learned* models of the environment to inform the transfer process (Song et al. 2016; Laroche and Barlier 2017). Specifically, these models can be instrumental in determining which source policy might be most effective to transfer to a similar task.

However, in real-world applications, it can be impractical to have access to multiple source policies or datasets (Vettoruzzo et al. 2023; Tian et al. 2020). Furthermore, even with exact or analytic models of the environment at hand, existing works fall short of leveraging this knowledge for efficient transfer (Dulac-Arnold et al. 2021). Particularly in scenarios where the source and target domains are known and dynamically similar but their reward differs in structurally simple ways (e.g., an additive term), there is a significant opportunity for more targeted and effective transfer learning, which has been largely overlooked. To better illustrate this, we first define the difference in models and rewards.

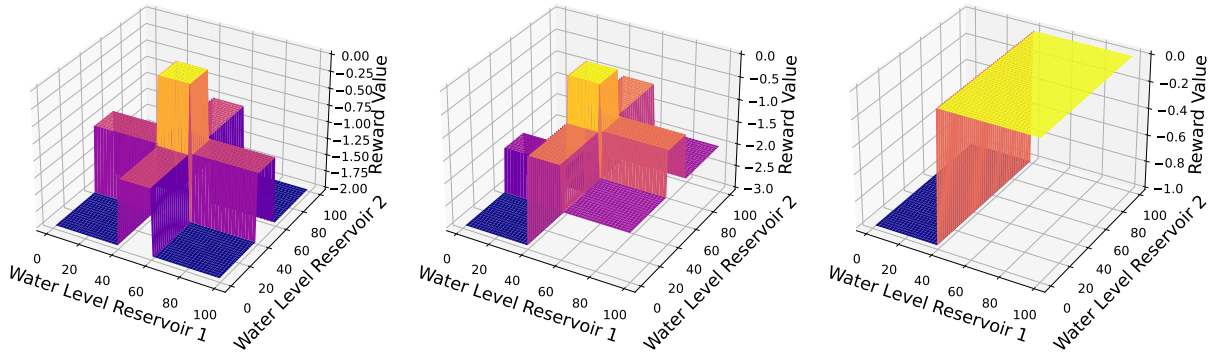
Definition 1 (Difference Model and Reward)**.** *Let M_s and M_t respectively denote the* source *and* target *tasks, modeled as Markov decision processes (MDP).*[1] *The two MDPs only differ in their reward functions, defined for state s and action a (i.e., $R_s(s,a)$ and $R_t(s,a)$, respectively). We define the* ***Difference MDP*** *as the MDP that shares the transition dynamics with M_s and M_t but whose reward function follows:*

$$R_d(s,a) = R_t(s,a) - R_s(s,a). \tag{1}$$

We call $R_d(s,a)$ the ***Difference reward*** *of M_t from M_s.*

For example, Figure 1 shows the reward functions of M_s, M_t, and M_d in RESERVOIR MANAGEMENT problem introduced in Section . Here, the diff reward R_d is far simpler than the individual rewards. This simplification is a common occurrence in practical applications, such as in robotics, where tasks often include a complex smoothness penalty alongside a straightforward goal-conditioned reward. For different tasks with varied goals, the smoothness penalty would remain consistent, resulting in a diff reward that can

[1]The formal definition and notation follows in Section .



(a) Source reward: R_s (b) Target reward: R_t (c) "Diff" reward: R_d

Figure 1: Source and target reward in RESERVOIR MANAGEMENT along with the simpler "Diff reward" owing to a minor structural change.

be expressed in a much simpler analytic form. This scenario raises a pivotal research question: *how can we effectively leverage the reward difference for efficient transfer learning?*

To address this challenge, we introduce **ModelDiff**, a novel TL framework designed to utilize analytically available models for efficient policy transfer from a single source to a target domain. Central to ModelDiff is the exploitation of the simpler *difference MDP*, which allows for the exact evaluation of the source policy under this MDP, leading to the derivation of a symbolic value function. Utilizing this value function, we develop a modified Bellman operator, which leverages the evaluated value function to expedite the knowledge transfer process from the source to the target domain.

The efficacy of ModelDiff is showcased through its integration with a standard model-free RL algorithm, DQN (Mnih et al. 2015), across three distinct single-source transfer learning scenarios. A key advantage of ModelDiff lies in the exactly evaluated value function of the policy, which serves as a lower bound for the temporal difference (TD) value targets. This lower bound acts as a robust learning signal, enhancing efficient exploration in cases of positive transfer. Importantly, it also ensures that the learning process is not adversely affected in scenarios of negative transfer.

We summarize three of our main contributions as follows:

1. We derive a novel Bellman-style operator on Q-value functions that provides a lower bound estimate on the optimal Q-value Q^*, and we prove its convergence to Q^*. We also establish a value iteration-style recurrence for the difference model that lower bounds Q^*.
2. Based on the above derivation, we introduce ModelDiff to provide lower bound (LB) guidance for DQN agents (i.e., LB-DQN). ModelDiff-informed LB-DQN significantly boosts learning efficiency in three experimental TL tasks compared to related work and baselines.
3. Lastly, we show that ModelDiff-informed LB-DQN effectively avoids negative transfer where the source policy might adversely affect learning in the target domain.

Preliminaries

Reinforcement Learning

In the reinforcement learning (RL) setting, an agent acts in an environment to optimize its performance based on a given reward function. We model the environment as a Markov Decision Process (MDP), defined by a tuple $\langle S, A, T, R, \gamma \rangle$. Here, S is the set of states, A is the set of valid actions, T is a Markovian transition function that describes the distribution $p(s'|s, a)$ of the next state s' given the current state s and action a, and r is the immediate reward function. The discount factor $\gamma \in (0, 1)$ ensures that rewards received far into the future receive less weight than immediate rewards.

The objective of an RL agent is to learn a policy $\pi : S \to A$ that maximizes the expected (discounted) sum of future rewards $Q^\pi(s, a) = \mathbb{E}_{s_t}\left[\sum_t \gamma^t R(s_t, \pi(s_t)) \,|\, s_0 = s,\, a_0 = a\right]$. Value-based RL aims to learn the optimal Q-values Q^* that are the fixed point $Q^* = \mathcal{B}(Q^*)$ of the Bellman operator

$$\mathcal{B}(Q) = \mathbb{E}_{s'}\left[R(s, a) + \gamma \max_{a'} Q(s', a')\right], \qquad (2)$$

which is a γ-contraction over the space of Q-value functions.

Q-learning (Sutton and Barto 2018) is a popular value-based RL algorithm that learns the Q value function via the so-called temporal difference (TD) update

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a)].$$

Deep Q-Network (DQN) is a neural net-based version of Q-learning that learns a neural network function approximator of the Q function (Mnih et al. 2015; van Hasselt, Guez, and Silver 2015).

Symbolic Dynamic Programming

Factored MDPs (Boutilier, Dean, and Hanks 1999) share the same components $\langle S, A, T, R, \gamma \rangle$ as MDPs. However, they assume that the state and dynamics are factorized in terms of a vector of discrete and continuous state variables. Symbolic Dynamic Programming (SDP) (Hoey et al. 1999) is a Value

Iteration algorithm designed to compute Bellman backups in (2) so as to provide *compact, structured* representations of the Q-values as Extended Algebraic Decision Diagrams (XADDs) in both the mixed discrete and continuous state and action settings (Sanner, Delgado, and de Barros 2011; Zamani, Sanner, and Fang 2012).

