Effective Reinforcement Learning through State Abstraction

Author
David Abel

Doctoral Advisor
Dr. Michael L. Littman

Ph.D Thesis Proposal

November 8, 2018
Abstract

Reinforcement Learning presents a challenging problem: agents must generalize experiences, efficiently explore their world, and learn from feedback that is sparse and delayed, all under a limited computational budget. Abstraction is essential to all of these endeavors. Through abstraction, agents can form concise models of both their surroundings and behavior, enabling effective decision making in diverse and complex environments. In this work, we characterize the role of state abstraction in reinforcement learning, offering three desiderata clarifying what it means for a state abstraction to be useful, and prove when and how a state abstraction can satisfy these desiderata. Our primary contributions develop theory for state abstractions that can 1) preserve near-optimal behavior, 2) be learned and computed efficiently, and 3) lower planning or learning time. Collectively, these results provide a partial path toward abstractions that are guaranteed to minimize the complexity of decision making while still retaining near-optimality. We close by discussing the road forward, which focuses on an information theoretic paradigm for analyzing abstraction, and a framework for using state abstraction to construct hierarchies that adhere to the introduced desiderata.

Thesis: Through computational complexity, information theory, and reinforcement learning, we find efficient algorithms for constructing state abstractions that minimize the complexity of decision making while retaining near-optimality.
Contents

1 Introduction ............................. 4
   1.1 Desiderata .......................... 5

2 Reinforcement Learning Background .... 7
   2.1 Markov Decision Processes .......... 7
   2.2 Reinforcement Learning .............. 8

3 Abstraction Background ............... 10
   3.1 State Abstraction .................... 10
       3.1.1 Prior Work ........................ 12
   3.2 Action Abstraction .................. 16
   3.3 Satisfying the Desiderata .......... 20
       3.3.1 Satisfying One or Two Desiderata .......................... 22
       3.3.2 Satisfying All Three .......... 23

4 Completed Work ........................ 24
   4.1 Approximate State Abstraction ...... 24
       4.1.1 Optimal Q Function: $\phi_{Q^*,\varepsilon}$ ....... 26
       4.1.2 Model Similarity: $\phi_{\text{model},\varepsilon}$ ......... 26
       4.1.3 Boltzmann over Optimal Q ............ 26
       4.1.4 Multinomial over Optimal Q: $\phi_{\text{mult},\varepsilon}$ .... 27
   4.2 State Abstractions for Lifelong RL 28
       4.2.1 Transitive Abstractions ............ 29
       4.2.2 PAC Abstractions .................. 31
       4.2.3 Experiments ...................... 33
   4.3 State Abstraction as Compression .. 36
       4.3.1 Information Theory Background .... 36
       4.3.2 State Abstraction as Compression 40
       4.3.3 IB as an Upper Bound on J ....... 41
       4.3.4 Relating KL and $H$, $V$ and $|S_\phi|$ ....... 41
       4.3.5 Experiments ...................... 42
       4.3.6 Related Work ..................... 45
   4.4 Action Abstractions that Minimize Planning Time 46
       4.4.1 Point options ..................... 47
       4.4.2 Complexity Results ................ 48
       4.4.3 Generalizations of MIMO and MOMI 49
       4.4.4 Approximation Algorithms ......... 49
       4.4.5 A-MIMO ......................... 50
       4.4.6 A-MOMI .......................... 51
       4.4.7 Experiments ...................... 52
5 Future Work

5.1 Simple Hierarchical RL .............................................................. 55
5.2 New Multi-Time Model ............................................................... 57
5.3 Abstraction-as-Compression 2.0 .................................................. 58
5.4 State Abstraction with Continuous State ................................. 60
5.5 Dynamic State Abstraction ......................................................... 61
5.6 Timeline ..................................................................................... 62
1 Introduction

Reinforcement Learning (RL) poses a challenging problem. Agents must learn about their environment through high-dimensional and often noisy observations while receiving sparse and delayed evaluative feedback. An intelligent agent’s ability to understand its surroundings well enough to support effective decision making under these conditions is a remarkable feat, and a hallmark of intelligent behavior.

Abstraction captures precisely this ability—the formation of a world model that is both sufficiently compressed so as to facilitate quick inference, while still detailed enough to be useful for solving any problems encountered during the agent’s lifetime [51]. An example of an apparently useful abstraction is pictured in Figure 1—with a smaller model that retains the same essential information, an agent can carry out more computation, explore less, make more general inferences, and predict further into the future.

Abstraction can be divided into three categories: (1) state abstraction, (2) action abstraction, and (3) hierarchical abstraction.

Work on state abstraction [90, 3, 38, 47] investigates methods for treating similar configurations of the environment as identical, or translating between similar experiences or useful features. State abstraction can be useful for many things, but perhaps most directly, it can make planning and exploration more efficient. Solving for optimal behavior in MDPs in a planning setting P-Complete in the size of the state space [109, 92]. Similarly, many RL algorithms for solving MDPs require samples polynomial in the size of the state space [129]. Although polynomial runtime or sample complexity may seem like a reasonable constraint, the size of the state space of an MDP grows super-polynomially with the number of variables that characterize the domain—a result of Bellman’s curse of dimensionality. Thus, solutions polynomial in state space size are often ineffective for sufficiently complex tasks. For instance, a robot involved in a pick-and-place task might be able to employ planning algorithms to solve for how to manipulate some objects into a desired configuration in time polynomial in the number of states, but the number of states it must consider grows exponentially with the number of objects with which it is working [2]. Much of this document is dedicated to state abstraction, though, the other two forms of abstraction are naturally relevant to our inquiry.

Action abstractions, often referred to as skills, options [134], temporal abstractions, or macro-actions [54], can serve as mechanisms for simplifying tasks [79, 83, 81]. As their name suggests,
action abstractions consolidate information over action executions, contrasted with state abstractions that group similar instantaneous experiences. Action abstractions typically simplify problems by empowering the action space, thereby enabling agents to probe more deeply into the future than low-level actions. State and action abstraction can jointly coalesce into hierarchical representations [20, 79, 32, 68, 55, 87, 18, 99, 110, 84], dating as far back as ABSTRIPS [118], and later Feudal Reinforcement Learning [35], Hierarchical Task Networks [44], Hierarchies of Abstract Machines [111], and the MAXQ framework [40]. Hierarchies offer multiple levels of representation, each capturing a different degree of detail about the environment. Riesenhuber and Poggio [116] demonstrate that hierarchies are a central piece of human processing and representation, and so their application as a computational tool for machine learning and artificial intelligence more generally has been of significant interest.