Related Work

Knowledge transfer in RL can take the form of polices, value functions, state transitions, rewards, or other sources (Zhu, Lin, and Zhou 2020). One common method is Probabilistic Policy Reuse (PPR) (Fernández and Veloso 2006), where exploration is biased by assigning a probability to act according to a set of policies learned from similar tasks. Potential based Reward Shaping (PBRS) (Ng, Harada, and Russell 1999) is another common technique that biases the reward function on a target domain (Brys et al. 2015; Vecerík et al. 2017). Our work also shares some characteristics with Successor Features-based transfer (GPI&SF) (Barreto et al. 2017, 2018), which assume all task rewards are linear combinations of shared reward components. However, while the latter work typically works in a restricted reward space (i.e. linear functions) and in the model-free setting, our work assumes the reward function is known and leverages the structural reward difference in (arbitrary) source and target rewards for transfer.

Now we turn to the use of MDP models for transfer in RL. (Song et al. 2016) proposed a method to select suitable policies and value functions for transfer via a distance measure among MDPs. (Brys et al. 2015) used reward shaping techniques to transfer policies via inter-task state space matching, while (Gimelfarb, Sanner, and Lee 2021) used transition function similarity. Meanwhile, (Lazaric, Restelli, and Bonarini 2008) and (Laroche and Barlier 2017) reused previous source task interactions by utilizing task and reward similarities. Work in this area often focused on using models to select the best policies for transfer. However, we focus on evaluating performance of any given policy in the target domain and the difference between models. Another key aspect often ignored by previous work is the high computational overhead required to reason over large models. Our work aims to reduce computational overhead required by analyzing model differences.

Methodology

We assume two MDPs $M_s = \langle S, A, T, R_s, \gamma \rangle$ and $M_t = \langle S, A, T, R_t, \gamma \rangle$, which are designated as source and target tasks, respectively. We assume that both tasks share identical S, A, and state transitions, but differ in their reward functions. A policy π_s and its corresponding value and Q-value functions, $V_s^{\pi_s}(s)$ and $Q_s^{\pi_s}(s,a)$, respectively, are first estimated in M_s (as a side effect, also producing a data set of past state observations that we will also use). Our goal is to use π_s, and either $V_s^{\pi_s}(s)$ or $Q_s^{\pi_s}(s,a)$, to accelerate learning an optimal policy in M_t by leveraging the reward difference in Definition 1.

A Lower Bound on the Optimal Q-Value and a Modified Bellman Operator

At a high level, we want to produce a well-informed estimate of $Q_t^{\pi_t}$ that we can leverage to quickly learn the target task. However, as mentioned previously, we have access to a source policy π_s and $Q_s^{\pi_s}$, as well as the reward difference between the two MDPs. To accomplish transfer, we will derive a lower bound $Q_t^{\pi_t}$ using the knowledge of $Q_s^{\pi_s}$, π_s, and the reward difference R_d.

We begin by defining the difference between rewards, value functions and Q-function for M_s and M_t for horizon h evaluated on policy π_s:

$$R_d(s,a) = R_t(s,a) - R_s(s,a) \tag{3}$$

$$V_d^{\pi_s,h}(s) = V_t^{\pi_s,h}(s) - V_s^{\pi_s,h}(s) \tag{4}$$

$$Q_d^{\pi_s,h}(s,a) = Q_t^{\pi_s,h}(s,a) - Q_s^{\pi_s,h}(s,a) \tag{5}$$

Rearranging these equations, we have:

$$V_t^{\pi_s,h}(s) = V_s^{\pi_s,h}(s) + V_d^{\pi_s,h}(s) \tag{6}$$

$$Q_t^{\pi_s,h}(s,a) = Q_s^{\pi_s,h}(s,a) + Q_d^{\pi_s,h}(s,a) \tag{7}$$

Furthermore, since $Q_t^{\pi_s,h}(s,a)$ is a lower bound on $Q_t^{\pi_t,h}(s,a) = Q_t^*(s,a)$, and assuming that $Q_d^{\pi_s,h}$ can be accurately estimated, then (7) provides a valid lower bound on the optimal target task Q-value.

Inspired by this observation, we define a modified Bellman operator $\mathcal{L}_f$ that can take advantage of the lower bound information to achieve transfer:

$$\begin{aligned} &\mathcal{L}_f(Q)(s,a) \\ &= \max\left(\mathcal{B}(Q)(s,a), f(s,a)\right) \\ &= \max\left(\mathbb{E}_{s'}\left[R(s,a) + \gamma \max_{a'} Q(s',a')\right], f(s,a)\right), \end{aligned} \tag{8}$$

where $f(s,a)$ is a lower bound on Q^*, i.e. $f(s,a) \leq Q^*(s,a), \forall s,a$. Intuitively, if f is close to Q^*, then updating the Q-value using $\mathcal{L}_f$ will drive the Q-value estimates closer to Q^* than when using $\mathcal{B}$. Thus, we can interpret f as a source of knowledge transfer.

Fortunately, this operator also converges to Q_t^*.

Theorem 1. *Starting with Q_0, and f such that $f(s,a) \leq Q^*(s,a), \forall s,a$, the sequence Q_k produced by $Q_{k+1} = \mathcal{L}_f(Q_k)$ converges to the optimal value function Q^* in the usual sup-norm $\|\cdot\|_\infty$, i.e. the fixed point $Q^* = \mathcal{B}(Q^*)$.*

Proof. First, observe that for any k, we have

$$\begin{aligned} &|\max(\mathcal{B}(Q_k)(s,a), f(s,a)) - Q^*(s,a)| \\ &\leq |\mathcal{B}(Q_k)(s,a) - Q^*(s,a)|. \end{aligned}$$

To prove this, observe that if $f(s,a) \geq \mathcal{B}(Q_k)(s,a)$, then

$$\begin{aligned} 0 &\leq Q^*(s,a) - \max(\mathcal{B}(Q_k)(s,a), f(s,a)) \\ &\leq Q^*(s,a) - \mathcal{B}(Q_k)(s,a). \end{aligned}$$

On the other hand, if $f(s,a) < \mathcal{B}(Q_k)(s,a)$, then

$$\begin{aligned} &Q^*(s,a) - \max(\mathcal{B}(Q_k)(s,a), f(s,a)) \\ &= Q^*(s,a) - \mathcal{B}(Q_k)(s,a). \end{aligned}$$

Using this result, we continue as follows:

$$
\begin{aligned}
&|Q_{k+1}(s,a) - Q^*(s,a)| \\
&= |\mathcal{L}_f(Q_k)(s,a) - Q^*(s,a)| \\
&= |\max\left(\mathcal{B}(Q_k)(s,a), f(s,a)\right) - Q^*(s,a)| \\
&\leq |\mathcal{B}(Q_k)(s,a) - Q^*(s,a)| \\
&= |\mathcal{B}(Q_k)(s,a) - \mathcal{B}(Q^*)(s,a)| \\
&\leq \gamma \|Q_k - Q^*\|_\infty,
\end{aligned}
$$

where in the last step we have used the fact that the Bellman operator $\mathcal{B}$ is a γ-contraction. Thus, taking sup of both sides over s, a, we conclude that

$$\|Q_{k+1} - Q^*\|_\infty \leq \gamma \|Q_k - Q^*\|_\infty,$$

and thus that $Q_k \to Q^*$ as $k \to \infty$, as claimed. □

Thus, setting $f = Q_t^{\pi_s}$, we have derived a modified Bellman operator $\mathcal{L}_{Q^{\pi_s}}$ that makes use of the lower bound information for updating the target task Q-values, while still achieving convergence to Q_t^*. As an added bonus, the last argument in the proof, namely the inequality $|Q_{k+1}(s,a) - Q^*(s,a)| \leq |\mathcal{B}(Q_k)(s,a) - Q^*(s,a)|$, can be used to show (by induction) that value iteration using $\mathcal{L}_f$ converges to Q^* at least as fast as $\mathcal{B}$. However, to apply this operator to learn the target Q-value function, we still need to know $Q_d^{\pi_s}$.

Symbolic Dynamic Programming for Value Difference Estimation

In order to estimate $V_d^{\pi_s}$ or $Q_d^{\pi_s}$, we first establish that they can be computed using the ordinary policy evaluation equation:

$$
\begin{aligned}
&V_d^{\pi_s,h}(s) = V_t^{\pi_s,h}(s) - V_s^{\pi_s,h}(s) \\
&= R_t(s,\pi_s(s)) + \gamma \sum_{s'} P(s'|s,\pi_s(s)) V_t^{\pi_s,h-1}(s') \\
&\quad - \left(R_s(s,\pi_s(s)) + \gamma \sum_{s'} P(s'|s,\pi_s(s)) V_s^{\pi_s,h-1}(s')\right) \\
&= R_d(s,\pi_s(s)) + \gamma \sum_{s'} P(s'|s,\pi_s(s)) V_d^{\pi_s,h-1}(s'). \quad (9)
\end{aligned}
$$

An identical relation also holds for the Q-value functions:

$$
\begin{aligned}
&Q_d^{\pi_s,h}(s,a) = Q_t^{\pi_s,h}(s,a) - Q_s^{\pi_s,h}(s,a) \\
&= R_d(s,a) + \gamma \sum_{s'} P(s'|s,a) Q_d^{\pi_s,h-1}(s',\pi_s(s')). \quad (10)
\end{aligned}
$$

Therefore, $Q_d^{\pi_s}$ can be derived by performing a Bellman backup using knowledge of R_d! However, recall that R_d defines an MDP with a much simpler reward function than R_s and R_t (cf. Figure 1), thus enabling efficient symbolic computation of the value function.

Based on this observation and the recursion (10), we can apply symbolic dynamic programming (SDP) leveraging an extended algebraic decision diagram (XADD) representation of the source and target models (Sanner, Delgado, and de Barros 2011; Zamani, Sanner, and Fang 2012) to evaluate policy π_s in the difference MDP, which would allow us to efficiently compute $Q_d^{\pi_s}$. However, to accomplish this, the source policy π_s must be in a symbolic XADD form compatible with SDP computations.

For scenarios where the source policy is manually specified, we can directly convert such a policy into a symbolic representation like an XADD. Policies deployed in real-world are often specified by domain experts due to the need for safety and variability. Additionally, learning a sufficiently good policy is often computationally costly. However, in cases where the source policy is implicitly represented in a function approximator like a DQN, we can obtain a sufficiently good approximation of the source policy by sampling state-action pairs from the neural network. To do this, we construct a dataset $D = \{(s_i, \pi_s(s_i))\}_i$ from past states s_i observed while learning the source policy π_s in the source MDP, and use it to recover a decision tree (DT) policy $\hat{\pi_s} \approx \pi_s$. This last step can be accomplished by training a standard off-the-shelf decision tree tool to do supervised classification on D. (the details of this procedure are discussed in the Appendix A). Fortunately, *decision trees are decision diagrams* so the learned DT policy $\hat{\pi_s}$ can be converted directly into XADD form and used to recover the value $Q_d^{\pi_s}$ via SDP, which in turn provides a lower bound on $Q_t^{\pi_t}$, i.e. $Q_t^{\pi_t}(s,a) \geq Q_t^{\pi_s}(s,a) = Q_s^{\pi_s}(s,a) + Q_d^{\pi_s}(s,a)$. We found that, given enough tree depth and a small sampling interval, we can extract a policy that sufficiently mimics the exact behavior in the reachable states of the agent.

Algorithm 1: Lower Bound DQN (LB-DQN)

Initialize replay memory D with capacity N
Initialize Q-network with weights θ
Initialize target network with weights $\theta^- = \theta$
for episode $= 1, \ldots, M$ **do**
  Initialize state s
  for $t = 1, \ldots, H$ **do**
    With probability ϵ, select a random action a
    Otherwise select $a = \arg\max_{a'} Q(s, a'; \theta)$
    Execute action a, observe reward r and next state s'
    Store transition (s, a, r, s') in D
    Sample a batch $\{(s_j, a_j, r_j, s'_j)\}_{j=1}^{J}$ from D
    Compute target for all $j \in \{1, \ldots, J\}$

$$y_j \leftarrow \max\left(r_j + \gamma \max_{a'} Q(s'_j, a'; \theta^-), Q_t^{\pi_s}(s_j, a_j)\right)$$

    Update $\theta \leftarrow \theta - \frac{\eta}{J} \nabla_\theta \sum_j (y_j - Q(s_j, a_j; \theta))^2$
    Update target network $\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$
    Set $s \leftarrow s'$
  end for
end for

Incorporating the Lower Bound into RL

Finally, we incorporate the derived lower bound $Q_t^{\pi_s}$ into Q-learning by replacing the greedy Bellman backup $\mathcal{B}$ with the modified backup $\mathcal{L}_{Q_t^{\pi_s}}$.

Specifically, during the TD update step, we take the maximum between the bootstrap target value for each (s, a, r, s'), and our established lower bound, $Q_t^{\pi_s}(s, a)$. For each state-

action pair, the target update is represented as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [y_{target} - Q(s,a)]$$
$$y_{target} = \max\left(r(s,a) + \gamma \max_{a'} Q(s',a'), Q_t^{\pi_s}(s,a)\right). \tag{11}$$

Thus, intuitively, the lower-bound forces the agent to act according to $Q_t^{\pi_s^*}$ in the initial steps of training, thus providing a "model-informed" warm-start. In the later stages of training, when it is anticipated that the target agent's Q-function is a good approximation of Q_t^*, the target value above becomes the usual Q-learning target.

To make the algorithm scale well in large state or action spaces, we parameterize the Q-function as a Deep Q-Network (DQN) (Mnih et al. 2015). Let $Q(s,a;\theta) \approx Q(s,a)$ be a neural network with weights θ. The modified DQN loss for a transition (s,a,s'), derived from $\mathcal{L}_{Q_t^{\pi_s}}$, is:

$$L(\theta) = \frac{1}{2}\left(y_{target} - Q(s,a;\theta)\right)^2,$$
$$y_{target} = \max\left(r(s,a) + \gamma \max_{a'} Q(s',a';\theta^-), Q_t^{\pi_s}(s,a)\right) \tag{12}$$

where θ^- indicates that the gradient ∇_θ does not propagate through $Q(s',a';\theta^-)$. The LB-DQN implementation using the ModelDiff lower bound can be found in Algorithm 1.

Empirical Evaluation

We evaluated LB-DQN with the ModelDiff derived lower bound on three different domains in a transfer learning setting with source and target tasks in each. All the domains and tasks have been implemented and trained in *pyRDDLGym* (Taitler et al. 2022). The experiments are designed to answer two research questions:

1. **RQ 1:** Can ModelDiff with LB-DQN accelerate learning on a target task based on the model and a policy of a source task, compared to baselines that ignore the presence of an explicit model?
2. **RQ 2:** Can ModelDiff with LB-DQN effectively prevent negative transfer, ensuring that the policy learned on the source task does not adversely impact performance on the target task, potentially leading to outcomes worse than learning from scratch?

Experimental Setup

We evaluate our ModelDiff-informed LB-DQN (from here out, shortened to just LB-DQN) on three benchmark domains: POWERGEN, PICK-AND-PLACE, and RESERVOIR. For each domain, we know the analytic forms of M_s and M_t, and a source policy to be transferred is given. Specifically, we use a pre-trained neural policy derived by a DQN agent on M_s and leverage it in the ModelDiff framework of Section to derive the lower bound for M_t.