1.1 Desiderata

In an effort to unify existing results and provide structure for future research, we propose three desiderata that characterize what it means for an abstraction to be effective in RL:

(D1) Efficient Decision Making: Enable fast planning and efficient learning.

(D2) Near-Optimality: Solutions produced from the abstracted model should be useful enough for solving relevant problems.

(D3) Efficient-Construction: Creating the abstractions should not require unrealistic statistical or computational budgets.

Our first desiderata, Efficient Decision Making suggest that algorithms for reinforcement learning, when paired with abstractions, should succeed under limited computational and statistical budgets, or enable the right kinds of tradeoffs between slow-but-thorough and quick-but-lossy decision making. One approximation of this property is that abstractions produce compact representations—we will explore connections between the compactness of induced representations and the efficiency with which models can be manipulated in Section 4.3. Near-Optimality states that abstractions ought to produce representations that preserve some notion of effectiveness in gathering reward: solutions to the problem resulting from the abstraction ought to achieve a near-optimal solution to the original problem (demonstrated either theoretically or empirically). In brief abstractions should be useful, where usefulness is measured in terms of how effectively an agent can solve a problem when using this representation. Lastly, Efficient-Construction an abstraction ought to rely on information that is easier to compute than the solution to the MDP(s) of interest. This final criteria can be summarized as saying that abstractions should be easy to obtain. Together, they advance the following thesis:

| (D3) | Good abstractions can be formed with reasonable statistical and computational budgets while enabling efficient planning and learning that yields near-optimal behavior. |
| (D1) | (D2) |
As we will show, any one of these desiderata is easy to satisfy in isolation. Surprisingly, any pair of the desiderata taken together are trivially satisfied, too. (Proved in Section 3.3.1.) However, satisfying all three remains elusive.

In this document, we present new theory for state abstraction in RL. We start by introducing a simple recipe for studying state abstraction, and subsequently showcase how to satisfy the desiderata. Our main contributions develop theory for state abstractions that can (1) preserve near-optimal behavior, (2) be computed efficiently, (3) can lower planning or learning time, (4) be transferred across similar tasks, and (5) induce a trade-off between compression and performance. At a high level, our contributions are as follows:

<table>
<thead>
<tr>
<th>Summary of Contributions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- A framework for approximate state abstraction that preserve near-optimal behavior in arbitrary MDPs [3].</td>
</tr>
<tr>
<td>- Characterizing and computing state abstractions for lifelong RL that preserve near-optimal behavior [5].</td>
</tr>
<tr>
<td>- Computing action abstractions/hierarchies from state abstractions [4].</td>
</tr>
<tr>
<td>- Information theoretic view for understanding abstraction as compression.</td>
</tr>
<tr>
<td>- Characterizing the computational complexity of finding options that minimize planning time, and an approximation algorithm.</td>
</tr>
</tbody>
</table>

The rest of the paper is outlined as follows. In the next section, I provide background on RL and planning. Section 3 provides necessary background on abstraction. I then summarize the completed work in Section 4. Lastly, in Section 5, I suggest a road map for future work.
2 Reinforcement Learning Background

RL defines the problem of an agent learning to make decisions to maximize reward through interaction alone, pictured in Figure 2.

2.1 Markov Decision Processes

We make the standard assumption that the environment can be accurately modeled by a Markov Decision Process [113]. An MDP is a highly general method of representing an environment:

<table>
<thead>
<tr>
<th>Definition 1 (Markov Decision Process (MDP)): A Markov Decision Process (MDP) is a six-tuple: $(S, A, T, R, \gamma, \rho_{\text{init}})$, where:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- $S$ is a possibly infinite set of states, also called the state space.</td>
</tr>
<tr>
<td>- $A$ is a possibly infinite set of actions, also called the action space.</td>
</tr>
<tr>
<td>- $T$ denotes $T(s'</td>
</tr>
<tr>
<td>- $R : S \times A \rightarrow [\text{RMin}, \text{RMax}]$ denotes the real valued reward received by the agent for occupying state $s$ and applying action $a$.</td>
</tr>
<tr>
<td>- $\gamma \in [0, 1)$ is a discount factor that defines how much the agent prefers immediate rewards over future rewards (the agent prefers immediate over future rewards as $\gamma$ decreases).</td>
</tr>
<tr>
<td>- $\rho_{\text{init}}$ is a probability distribution on states, indicating the probability of starting in each state.</td>
</tr>
</tbody>
</table>

MDPs generalize Markov chains by allowing an agent to influence the state distribution and

---

1Recent work has considered the General Reinforcement Learning problem in non-Markovian, non-ergodic settings [59, 142]. In this work, we focus only on agents learning in Markovian environments, though we note that there is interesting and important work to be done in clarifying the role of abstraction in the partially observable [71] and general RL [89] settings.

---

![Figure 2: The Reinforcement Learning Problem.](image-url)
associated reward, \( \mathcal{R}(s, a) \) according to the agent’s behavior. The solution to an MDP is called a policy, a possibly probabilistic function from states to actions denoted \( \pi : \mathcal{S} \rightarrow \mathcal{A} \).

In this work we will also consider \textit{lifelong} learning, in which the agent’s environment is modeled as a distribution over MDPs, \( D \). Prior work has studied various instantiations of these distributions. To maintain generality, we give the following treatment of MDP distributions:

**Definition 2 (MDP Distribution):** We let \( D \) denote a distribution over \( (\mathcal{R}, \mathcal{T}, \rho_{\text{init}}) \) triples, with \( \mathcal{S}, \mathcal{A}, \) and \( \gamma \) fixed.