For each domain, we define two target tasks that share the same states, actions, and transition function as in M_s but have different reward functions. The first target task is designed to test *positive transfer*; therefore, the difference in the reward functions is small, and the source policy can achieve high rewards in all but a small subset of states in the target task. The second target task is used to test *negative transfer*. In this scenario, the source policy behaves adversarially in the target task. We aim to assess if LB-DQN can effectively disregard the adversarial behavior of this suboptimal policy. The detailed transition and rewards of the domains can be bound in Appendix B.

All the experiments are compared to baseline methods that are appropriate for our experimental setting:

- **DQN** – the vanilla model-free DQN, that learns from scratch on the target domain.
- **Probabilistic Policy Reuse (PPR)** (Fernández and Veloso 2006) – a DQN-based method that assigns a probability of using the given policy and thus collects also transitions related to that policy.
- **Warm Start (WS)** – DQN with the value function being directly initialized with the source value function.

In the positive transfer experiments, all methods are expected to learn more efficiently than vanilla DQN. In the negative transfer experiments, success is measured by the method's ability to be at least as good as DQN that intrinsically ignores transfer, which in this case is negative transfer.

Benchmark Domains

RESERVOIR Drawing from the water reservoir management problem (Yeh 1985), we set up a 2-reservoir system with one upstream reservoir t_1 and one downstream reservoir t_2. The water released from t_1 flows to t_2, and the water released from t_2 is directly released to the ocean and leaves the system. The objective is to maintain the water levels of both reservoirs within a maximum and minimum threshold. We give a negative reward if the water level goes outside the given thresholds. At each time step, there is a chance for rainfall, modeled by a stochastic Bernoulli random variable (RV). When it rains, a fixed amount of water is added to the reservoirs. The RV of the first reservoir is independent of the variable of the other reservoir. We have a Boolean decision for each reservoir, dictating whether to release water.

- For the positive transfer case, we reduce the penalty for being below the minimal threshold by half for the upstream reservoir t_1, which should only slightly alter optimal actions in the new task.
- For the negative transfer case, we move both the maximum and the minimum threshold. For the majority of states in the target task, optimal decisions differ from those in the source task.

PICK-AND-PLACE In this domain, an agent needs to discover an object of interest, pick it up, and place it in another location. The problem is inspired by a 2D navigation problem specified in (Taitler et al. 2022). We added a discrete state variable *has-object* in this domain to indicate if the agent holds the object. The agent can *pick-up* an object if it is in the vicinity of the agent. A reward of 1 is given if the agent is holding the object in one of the two goal regions, and 0 otherwise. The agent can move in the x or y direction with a fixed step size.

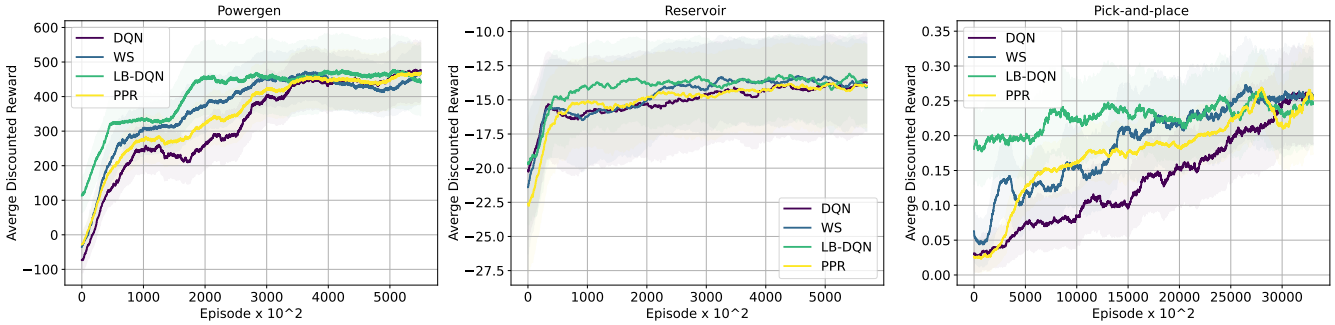


Figure 2: Average cumulative discounted rewards in Positive Transfer tasks

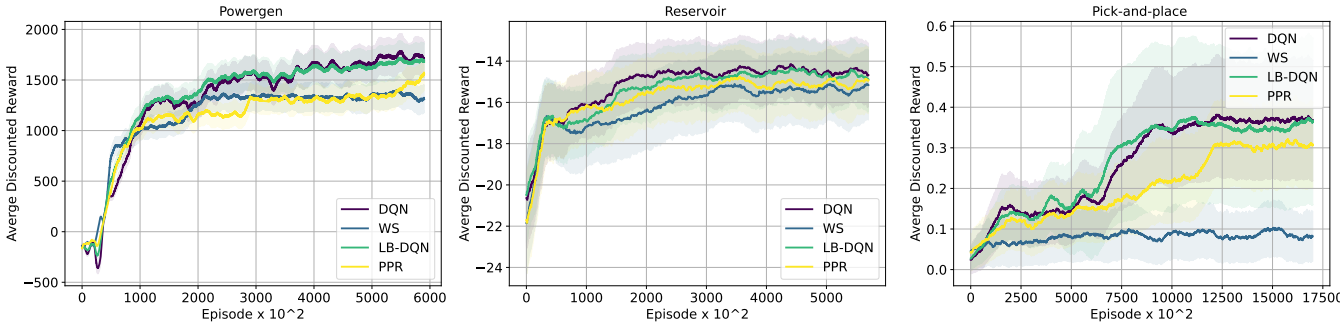


Figure 3: Average cumulative discounted rewards in Negative Transfer tasks

- For positive transfer case, we increase the reward of one of the goal locations.
- For negative transfer case, we reverse the reward of the two goal locations.

POWERGEN We use a modified version of the Power Unit Commitment problem (Padhy 2004) as the third benchmark domain. Here, it models 3 power producers that act cooperatively to meet daily demand. The demand for each timestep is a Bernoullli RV. The agent receives a reward for each demand successfully met. If the demand is not met, then a large penalty occurs to deter overproduction.

- For the positive transfer case, we increase the production cost of a single power producer.
- For the negative transfer case, we increase the production cost of all power producers.

Table 1: Hyper-parameters for DQN and LB-DQN Training

Hyperparameter	***PowerGen***	***Reservoir***	***Pick-and-Place***
Learning Rate	0.001	0.001	0.001
Batch Size	64	64	128
Discount Factor γ	0.9	0.9	0.9
Soft Update τ	0.001	0.001	0.001
Optimizer	Adam	Adam	Adam
Replay Buffer	500000	500000	1000000
Exploration Rate	0.1	0.1	0.1
Network	[64, 64]	[64, 64]	[64, 128, 64]
SDP Steps	20	20	20

Training Details

We conducted a hyper-parameter search for a vanilla DQN agent across each of the three domains, utilizing the source tasks. The selected hyper-parameters are given in Table 1. These hyper-parameter values are consistently used by all methods during the learning phase on the target tasks including LB-DQN . Each task is set with a horizon of 20, and the discount factor γ is fixed at 0.9. To evaluate and record the agent's performance, we freeze network weights after every 10 training episodes.

Empirical Results

For all the experiments conducted in this work, 10 runs were executed, and both the average and standard deviation were calculated. In the provided plots, the bold colored lines represent the average, while the grayed-out envelopes indicate the standard deviation across the 10 runs.

Positive Transfer Results The results for the positive transfer experiments for all three domains are given in Figure 2. These scenarios are designed such that a large portion of the policy of the source task can be used on the target task. Consequently, all three transfer methods outperform the vanilla DQN on all three domains. We observe that LB-DQN outperforms all other transfer baselines, i.e., the reward is maximized faster in LB-DQN , while ultimately all the methods converge to the same maximum rewards, illustrating the sample complexity of each method. The benefit of using the explicit model directly, is showcased especially

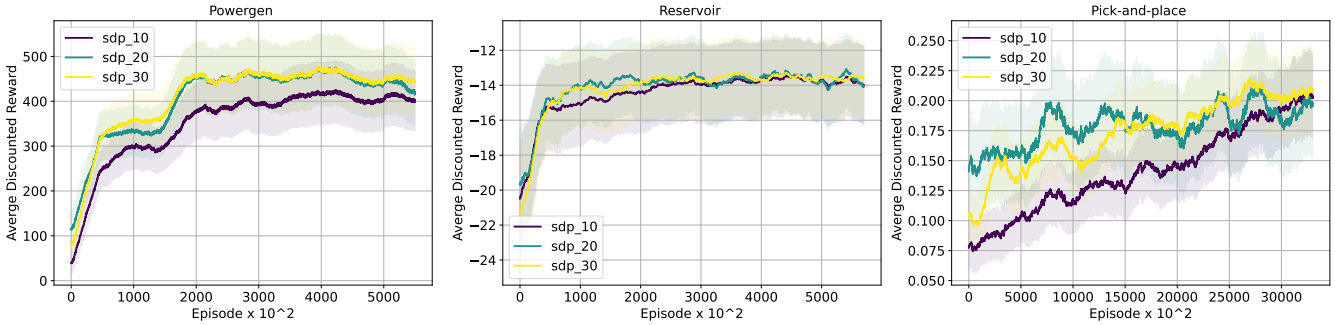


Figure 4: Number of SDP Steps Ablation

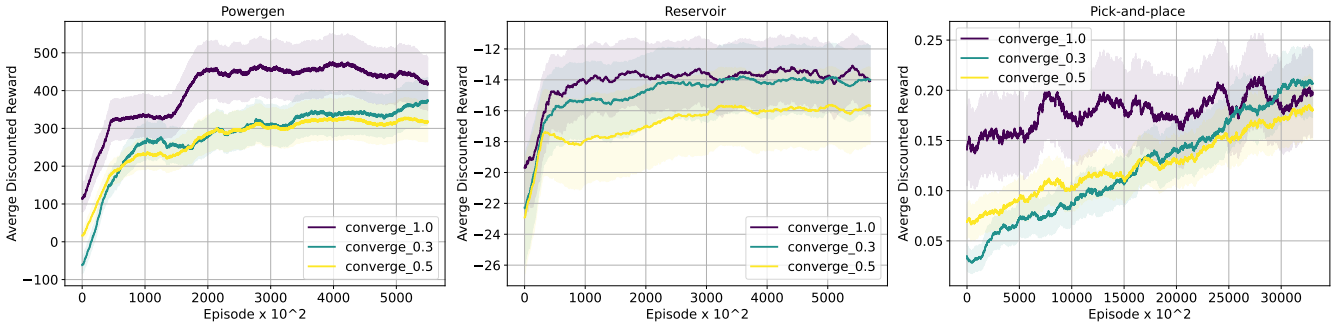


Figure 5: Value Convergence Ablation

in the *pick-and-place* domain where the initial reward at the beginning of learning is higher than all the others by a significant gap.

Negative Transfer Results In situations where the target task and the source task are different in such a way that a policy suited for the source task might be a hindrance for the target task, these cases are referred to as negative transfer. We desire for a transfer approach to ignore such transfer as it might lead to slow learning or failure to learn altogether. The negative transfer results are shown in Figure 3. We observe in all three domains that all methods learn slowly initially. We also critically observe that using an adversarial policy might not just delay performance, but prevent the algorithm from finding a good policy altogether as in the *pick-and-place* experiment where WS clearly gets stuck. This case shows that warm-starting from an adversarially initialized policy value might be even worse than starting from a randomly initialized value (i.e., network weights) and learning from scratch. We note that across all three domains LB-DQN is on par with DQN, suggesting it does not do better but is also able to ignore the negative transfer. PPR and WS on the other hand all suffer across all three domains from negative transfer effects.

Ablations

As ModelDiff-informed LB-DQN is comprised of several components and parameters, such as the source trained policy and Q-network for M_s and the ModelDiff SDP evaluation horizon, we conduct two experiments designed at assessing the sensitivity of these components. Both experiments are conducted on the positive transfer scenario to isolate the desired effects without the influence of negative transfer.

SDP Horizon The lower bound $Q_t^{\pi_s}$, derived from the ModelDiff is computed via SDP as described in Section . Computationally, running this algorithm to convergence is expensive and thus as the SDP horizon parameter H increases, time increases and the lower bound tightness improves.

We examine how LB-DQN performs vs. $H \in \{10, 20, 30\}$ as shown in Figure 4. We note that increasing the horizon to 30 does not yield significantly better results. Also, while the performance of LB-DQN under 10 SDP steps is inferior to the other runs, we note that the difference is not significant, suggesting that the number of SDP steps required for the lower bound calculation does not need to be large. However, the value of H that optimally trades off performance and evaluation quality can vary between between domains.

Source Policy Quality Key components required for transfer include the reward diff R_d, a source policy π_s for M_s and its corresponding Q-value function. In principle, any π_s can be used since it would provide a lower bound value for M^t (tight, if serendipitously optimal). Nonetheless, policy quality is expected to impact the performance of LB-DQN. In this ablation, we aim to assess how sensitive the performance of LB-DQN is in relation to the quality of

the provided policy and Q-values. We achieve this in a controlled setting by training three Q-functions and policies on the source task with varying quality, simply by varying the training time. Namely, we train DQN in three regimes: (1) to optimality, (2) training for half the time required to achieve optimality, and (3) training for 30% of the time required to achieve optimality.

The results for this study are given in Figure 5. LB-DQN is able to effectively transfer from all three policies and Q-functions, although we note that the best performance is obtained by using the optimal source task policy, as expected in this setting of positive transfer. The benefits of transfer learning are most notable on the *pick-and-place* domain, where the optimal source task policy resulted in immediate (warm-start) improvement on the target task. Nonetheless, LB-DQN converges towards the optimal target task policy even when provided with the *highly sub-optimal* source task policy.

Conclusion

In this paper, we asked how we could leverage model knowledge of a source and target MDP with matching dynamics but differing reward functions to facilitate transfer learning in a deep reinforcement learning (DRL) setting. We exploited the structural differences between the reward functions to symbolically derive a so-called ModelDiff lower bound on the target MDP. This ModelDiff lower bound was integrated into an LB-DQN extension of the standard DQN algorithm, where it sped convergence of DQN on the target MDP.

Our experimental results demonstrate that our ModelDiff-informed LB-DQN learns faster than other transfer learning techniques in the positive transfer setting and is critically able to ignore negative transfer in contrast to competing methods. Ablation studies further show that increasing the ModelDiff horizon has positive, but diminishing returns, while source MDP policy quality has a significant impact on transfer efficiency. Overall, this work opens a novel and effective direction for model-informed transfer to help address longstanding sample complexity issues in DRL and inspire novel model-informed transfer extensions beyond LB-DQN.

References

Arulkumaran, K.; Deisenroth, M. P.; Brundage, M.; and Bharath, A. A. 2017. A Brief Survey of Deep Reinforcement Learning. *CoRR*, abs/1708.05866.

Barreto, A.; Borsa, D.; Quan, J.; Schaul, T.; Silver, D.; Hessel, M.; Mankowitz, D. J.; Zídek, A.; and Munos, R. 2018. Transfer in Deep Reinforcement Learning Using Successor Features and Generalised Policy Improvement. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, 510–519. PMLR.