### 2.2 Reinforcement Learning

The RL problem is defined as follows:

**Definition 3 (Reinforcement Learning Problem):** A Reinforcement Learning (RL) agent interacts with an MDP \( M \) via the repetition of the following three steps:

1. The agent receives a state \( s \) and a reward \( r \).
2. The agent learns from this interaction and outputs an action, \( a \).
3. The MDP outputs the next state, \( s' \), determined by \( s' \sim \mathcal{T}(s' \mid s, a) \), and reward \( r' \) determined by \( \mathcal{R}(s, a) \).

The agent’s goal is to make decisions to maximize discounted reward:

\[
\sum_{t=1}^{\infty} r_t \gamma^t
\]  

The objective of an RL agent is to solve for the policy, \( \pi \), that maximizes its expected discounted reward from any state. We denote the expected discounted reward for taking action \( a \in \mathcal{A} \) and then following policy \( \pi \) from state \( s \) forever after as the \textit{action-value} function, \( Q^\pi(s, a) \), defined by the Bellman Equation introduced by Bellman [22] as:

\[
Q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s'} \mathcal{T}(s' \mid s, a) \sum_{a' \in \mathcal{A}} Q^\pi(s', a') \pi(a' \mid s').
\]  

The value of a state under a given policy is defined by the \textit{value} function:

\[
V^\pi(s) = \sum_{a \in \mathcal{A}} Q^\pi(s, a) \pi(a \mid s).
\]  

We denote the value (\( V \)) and action-value (\( Q \)) functions under the optimal policy as \( V^* \) and \( Q^* \) respectively, which are determined by applying the max operator to the Bellman Equation:

\[
V^*(s) = \max_a \left( \mathcal{R}(s, a) + \gamma \sum_{s'} \mathcal{T}(s' \mid s, a) V^*(s') \right)
\]  

Since the codomain of \( \mathcal{R} \) is the real valued interval \([\mathcal{R}_{\text{MIN}}, \mathcal{R}_{\text{MAX}}]\), we let \( Q_{\text{MAX}} = V_{\text{MAX}} \leq \frac{\mathcal{R}_{\text{MAX}}}{1-\gamma} \), denote an upper bound on the total value achievable.
For further background on Markov Decision Process, see Puterman [113], and for more background on Reinforcement Learning, see Kaelbling et al. [70] and Sutton and Barto [133].
3 Abstraction Background

We first offer the necessary background on abstraction, divided into state and action abstraction.

3.1 State Abstraction

In an MDP, a state fully describes the current configuration of the environment. States are traditionally enumerated, in the sense that the true state space $\mathcal{S}$ is the set containing the numbers $1, 2, \ldots, |\mathcal{S}|$. The actions, $\mathcal{A}$, change the MDP to different states according to the transition dynamics defined by $\mathcal{T}$. However, this enumeration view on states is somewhat limiting; a key aspect of intelligent behavior is the awareness of structural similarity in one’s experiences. It is precisely these similarities that enable perceptual and representational ontologies like objects, properties, relations, and universals. To facilitate this more general notion of state, an MDP state is often defined as a vector of variables or features, sometimes formalized as a Factored MDP [78, 52], similar to the usual learning problems found in supervised and unsupervised learning. Other types of MDPs have been introduced that leverage implicit ontological structure, such as Relational MDPs [73, 50], and Object-Oriented MDPs [42], which explicitly carve the world into objects, their classes, and functions thereon.

Regardless of the mechanism for representing states, the goal of state abstraction is to reduce the size of the state space by grouping together similar states in a way that doesn’t change the underlying problem being solved. That is, a state abstraction function maps each true environmental state into an abstract state:

**Definition 4 (State Abstraction):** A state abstraction $\phi : \mathcal{S} \rightarrow \mathcal{S}_\phi$ maps each true environmental state, $s \in \mathcal{S}$ into an abstract state, $s_\phi \in \mathcal{S}_\phi$, where typically $|\mathcal{S}| << |\mathcal{S}_\phi|$.

The abstracted state corresponds to the agent’s model of the current configuration of the environment: it is often not perfect in that choosing to represent the current world state in terms of the abstracted state necessarily disregards some information. Determining what information to discard is the central question behind the theory of abstraction: how do intelligent agents come up with the right abstract understanding of the worlds they inhabit? We study this question through the above introduced state abstraction functions. Why study such a broad question using such a specific, and simple, formalism? Simply: from a methodological perspective, it is important to first attend to our question’s simplest unanswered form. That way, we ensure we build a firm foundation upon which subsequent inquiry can take place.

Using these new abstracted states, we can construct a full MDP, which we denote the abstract MDP, $M_\phi$. The abstract MDP is defined by a weighting function [90], $w : \mathcal{S} \rightarrow [0, 1]$, such that:

$$\forall s_\phi \in \mathcal{S}_\phi : \sum_{s \in \phi^{-1}(s_\phi)} w(s) = 1,$$

and a state abstraction function $\phi$. Together, the remaining constituents of the abstract MDP can be defined:
**Definition 5 (Abstract Reward Function):** The abstract reward function, $R_\phi : S_\phi \times A \rightarrow [0,1]$, is a weighted sum of the rewards of each of the ground states that map to the same abstract state:

$$R_\phi(s_\phi, a) = \sum_{s \in \phi^{-1}(s_\phi)} R(s, a)w(s). \quad (6)$$

We let $\phi^{-1}(s_\phi)$ denote the projection from an abstract state to the set of environmental states $s_1, \ldots, s_n \in S$ that belong to the abstract state.

**Definition 6 (Abstract Transition Function):** The abstract transition function, $T_\phi : S_\phi \times A \times S_\phi \rightarrow [0,1]$, is a weighted sum of the transitions of each of the ground states that map to the same abstract state:

$$T_\phi(s_\phi, a, s'_\phi) = \sum_{s \in \phi^{-1}(s_\phi)} \sum_{s' \in \phi^{-1}(s'_\phi)} T_G(s, a, s')w(s). \quad (7)$$

The alternative ontological MDPs mentioned previously (such as OO-MDPs) can, in some sense, be thought of as making a certain assumption about what can be abstracted. The metaphor is relatively brittle, though, as notions of state similarity are lacking in the notion of aggregation introduced above, as aggregation is an all-or-nothing operation. Some prior work has investigated MDP and state distance metrics, which will be discussed later in the section.