Barreto, A.; Dabney, W.; Munos, R.; Hunt, J. J.; Schaul, T.; van Hasselt, H. P.; and Silver, D. 2017. Successor features for transfer in reinforcement learning. *NeurIPS*, 30.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11: 1–94.

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees*. Wadsworth.

Brys, T.; Harutyunyan, A.; Taylor, M. E.; and Nowé, A. 2015. Policy Transfer using Reward Shaping. In *AAMAS*, 181–188. ACM.

Dulac-Arnold, G.; Levine, N.; Mankowitz, D. J.; Li, J.; Paduraru, C.; Gowal, S.; and Hester, T. 2021. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Mach. Learn.*, 110(9): 2419–2468.

Fernández, F.; and Veloso, M. M. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *AAMAS*, 720–727. ACM.

Gimelfarb, M.; Sanner, S.; and Lee, C.-G. 2021. Contextual policy transfer in reinforcement learning domains via deep mixtures-of-experts. In *UAI*, 1787–1797. PMLR.

Gu, S.; Yang, L.; Du, Y.; Chen, G.; Walter, F.; Wang, J.; Yang, Y.; and Knoll, A. 2022. A Review of Safe Reinforcement Learning: Methods, Theory and Applications. *arXiv preprint arXiv:2205.10330*.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic Planning using Decision Diagrams. In *UAI*, 279–288. Stockholm.

Laroche, R.; and Barlier, M. 2017. Transfer Reinforcement Learning with Shared Dynamics. In *AAAI*, 2147–2153. AAAI Press.

Lazaric, A.; Restelli, M.; and Bonarini, A. 2008. Transfer of samples in batch reinforcement learning. In *ICML*, volume 307 of *ACM International Conference Proceeding Series*, 544–551. ACM.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518: 529–533.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *ICML*, 278–287. Morgan Kaufmann.

Padhy, N. P. 2004. Unit commitment-a bibliographical survey. *IEEE Transactions on power systems*, 19(2): 1196–1205.

Sanner, S.; Delgado, K. V.; and de Barros, L. N. 2011. Symbolic Dynamic Programming for Discrete and Continuous State MDPs. In *UAI*, 643–652. AUAI Press.

Sanner, S.; Delgado, K. V.; and de Barros, L. N. 2011. Symbolic Dynamic Programming for Discrete and Continuous State MDPs. In *UAI*. Barcelona.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T. P.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D.

2016. Mastering the game of Go with deep neural networks and tree search. *Nat.*, 529(7587): 484–489.

Song, J.; Gao, Y.; Wang, H.; and An, B. 2016. Measuring the Distance Between Finite Markov Decision Processes. In Jonker, C. M.; Marsella, S.; Thangarajah, J.; and Tuyls, K., eds., *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, 468–476. ACM.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*.

Taitler, A.; Gimelfarb, M.; Gopalakrishnan, S.; Liu, X.; and Sanner, S. 2022. pyRDDLGym: From RDDL to Gym Environments. *CoRR*, abs/2211.05939.

Taylor, M. E.; and Stone, P. 2009. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, 10(56): 1633–1685.

Tian, Y.; Wang, Y.; Krishnan, D.; Tenenbaum, J. B.; and Isola, P. 2020. Rethinking Few-Shot Image Classification: A Good Embedding is All You Need? In *ECCV (14)*, volume 12359 of *Lecture Notes in Computer Science*, 266–282. Springer.

van Hasselt, H.; Guez, A.; and Silver, D. 2015. Deep Reinforcement Learning with Double Q-learning. *CoRR*, abs/1509.06461.

Vecerík, M.; Hester, T.; Scholz, J.; Wang, F.; Pietquin, O.; Piot, B.; Heess, N.; Rothörl, T.; Lampe, T.; and Riedmiller, M. A. 2017. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *CoRR*, abs/1707.08817.

Vettoruzzo, A.; Bouguelia, M.; Vanschoren, J.; Rögnvaldsson, T. S.; and Santosh, K. 2023. Advances and Challenges in Meta-Learning: A Technical Review. *CoRR*, abs/2307.04722.

Yeh, W. W.-G. 1985. Reservoir management and operations models: A state-of-the-art review. *Water resources research*, 21(12): 1797–1818.

Zamani, Z.; Sanner, S.; and Fang, C. 2012. Symbolic Dynamic Programming for Continuous State and Action MDPs. In *AAAI*. Toronto, Canada.

Zhu, Z.; Lin, K.; and Zhou, J. 2020. Transfer Learning in Deep Reinforcement Learning: A Survey. *CoRR*, abs/2009.07888.

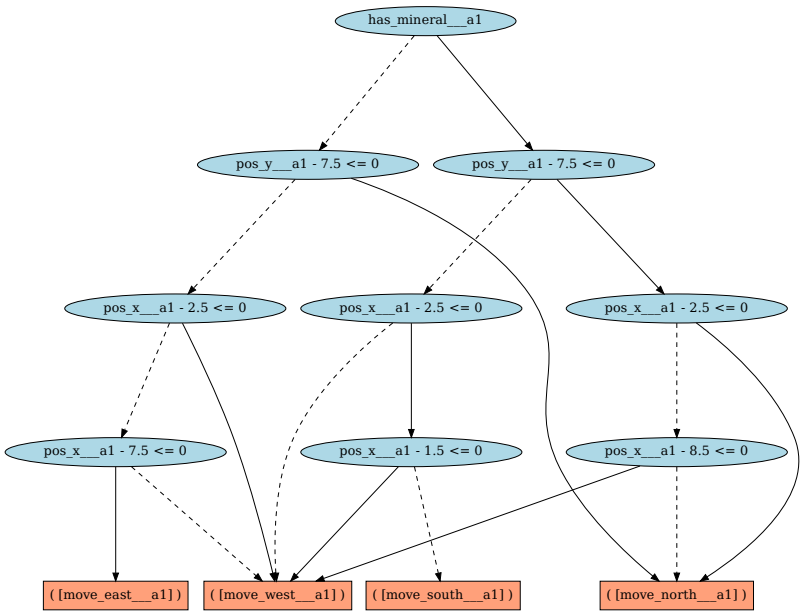


Figure 6: Example learned Decision Tree policy for Pick-and-Place converted to an XADD for use with Symbolic Dynamic Programming.

Appendix A: Symbolic Policy Extraction

Symbolic Dynamic Programming (SDP) operations require an explicit symbolic extended algebraic decision diagram (XADD) representation of the source policy (Sanner, Delgado, and de Barros 2011). In this section, we show how to distill a symbolic policy from the source DQN agent. The process takes three steps: (1) We generate a dataset $D = \{(s_k, \pi_s(s_k))\}_k$ using a source policy π_s. (2) We use the resulting dataset to train an off-the-shelf Decision Tree (DT) classifier $\hat{\pi_s}$ that is an approximation of the source policy $\hat{\pi_s} \approx \pi_s$. (3) We convert the resulting DT to the XADD format that can be used for SDP.

Data Generation

The first step is to generate a set of input-output pairs D using the source policy π_s. For a continuous state variable c_i, we set max_{c_i}, min_{c_i} and an interval value a_i to construct a set $C_i = \{min_{c_i} + n \cdot a_i \mid 0 \leq n \leq \frac{max_{c_i} - min_{c_i}}{a}\}$. We assign the set $B_j = \{1, 0\}$ for each Boolean variable b_j. We then independently and uniformly sample from C_i and B_i for each input state $s_k = (c_1, \ldots, c_i, b_1, \ldots, b_j)$. The input dataset is then constructed as $D = \{(s_k, \pi_s(s_k))\}_k$ using the source policy. The number of samples is set to be $|D| = \prod_{m=1}^{i} |C_m| \cdot \prod_{n=1}^{j} |B_n|$ to ensure sufficient coverage of the relevant state space.