As a motivating example, suppose an agent is placed into a wide hallway with the goal of reaching the exit at the far end of the hall. A traditional representation for this problem might yield a Cartesian grid: the agent has an $x$ and a $y$ coordinate, and the up, down, left, and right actions, and must navigate until its $y$ coordinate is sufficiently large (and so has reached the exit of the hallway). This problem is pictured in Figure 3.

How might one abstract states? Of course, there are many possible groupings. If we suppose the agent coordinates are the natural numbers, then a grid of size $M \times N$ yields the absurdly large:

$$(N \times M)^{N \times M}, \quad (8)$$

number of possible groupings. However, given the inherent structure of the problem, the agent’s $x$ coordinate is actually irrelevant for computing optimal behavior, or for computing the value function $V$, and action-value function, $Q$, or for learning any of these quantities. Thus, consider the function $\phi$ that projects the ground state to an abstract state that only tracks the $y$ coordinate. So all ground states with the same $y$ coordinate get treated as the same abstract state. In this way, the true state space of size $N \times M$ down to an abstract state space of size equivalent to the length of the hallway, $M$. This problem is adapted from the Upworld task introduced by our earlier work [3].

The key technical question of state abstraction is to determine which groupings are effective; much of previous literature investigates when, how, and why state abstractions are useful in the context of different sequential decision making problems. The technique is based on early applications of aggregation to dynamic programming, as in Bertsekas and Castanon [24], Mendelssohn [101] and Reyman and van der Wal [115].
A natural way to organize the space of possible abstractions is to define types of state abstraction functions. The initial characterization of the space of state abstractions was done by Li et al. [90]. In a state abstraction type, we associate abstract state clusters with some property that must hold between all the ground states in the cluster. Formally, a state abstraction type is defined with respect to a two-argument predicate on state pairs:

**Definition 7** (State Abstraction Type): A state abstraction type is a collection of functions $\phi: S \rightarrow S_\phi$ associated with a fixed predicate on state pairs:

$$p: S \times S \rightarrow \{0, 1\},$$

such that when $\phi$ clusters state pairs, the predicate must be true for that state pair:

$$\phi(s_1) = \phi(s_2) \implies p(s_1, s_2).$$

Candidate types introduced in previous works are shown in Table 1, along with many notable properties we will discuss shortly.

### 3.1.1 Prior Work

We next summarize existing work in state abstraction as applied to RL.

The earliest known instance of state abstraction is by Dean et al. [38] and Dean and Givan [37], who introduce the approximate bisimulation metric for partitioning an MDP's state space into clusters of states whose transition model and reward function are within $\varepsilon$ of each other. They develop an algorithm called Interval Value Iteration (IVI) that converges to the correct bounds on a
family of abstract MDPs called Bounded MDPs. Bounded MDPs summarize the space of possible MDPs the agent could be in, given its current knowledge of the MDP. This work builds on the earlier work in approximating dynamic programs, by Whitt [145].

Several approaches build on Dean et al. [38]. Ferns et al. [47] and Ferns et al. [48] investigated state similarity metrics for MDPs; they bounded the value difference of ground states and abstract states for several bisimulation metrics that induce an abstract MDP. This differs from our work which develops a theory of abstraction that bounds the suboptimality of applying the optimal policy of an abstract MDP to its ground MDP, covering four types of state abstraction, one of which closely parallels bisimulation. Even-Dar and Mansour [45] analyzed different distance metrics for several bisimulation metrics that induce an abstract MDP. This differs from our work which develops a theory of abstraction that bounds the suboptimality of applying the optimal policy of an abstract MDP to its ground MDP, covering four types of state abstraction, one of which closely parallels bisimulation. Even-Dar and Mansour [45] analyzed different distance metrics used in identifying state space partitions subject to \( \varepsilon \)-similarity, also providing value bounds (their Lemma 4) for \( \varepsilon \)-homogeneity subject to the \( L_\infty \) norm, which parallels our Claim 2. Ortner [105] developed an algorithm for learning partitions in an online setting by taking advantage of the confidence bounds for \( T \) and \( R \) provided by the Bayesian RL algorithm, Upper Confidence bounds for Reinforcement Learning (UCRL) [16].

Li et al. [90] offer the first general framework for state abstraction in MDPs, which makes up much of the notation and perspectives on state abstraction introduced here. They highlight five types of state aggregation functions, inspired by existing methods for state aggregation in MDPs from previous literature. These five types are shown in Figure 1. To quickly summarize, the predicates are of increasing degree of strictness, requiring that various aspects of the state that relate to optimal behavior must be equal across states. For instance, one such type, the \( \phi_{\pi^*} \) type, says that any two states that are place in the same abstract state must have the same optimal action, \( a^* \).

The full set of five state abstraction types they introduce are as follows:

1. \( \phi_{\text{model}} \): The reward in the ground, and the resulting abstract transition functions of states are the same,
   \[
   \phi(s_1) = \phi(s_2) \implies \forall a : R(s_1, a) = R(s_2, a) \wedge \forall a : \sum_{s' \in \phi^{-1}(s)} \mathcal{T}(s_1, a, s') = \sum_{s' \in \phi^{-1}(s)} \mathcal{T}(s_2, a, s')
   \]
   (11)

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{Name} & \text{Predicate} & \text{Value Loss} & \text{Transitive} \\
\hline
\phi_Q^* & \max_a |Q^*(s_1, a) - Q^*(s_2, a)| = 0 & 0 & \text{yes} \\
\phi_a^* & a_1^* = a_2^* \wedge V^*(s_1) = V^*(s_2) & 0 & \text{yes} \\
\phi_{\pi^*} & \pi^*(s_1) = \pi^*(s_2) & 0 & \text{yes} \\
\phi_Q^\varepsilon & \max_a |Q^*(s_1, a) - Q^*(s_2, a)| \leq \varepsilon & 2\varepsilon \frac{\max R}{(1-\gamma)^2} & \text{no} \\
\phi_{mult} & \max_a \left| \frac{Q^*(s_1, a)}{\sum_b Q^*(s_1, b)} - \frac{Q^*(s_2, a)}{\sum_b Q^*(s_2, b)} \right| \leq \varepsilon & 2\varepsilon \frac{|\max R + k|}{(1-\gamma)^2} & \text{no} \\
\phi_{bolt} & \max_a \left| \frac{Q^*(s_1, a)}{\sum_b e^{Q^*(s_1, b)}} - \frac{Q^*(s_2, a)}{\sum_b e^{Q^*(s_2, b)}} \right| \leq \varepsilon & 2\varepsilon \frac{|\max R + \varepsilon + k|}{(1-\gamma)^2} & \text{no} \\
\phi_{Q_d^*} & \forall a : \left[ \frac{Q^*(s_1, a)}{\max} \right] = \left[ \frac{Q^*(s_2, a)}{\max} \right] & 2\varepsilon \frac{\max R}{(1-\gamma)^2} & \text{yes} \\
\hline
\end{array}
\]

Table 1: A few existing state abstraction types [90, 3] and our transitive \( Q_d^* \) abstraction type.
2. $\phi_{Q^*}$: The $Q$ function under any policy is the same.

$$\phi(s_1) = \phi(s_2) \implies \forall_{a,\pi} Q^\pi(s_1, a) = Q^\pi(s_2, a) \quad (12)$$

3. $\phi_{Q^*}$: The $Q$ values for each action are the same.

$$\phi(s_1) = \phi(s_2) \implies \max_a |Q^*(s_1, a) - Q^*(s_2, a)| = 0 \quad (13)$$

4. $\phi_{a^*}$: The optimal action is the same and the $Q$ value is the same for that action in each state.

$$\phi(s_1) = \phi(s_2) \implies \arg\max_x Q^*(s_1, x) = \arg\max_y Q^*(s_2, y) \land |V^*(s_1) - V^*(s_2)| = 0 \quad (14)$$

5. $\phi_{\pi^*}$: The optimal policy chooses the same action in each state.

$$\phi(s_1) = \phi(s_2) \implies \arg\max_x Q^*(s_1, x) = \arg\max_y Q^*(s_2, y). \quad (15)$$

In addition to introducing the general framework for state abstraction, they present several new results. First, they introduce the ordering operator, $\phi_X \trianglerighteq \phi_Y$, which states that any instance of type $\phi_X$ is also an instance of $\phi_Y$. They prove the following ordering:

**Theorem 1.** (Theorem 2 from Li et al. [90]) For any MDP, $\phi_0 \trianglerighteq \phi_{\text{model}} \trianglerighteq \phi_{Q^\pi} \trianglerighteq \phi_{Q^*} \trianglerighteq \phi_{a^*} \trianglerighteq \phi_{\pi^*}$

That is, any element of $\phi_{\text{model}}$ is also an element of $\phi_{Q^*}$. This result is particularly useful for understanding planning performance in the abstracted models and for clarifying convergence behavior. The next result they present is an optimality result. That is, a result that addresses (D2): Near-Optimality.

**Theorem 2.** (Theorem 3 from Li et al. [90]) Any state abstraction function of type $\phi_{\text{model}}, \phi_{Q^\pi}, \phi_{Q^*}$ or $\phi_{a^*}$ preserves the optimal policy. That is, if the abstraction $\phi$ is any of the above types, then the value loss is $0$. Conversely, $\phi_{\pi^*}$ does not always preserve the optimal policy.

The final result from this paper shows that certain abstraction functions preserve the $Q$-learning convergence guarantees, so long as the traditional step-size decay properties hold:

$$\sum_t \alpha_t = \infty, \quad \sum_t \alpha_t^2 < \infty. \quad (16)$$

Under these conditions, they introduce the following result:

**Theorem 3.** (Theorem 4 from Li et al. [90]) If every state-action pair is visited infinitely often, then:

1. $Q$-learning converges to the optimal $Q$ function under $\phi_{\text{model}}, \phi_{Q^\pi}$ and $\phi_{Q^*}$.

2. $Q$-learning with $\phi_{a^*}$ does not necessarily converge.

3. $Q$-learning with $\phi_{\pi^*}$ can converge to a $Q$ function that results in a non-optimal greedy policy in the true MDP.
These results give further clarity on state abstractions as they relate to (D2): in what sense can we actually solve the environmental problem of interest? If our algorithms don’t converge to the desired solution (or a reasonable approximation thereof), then surely (D2) is not satisfied.

This paper is followed directly by Walsh et al. [143]. Here they investigate the power of state abstraction functions for facilitating efficient transfer learning. That is, in the multitask RL setting, they seek to maximize the speedup ratio:

\[
\frac{\mathbb{E}_{M \sim D}[T(M)]}{\mathbb{E}_{M \sim D, \tilde{M} \sim D}[T(M | \tilde{M})]},
\]

where \(T(M)\) denotes the time needed to find a good policy on MDP \(M\) given that information may be transferred from the collection of \(n\) source MDPs, \(\tilde{M}\), already sampled from \(D\). The central contribution of the paper is the General Abstraction Transfer Algorithm (GATA), and the translation of optimality and convergence results of state abstractions to the algorithm. GATA effectively targets the multitask RL setting introduced in the previous section wherein the agent samples an environmental MDP from a fixed but unknown distribution \(D\). In GATA, with a small sampling of MDPs, the agent computes a candidate state abstraction \(\phi\) based on the sampled MDPs and uses it to reduce learning time on future MDPs.

Mandel et al. [95] focus on the exploration-exploitation dilemma in the context of state abstraction. In particular, they introduce a Bayesian method for clustering states to facilitate effective exploration while generalizing across the statespace appropriately. The core contribution is the algorithm Thompson Clustering for Reinforcement Learning (TCRL), which addresses the large space of possible state space clusterings by exploiting explicit state space structure. As discussed in the Hallway example earlier in the section, the space of clusterings is too vast to search over. TCRL narrows the search delicately to both improve learning speed and retain Bayesian regret guarantees. A slight variant of TCRL achieves regret similarly to that of Posterior Sampling for Reinforcement Learning (PSRL) [106].

Asmuth et al. [13] are of a similar flavor to the TCRL work. Here they introduce Best of Sampled Set (BOSS), a Bayesian approach to exploration in RL that easily accommodates priors for clustering states. The algorithm itself resembles PSRL: maintain a posterior on models, sample from the posterior, and use the samples to inform decision making.