Decision Tree (DT) Policy Construction

DT classifiers are non-parametric supervised learning methods that are commonly used for classification tasks. Nodes of a DT classifier are discrete variable assignment tests or arithmetic comparisons and leaves are output classes. We use an off-the-shelf DT classifier (SKLEARN.TREE.DECISIONTREECLASSIFIER in the Scikit-learn Python package) that takes the dataset D as input-output pairs to train. We use the Gini index (Breiman et al. 1984) to split decision nodes with a maximum tree depth of 20 and the default setting for remaining parameters. The trained DT classifier $\hat{\pi_s}$ is a policy function that mimics behaviour of the source policy $\hat{\pi_s} \approx \pi_s$.

XADD Conversion

Our DT classifier policy representation is already in the form of a decision diagram without reconvergent branches and is thus easily reduced to XADD form (Sanner, Delgado, and de Barros 2011) with a few syntactic translation steps. Since most off-the-shelf DT classifiers do not handle mixed discrete and continuous variables, we convert values for a Boolean variable b_j from $\{True, False\}$ to $\{1.0, 0.0\}$ during the training process. A DT comparison node for Boolean variable b_j is in the form of $b_j \leq 0.5$. For decision tests on Boolean b_j, we convert the comparison operator back to a Boolean test for $False$ if $b_j \leq 0.5$,

else $True$. A DT comparison node for continuous variable c_i is represented by an inequality with learned threshold t_i in the form of $c_i \leq t_i$. Since XADDs directly support arithmetic inequalities, these DT nodes are mapped one-to-one to XADD decision nodes. An example XADD policy for the Pick-and-Place domain is shown in Figure 6 with DT max depth set to 4 for illustrative purposes.

Appendix B: Domain Formulation

Reservoir

We examine a two reservoir system with one upstream reservoir t_1 and one downstream reservoir t_2. The water released from t_1 flows to t_2, and the water released from t_2 is directly released to the ocean. Each reservoir has a continuous state variable ($l1$, $l2$) that indicates water level. We set ranges for both variables as $l1 \in [0, 100]$ and $l2 \in [0, 100]$; values outside of the range are clipped (below). We call each threshold $T1 = (l1_{min}, l1_{max})$ and $T2 = (l2_{min}, l2_{max})$. At each time step, there is a chance for rainfall for each reservoir, modeled by stochastic Bernoulli random variables $b1$ and $b2$. When it rains, a fixed amount of water is added to the reservoirs. There are two Boolean actions $A = \{r1, r2\}$ which release a fixed amount of water downstream. The agent can only take one action at a time. The objective is to maintain the water levels of both reservoirs within a maximum and minimum threshold. For each time step, we give a negative reward if the water level underflows or overflows the respective thresholds. The transition and reward function are shown as follows:

$$P(b1) \sim Bernoulli(0.5)$$
$$P(b2) \sim Bernoulli(0.5)$$

$$l1' = \begin{cases} r1: & \min(\max(0, l1 + b1 \cdot 8 - 5), 100) \\ \neg r1: & \min(\max(0, l1 + b1 \cdot 8), 100) \end{cases}, \quad l2' = \begin{cases} r1 \wedge r2: & \min(\max(0, l2 + b2 \cdot 8 - 5 + \min(l1, 5)), 100) \\ \neg r1 \wedge r2: & \min(\max(0, l2 + b2 \cdot 8 - 5), 100) \\ r1 \wedge \neg r2: & \min(\max(0, l2 + b2 \cdot 8 + \min(l1, 5)), 100) \\ \neg r1 \wedge \neg r2: & \min(\max(0, l2 + b2 \cdot 8), 100) \end{cases},$$

$$R(l1, l2) = \begin{cases} ((l1 \geq l1_{max}) \vee (l1 \leq l1_{min})) \wedge ((l2 \geq l2_{max}) \vee (l2 \leq l2_{min})): & -2 \\ \neg" \wedge ((l1 \geq l1_{max}) \vee (l1 \leq l1_{min})) \wedge \neg((l2 \geq l2_{max}) \vee (l2 \leq l2_{min})): & -1 \\ \neg" \wedge \neg((l1 \geq l1_{max}) \vee (l1 \leq l1_{min})) \wedge ((l2 \geq l2_{max}) \vee (l2 \leq l2_{min})): & -1 \\ otherwise: & 0 \end{cases},$$

For the positive transfer case, we set $T1 = (35, 55), T2 = (35, 55)$, and halve the penalty for going below the minimum threshold for $t1$. The reward functions are:

$$R_s(l1, l2) = \begin{cases} ((l1 \geq 55) \vee (l1 \leq 35)) \wedge ((l2 \geq 55) \vee (l2 \leq 35)): & -2 \\ \neg" \wedge ((l1 \geq 55) \vee (l1 \leq 35)) \wedge \neg((l2 \geq 55) \vee (l2 \leq 35)): & -1 \\ \neg" \wedge \neg((l1 \geq 55) \vee (l1 \leq 35)) \wedge ((l2 \geq 55) \vee (l2 \leq 35)): & -1 \\ otherwise: & 0 \end{cases},$$

$$R_t(l1, l2) = \begin{cases} (l1 \geq 55) \wedge ((l2 \geq 55) \vee (l2 \leq 35)): & -2 \\ \neg" \wedge (l1 \leq 35) \wedge ((l2 \geq 55) \vee (l2 \leq 35)): & -1.5 \\ \neg" \wedge (l1 \geq 55) \wedge \neg((l2 \geq 55) \vee (l2 \leq 35)): & -1 \\ \neg" \wedge (l1 \leq 35) \wedge \neg((l2 \geq 55) \vee (l2 \leq 35)): & -0.5 \\ \neg" \wedge \neg((l1 \geq 55) \vee (l1 \leq 35)) \wedge ((l2 \geq 55) \vee (l2 \leq 35)): & -1 \\ otherwise: & 0 \end{cases},$$

For the negative transfer case, we set $T1 = (55, 75), T2 = (55, 75)$ The reward functions are:

$$R_s(l1, l2) = \begin{cases} ((l1 \geq 55) \vee (l1 \leq 35)) \wedge ((l2 \geq 55) \vee (l2 \leq 35)): & -2 \\ \neg" \wedge ((l1 \geq 55) \vee (l1 \leq 35)) \wedge \neg((l2 \geq 55) \vee (l2 \leq 35)): & -1 \\ \neg" \wedge \neg((l1 \geq 55) \vee (l1 \leq 35)) \wedge ((l2 \geq 55) \vee (l2 \leq 35)): & -1 \\ otherwise: & 0 \end{cases},$$

$$R_t(l1, l2) = \begin{cases} ((l1 \geq 75) \vee (l1 \leq 55)) \wedge ((l2 \geq 75) \vee (l2 \leq 55)): & -2 \\ \neg" \wedge ((l1 \geq 75) \vee (l1 \leq 55)) \wedge \neg((l2 \geq 75) \vee (l2 \leq 55)): & -1 \\ \neg" \wedge \neg((l1 \geq 75) \vee (l1 \leq 55)) \wedge ((l2 \geq 75) \vee (l2 \leq 55)): & -1 \\ otherwise: & 0 \end{cases},$$