Jiang [64] analyze the problem of choosing between two candidate state abstractions for model-based RL, (a direction that has also been explored for action abstraction [83]). They develop an algorithm based on statistical tests that trades of the approximation error with the estimation error of the two abstractions, yielding a loss bound on the quality of the chosen policy, which in effect, addresses D2 (Near-optimality).

Many previous works have targeted the creation of algorithms that enable state abstraction for MDPs. Andre and Russell [9] investigated a method for state abstraction in hierarchical reinforcement learning leveraging a programming language called ALISP that promotes the notion of safe state abstraction. Agents programmed using ALISP can ignore irrelevant parts of the state, achieving abstractions that maintain optimality. Dietterich [40] developed MAXQ, a framework for composing tasks into an abstracted hierarchy where state aggregation can be applied. Jong and Stone [67] introduced a method called policy-irrelevance in which agents identify (online) which state variables may be safely abstracted away in a factored-state MDP.

Hutter [61, 60] investigate state aggregation beyond the MDP setting. Hutter presents a variety
of results for aggregation functions in reinforcement learning. Most relevant to our investigation is
Hutter’s Theorem 8, which illustrates properties of aggregating states based on similar $Q$ values.
Hutter’s Theorem part (a) bounds the value difference between ground and abstraction states, and
part (b) bounds the value difference of applying the optimal abstraction policy in the ground, and
part (c) is a repetition of the comment given by Li et al. [90] that $Q^*$ abstractions preserve the
optimal value function.

State Abstraction and Planning Approximate state abstraction has also been applied to the planning problem, in which the agent is given a model of its environment and must compute a plan that satisfies some goal. Hostetler et al. [58] apply state abstraction to Monte Carlo Tree Search and expectimax search, giving value bounds of applying the optimal abstract action in the ground tree(s), similarly to our own agenda. Other work builds on these approaches, developing methods for incorporating state abstraction into Monte Carlo style planning algorithms [7, 8]. Dearden and Boutilier [39] also formalize state-abstraction for planning, focusing on abstractions that are quickly computed and offer bounded value. Their primary analysis is on abstractions that remove negligible literals from the planning domain description, yielding value bounds for these abstractions and a means of incrementally improving abstract solutions to planning problems. Jiang et al. [65] analyze a similar setting, applying abstractions to the Upper Confidence Bound applied to Trees algorithm adapted for planning, introduced by Kocsis and Szepesvári [77]. We next turn to action abstraction.

3.2 Action Abstraction
Action abstractions encompasses the orthogonal application of abstraction to MDPs: changing, removing, or empowering the action space of an agent to probe more deeply into the search space, achieve efficient planning by reasoning toward a subgoal (as opposed to the full horizon of the problem), explore in a more targeted way, or prune away irrelevant action sequences based on knowledge of action-optimality correlations. The full extent and power of action abstractions is still being understood and explored, as is the case with state abstraction. Notably, the interplay between these two abstraction types is particularly promising for capitalizing on the power of abstraction, especially for fully satisfying the three desiderata.

Of course, as with computing the space of possible state space clusterings, there are many possible modifications, including adding new long horizon sequences of actions, pruning away actions, or adding actions that specifically try to satisfy some property or subgoal. As with state abstraction, we seek a unification of existing formalisms both in terms of the ideas introduced and in terms of notation. Toward this end, we concentrate on what has become the most canonical formalism for action abstraction, Options [134]. We note that the typical definition of an option does not fully contain the space of possible action abstractions, without augmenting the traditional policy space associated with an option, such as the semi-Markov options introduced alongside regular options in Sutton et al. [134]. More on this shortly.

To introduce options, we consider an extension of the previously introduced grid world for the purpose of showcasing the role of action abstractions. This problem is taken from Sutton et al. [134] appearing under the name “Four Rooms”, and one of the canonical tasks for experimenting with action abstractions. As with the original grid world, the agent may move up, down, left, and right (possibly with some slip probability) with the goal of getting from a particular cell in the grid to another. In this case we find several walls restricting the agent’s movement that change the structure of the problem. The flow of movement throughout the environment is thought to be
Figure 4: The classical Four Rooms with two candidate goal locations (left) and augmented by Options (right).

suggestive of certain types of action abstractions, such as those that take the agent to the doorways between rooms, or that transition between the rooms directly. We now introduce options:

**Definition 8** (Option Sutton et al. [134]): An option is a triple \((\mathcal{I}, \beta, \pi)\), where:

- \(\mathcal{I} : S \to \{0, 1\}\) is a predicate on states denoting when the option is available,
- \(\beta : S \to \text{Bernoulli}(\theta)\), assigns a Bernoulli random variable to each state denoting the probability that the option terminates during execution in that state,
- \(\pi : S \to \text{Pr}(A)\)

An action abstraction, then, is effectively a collection of options (along with the machinery to translate an executing option to an environmental action):

**Definition 9** (Action Abstraction): An action abstraction is a collection of options \(\mathcal{O}\), and a function \(\omega : \mathcal{O} \times B \to A\), that maps an option and the agent’s memory, \(B \in \{0, 1\}^n\), to a primitive action.

Using an action abstraction, we define can define an abstract MDP with a new action space. That is, when the agent enters state \(s\), the agent’s currently available actions are defined by the primitive actions and the set of options for which \(\mathcal{I}\) is true:

\[
\mathcal{A}_\omega(s) = \{a : a \in A \cup o \in \mathcal{O} \land \mathcal{I}_o(s)\}.
\] (18)

Then, when the agent makes a choice of action, if its an option, the agent commits to executing the policy associated with the option until the sampled terminating condition is true for a state the agent arrives in. For example, if the termination condition has \(\theta = 0\) on all states except \(s_4\)
(in which \( \theta = 1 \)), then the agent will execute the option’s policy indefinitely until arriving in \( s_4 \). When the agent reaches \( s_4 \), the agent will stop executing the option policy, and will make its next choice of action or option. So, options facilitate action pruning of a certain form; when an option is selected, every state the agent arrives in up until termination, the actions not chosen by the option policy are effectively pruned. The resulting decision-making problem slightly loses out on the Markov property, too, as any state encountered while executing the option will induce a different policy according to which option is currently being run. The options also induce new transition and reward functions based on where the options will terminate and the trajectory taken by the option’s policy.