Pick-and-Place

The Pick-and-Place domain consists of two continuous position variables x and y. We set ranges for both variables as $x \in [0, 10]$ and $y \in [0, 10]$. Values outside of the range are clipped to the range. We use a Boolean variable $b \in \{True, False\}$ to indicate if the agent is holding an item. The agent automatically picks up an item if it is within an item region. We bound the item region via a boundary tuple $I = (I_{min_x}, I_{max_x}, I_{min_y}, I_{max_y})$. Similarly, we define boundaries for two goal regions via $G1 = (G1_{min_x}, G1_{max_x}, G1_{min_y}, G1_{max_y})$ and $G2 = (G2_{min_x}, G2_{max_x}, G2_{min_y}, G2_{max_y})$. A reward of 1 will be given to the agent if it is holding an item and is within one of the goal region. The agent has four discrete actions $A = \{m_{east}, m_{west}, m_{north}, m_{south}\}$ that can move along the x and y axis with an increment of 1. The agent can only take one action at a time. The transition and reward functions are shown below:

$$P(b'|b,x,y) = \begin{cases} x \geq I_{min_x} \wedge x \leq I_{max_x} \wedge y \geq I_{min_y} \wedge y \leq I_{max_y} : & 1.0 \\ \neg" \wedge x \geq G1_{min_x} \wedge x \leq G1_{max_x} \wedge y \geq G1_{min_y} \wedge y \leq G1_{max_y} : & 0.0 \\ \neg" \wedge x \geq G2_{min_x} \wedge x \leq G2_{max_x} \wedge y \geq G2_{min_y} \wedge y \leq G2_{max_y} : & 0.0 \\ \neg" \wedge b = True : & 1.0 \\ otherwise : & 0.0 \end{cases},$$

$$x' = \begin{cases} m_{east} : & \min(\max(0, x+1), 10) \\ m_{west} : & \min(\max(0, x-1), 10) \\ otherwise : & x \end{cases}, \quad y' = \begin{cases} m_{north} : & \min(\max(0, y+1), 10) \\ m_{south} : & \min(\max(0, y-1), 10) \\ otherwise : & y \end{cases},$$

$$R(b,x,y) = \begin{cases} b \wedge (x \geq G1_{min_x} \wedge x \leq G1_{max_x} \wedge y \geq G1_{min_y} \wedge y \leq G1_{max_y}) : & 1 \\ \neg" \wedge b \wedge (x \geq G2_{min_x} \wedge x \leq G2_{max_x} \wedge y \geq G2_{min_y} \wedge y \leq G2_{max_y}) : & 1 \\ otherwise : & 0 \end{cases},$$

For the positive transfer case, we set $G1 = (0, 2, 8, 10), G2 = (0, 2, 0, 2)$ we set increase the reward for dropping off an item to $G2$ to 2. The reward functions are:

$$R_s(b,x,y) = \begin{cases} b \wedge (x \geq 0 \wedge x \leq 2 \wedge y \geq 8 \wedge y \leq 10) : & 1 \\ \neg" \wedge b \wedge (x \geq 0 \wedge x \leq 2 \wedge y \geq 0 \wedge y \leq 2) : & 1 \\ otherwise : & 0 \end{cases},$$

$$R_t(b,x,y) = \begin{cases} b \wedge (x \geq 0 \wedge x \leq 2 \wedge y \geq 8 \wedge y \leq 10) : & 1 \\ \neg" \wedge b \wedge (x \geq 0 \wedge x \leq 2 \wedge y \geq 0 \wedge y \leq 2) : & 2 \\ otherwise : & 0 \end{cases},$$

For the negative transfer case, we set $G1 = (0, 2, 8, 10), G2 = (8, 10, 0, 2)$ and invert the reward values:

$$R_s(b,x,y) = \begin{cases} b \wedge (x \geq 0 \wedge x \leq 2 \wedge x \geq 8 \wedge x \leq 10) : & 1 \\ \neg" \wedge b \wedge (x \geq 8 \wedge x \leq 10 \wedge x \geq 0 \wedge x \leq 2) : & -1 \\ otherwise : & 0 \end{cases},$$

$$R_t(b,x,y) = \begin{cases} b \wedge (x \geq 0 \wedge x \leq 2 \wedge x \geq 8 \wedge x \leq 10) : & -1 \\ \neg" \wedge b \wedge (x \geq 8 \wedge x \leq 10 \wedge x \geq 0 \wedge x \leq 2) : & 1 \\ otherwise : & 0 \end{cases},$$

Power Generation

We model 3 power producers $(p1, p2, p3)$ that act cooperatively to meet a daily demand d. The demand d depends on a Bernoulli random variable b that adds additional demands on top of a base demand amount. We model the respective product costs for the 3 producers as $C = (c1, c2, c3)$. The income per unit is a constant value for all producers. There is a large penalty for unfulfilled demand. For each power producer, the agent can increase or decrease production level via action $A = \{d1, d2, d3, i1, i2, i3\}$ by an increment of 1. The agent can only take one action at a time. The goal of the agent is to find the optimal production level $l1, l2, l3$ for each power producer, bounded by the range $[0, 10]$. The transition and reward functions are shown below:

$$P(b) \sim Bernoulli(0.5)$$
$$d = 25 + b * 5$$

$$l1' = \begin{cases} i1: & \min(\max(0, l1 + 1), 10) \\ d1: & \min(\max(0, l1 - 1), 10) \\ otherwise: & l1 \end{cases}$$

$$l2' = \begin{cases} i2: & \min(\max(0, l2 + 1), 10) \\ d2: & \min(\max(0, l2 - 1), 10) \\ otherwise: & l2 \end{cases}$$

$$l3' = \begin{cases} i3: & \min(\max(0, l3 + 1), 10) \\ d3: & \min(\max(0, l3 - 1), 10) \\ otherwise: & l3 \end{cases},$$

$$R(l1, l2, l3) = \begin{cases} (l1 + l2 + l3) \geq d: & d \cdot 8 - (l1 \cdot 1 + l2 \cdot 2 + l3 \cdot 3) \\ otherwise: & -50 + (l1 + l2 + l3) \cdot 8 - (l1 \cdot 1 + l2 \cdot 2 + l3 \cdot 3) \end{cases},$$

For the positive transfer case, we set cost values $C_s = (5, 5, 5)$ for the source domain and $C_t = (7, 5, 5)$ for the target domain.

$$R_s(l1, l2, l3) = \begin{cases} (l1 + l2 + l3) \geq d: & d \cdot 8 - (l1 \cdot 5 + l2 \cdot 5 + l3 \cdot 5) \\ otherwise: & -50 + (l1 + l2 + l3) \cdot 8 - (l1 \cdot 5 + l2 \cdot 5 + l3 \cdot 5) \end{cases},$$

$$R_t(l1, l2, l3) = \begin{cases} (l1 + l2 + l3) \geq d: & d \cdot 8 - (l1 \cdot 7 + l2 \cdot 5 + l3 \cdot 5) \\ otherwise: & -50 + (l1 + l2 + l3) \cdot 8 - (l1 \cdot 7 + l2 \cdot 5 + l3 \cdot 5) \end{cases},$$

For the negative transfer case, we set cost values $C_s = (5, 5, 5)$ for the source domain and $C_t = (7, 7, 7)$ for the target domain.

$$R_s(l1, l2, l3) = \begin{cases} (l1 + l2 + l3) \geq d: & d \cdot 8 - (l1 \cdot 5 + l2 \cdot 5 + l3 \cdot 5) \\ otherwise: & -50 + (l1 + l2 + l3) \cdot 8 - (l1 \cdot 5 + l2 \cdot 5 + l3 \cdot 5) \end{cases},$$

$$R_t(l1, l2, l3) = \begin{cases} (l1 + l2 + l3) \geq d: & d \cdot 8 - (l1 \cdot 7 + l2 \cdot 7 + l3 \cdot 7) \\ otherwise: & -50 + (l1 + l2 + l3) \cdot 8 - (l1 \cdot 7 + l2 \cdot 7 + l3 \cdot 7) \end{cases},$$