For simplicity, let \( \mathcal{R}_s^a \) denote \( \mathcal{R}(s,a) \), \( \mathcal{T}_{s,s'}^a \) denote \( \mathcal{T}(s' \mid s,a) \). Then, the multi-time model introduced by Sutton et al. [134] is given as:

\[
\mathcal{R}_\omega(s,a) = \mathcal{R}_s^\pi(s) + \sum_{s' \in S} (1 - \beta_{s'}(s')) \mathcal{T}_{s,s'}^\pi(s',a) \mathcal{R}_\omega(s',a)
\]

\[
\mathcal{T}_{s,x}^a = \sum_{s' \in S} [(1 - \beta_{s'}(s')) \mathcal{T}_{s',x}^\pi + \beta(s') \mathbb{1}_{s'=x}]
\]

If we combine an action abstraction \( \omega \) with a state abstraction \( \phi \), we end up with the resulting abstract reward and transition functions (for some choice of weighting function \( w \)):

\[
\mathcal{R}_{s,\phi}^a = \sum_{s \in \phi^{-1}(s_{\phi})} \mathcal{R}_\omega(s,a)w(s) \tag{19}
\]

\[
\mathcal{T}_{s,\phi}^a = \sum_{s \in \phi^{-1}(s_{\phi})} \sum_{s' \in \phi^{-1}(s'_{\phi})} \mathcal{T}_{s,x}^a w(s) \tag{20}
\]

In the Four Rooms problem, some typical options to consider are those that terminate in hallways, pictured on the right of Figure 4. In this case, we introduce two options, defined as follows:

\[
o_1 = \begin{cases}
\mathcal{I}(s) & s.x \leq 5 \wedge s.y \leq 5 \\
\beta(s) & \theta = 1 \{s = (3,6) \lor s = (6,3)\} \\
\pi(s) & \arg \max_a \{1 \{s = (3,6) \lor s = (6,3)\} + \gamma \sum_{s'} \mathcal{T}(s,a,s')V^*_1(s=(3,6)\lor s=(6,3))(s')\}
\end{cases}
\]

\[
o_2 = \begin{cases}
\mathcal{I}(s) & s.x \leq 5 \wedge s.y \geq 5 \\
\beta(s) & \theta = 1 \{s = (3,6) \lor s = (6,9)\} \\
\pi(s) & \arg \max_a \{1 \{s = (3,6) \lor s = (6,9)\} + \gamma \sum_{s'} \mathcal{T}(s,a,s')V^*_1(s=(3,6)\lor s=(6,9))(s')\}
\end{cases}
\]

The first option \( o_1 \) will initiate anywhere in the bottom left room, will only terminate when the agent arrives in one of the two hallways leaving the lower left room, and executes the policy that moves the agent to one of the hallways as quickly as possible. So, if the agent starts in state \( s_{\text{init}} = (1,1) \), the option might induce the following trajectory:

\[
(1,1), \text{right}, (2,1), \text{up}, (2,2), \text{right}, (3,2), \text{right}, (4,2), \text{right}, (5,2), \text{up}, (5,3), \text{right}, (6,3). \tag{21}
\]

For each state, the agent samples from the Bernoulli defined by \( o, \beta(s) \). If the sample comes up heads, then the agent stops following the option policy, and is again in a position to choose among its active options and primitive actions. In the trajectory above, we define the option \( o_1 \) such that
so the agent will stop executing the option at (6,3). At this point, the agent resumes its regular action selection process.

So, the question at the heart of the options framework is roughly that path prescribed by our desiderata: which options can be learned/computed efficiently that enable agents to effectively solve tasks of relevance efficiently? The work we here survey provides partial answers to this question.

Prior work has generated many heuristic methods for discovering good options [62, 97, 100, 127, 121, 122, 81, 93, 46]. These algorithms seek to capture varying intuitions about what makes behavioral abstraction useful. Jong et al. [69] sought to investigate the utility of options empirically and pointed out that introducing options might worsen learning performance. They argued that options can potentially improve the learning performance by encouraging exploitation or exploration. However, at the same time, they show that in some cases, the wrong options can in fact hurt performance. Stolle and Precup [127] propose to set states with high visitation count as subgoal states, resulting in identifying bottleneck states in the four-room domain. Şimşek and Barto [122] generalized the concept of the bottleneck to (shortest-path) betweenness of the graph to capture how pivotal the state is. Menache et al. [100] used a learned model of the environment to run a Max-Flow/Min-Cut algorithm to the state-space graph to identify bottleneck states. These methods generate options to leverage the idea that subgoals are states visited most frequently. On the other hand, Şimşek and Barto [121] proposed to generate options by generating options to relatively novel states, encouraging exploration. Eysenbach et al. [46] instead proposed learning a policy for each option so that the diversity of the trajectories by the set of options are maximized. These methods generate options to explore infrequently visited states. That being said, the problem of discovering efficient behavioral abstraction in reinforcement learning is still open. For planning, several works have shown empirically that adding a particular set of options or macro-operators can speed up planning algorithms [49, 132, 120, 80]. In terms of theoretical analysis, Mann et al. [96] analyzed the convergence rate of approximate value iteration with and without options and showed that options lead to faster convergence if their durations are longer and a value function is initialized pessimistically. As in reinforcement learning, how to find efficient temporal abstractions for planning automatically remains an open question.

Together, a state and action abstraction offer are sufficient for translating any abstracted policy (at the level of taking as input abstract states and outputting options) into the ground MDP. Consider the traditional RL loop augmented by each of these components, pictured in Figure 5.
Figure 6: A state and action abstraction are sufficient for applying an abstract policy in the true MDP.

Here we see that the ground state is projected into the abstract by $\phi$, which is then given to the agent. From the agents perspective, this is equivalent to perceiving the true MDP at the level of its abstracted model. Then, when the agent takes an action, it chooses among its available options $o \in \mathcal{O}$. The action abstraction $\omega$, with the help of the agent’s current memory (or even current state) is responsible for translating the chosen option into a low level action, $a$. Thus, the agent-world interface is retained using these two components.

As a consequence, we find that a state and action abstraction are sufficient for applying an abstraction policy, $\pi_\phi: \mathcal{S}_\phi \rightarrow \mathcal{O}$ into the true MDP. As an example, see Figure 6.

Thus, while the formalism we introduce for characterizing abstraction is relatively simple, it is sufficient for investigating our central research question:

**Central Research Question:** How do intelligent agents come up with the right abstract understanding of the worlds they inhabit?

Or, more specifically:

**Central Research Question (in the context of RL):** How do RL agents construct the right state and action abstractions?

What is meant by the “right” abstraction? We next attend to this question.

### 3.3 Satisfying the Desiderata

We propose the following three desiderata for abstraction as it relates to reinforcement learning. While we consider several learning and planning settings throughout this work, the most general of these settings is the lifelong learning setting, in which the agent must solve a collection of related problems. We offer the following formalism of this setting, taking inspiration from Thrun [137], Guestrin et al. [53], Jong and Stone [67], Brunskill and Li [30], Isele et al. [63], Walsh et al. [143] and Wilson et al. [147]

**Definition 10** (Lifelong RL): In Lifelong RL, the agent is given query access to a fixed but unknown distribution over MDPs, $D$. The following two steps repeat indefinitely:

1. The agent samples $M_i \sim D$.
2. The agent interacts with $M_i$ for $H$ timesteps.
We take this setting to be sufficiently general to capture the breadth of problems studied by RL (and planning) literature, particularly as it relates to the role of abstraction. In part this is due to abstraction’s critical role in consolidating knowledge about prior experiences. With that, we introduce the desiderata:

1. **Efficient Decision Making**: An abstraction ought to enable efficient decision making. That is, planning/learning with an abstraction should be at least as fast or faster than planning with the true environmental problem (and ideally faster).

   **Measurement**: We can measure such quantities mathematically in terms of the computational complexity of planning [31, 109, 92], sample complexity [72, 129], regret [16], or KWIK bounds [91], or empirically in terms of performance improvements on problems of relevance. We will also use the size of the induced abstract model as a proxy, as most sample complexity, computational complexity, and regret bounds depend on the size of the introduced problem (see the above references for further detail). This fact serves as motivation for our work in abstraction-as-compression, discussed in Section 4.3.

2. **Near Optimality**: An abstraction ought to induce a model that enables representation of a good solution to the true environmental problem.

   **Measurement**: We evaluate this condition theoretically based on some variant of value loss bounds, wherein the true optimal policy’s value is compared to the abstracted policy:

   $$\mathbb{E}_{s_0 \sim \rho_{init}} \left[ V^*(s_0) - V^{\pi}(s_0) \right] \leq \tau. \quad (22)$$

   We call this bound the value loss of the abstraction.

   Other measures of interest include recursive optimality and hierarchical optimality introduced by Dietterich [40]. Of course, this quality can be measured empirically, too, by comparing the values of computed optimal policies in problems of relevance.

3. **Efficient-ConSTRUCTION**: Making the abstraction should be easy.

   **Measurement**: The most natural evaluation for (D3) is to produce sample bounds or computational complexity results that showcase how much experience or computation is required to create the abstraction. Of course, experiments here corroborate claims when closed form math is unavailable, as is likely the case in high dimensional environments.

Abstraction, broadly speaking, reduces the dimensionality of an entity. In the context of sequential decision making, abstraction is a representational tool that reduces representational complexity to enable a trade-off between quick and effective decision making. This trade-off is captured by D1 (Efficient Decision Making) and D2 (Near-Optimality), with D3 adding that abstractions should be quick to create, given the computational and statistical

We take these three statements as guiding principles on abstractions for planning and RL, which, collectively, state that:

**Abstractions for sequential decision making should be easy to construct and enable efficient planning and learning while yielding solutions that are useful enough to solve problems of interest.**
3.3.1 Satisfying One or Two Desiderata

We now show that satisfying any one or any two desiderata is trivial. In each of the below remarks, we let $M_\phi = \langle S_\phi, A_\phi, R_\phi, T_\phi, \gamma_\phi, \rho_{\text{init}, \phi} \rangle$ denote the MDP resulting from the abstraction, and $M$ denote the MDP sampled from the environmental distribution $D$. Further, we let $\phi_I$ denote the identity abstraction: $\phi_I(M) = M$.

**Remark:** All three desiderata are trivial to satisfy individually.

**Proof of D1.**

For D1, suppose we replace the ground state and actions space with a single state and single action: $|S_\phi| = |A_\phi| = 1$. Clearly, such a resulting MDP satisfies our first desiderata - we are left with an MDP that is so small that it is trivial to plan or learn in the resulting MDP.

**Proof of D2.**

For D2, we consider $\phi_I$. The optimal policy for $M_{\phi_I}$ is exactly the optimal policy for $M$, consequently satisfying Optimality.

**Proof of D3.**

For D3, again we consider $\phi_I$. The abstraction is the identity function, and so requires no computation or learning.

We now show that any pair of desiderata are trivial to satisfy.

**Remark:** Any two desiderata are trivial to satisfy.

**Proof of D1 & D2.**

For D1 and D2, suppose we solve for the optimal policy $\pi^*$ in $M$ and abstract according to the optimal action(s) in each state. The resulting abstract MDP is as small as can be without losing the optimal policy, per the result of Li et al. [90], and so yields quick planning and learning. Further, the abstract policy $\pi^*_\phi$ is guaranteed to be optimal when applied in the ground: $\max_s V^*(s) - V^{\pi^*_\phi}(s) = 0$.

**Proof of D1 & D3.**

For D1 and D3, we again consider any maximally compressing abstraction that induces an abstract MDP consisting of one state and one action. Planning and learning in this setting is trivial, and the abstraction can be created without any computation or data.

**Proof of D2 & D3.**
For D2 and D3 we again invoke $\phi_I$. Clearly, the identity function is easy to compute and the optimal policy of $M_\phi$ will preserve optimality: $\max_{s \in S} V^*(s) - V^{*\phi}(s) = 0$. 

### 3.3.2 Satisfying All Three

The case of interest is an abstraction that satisfies *all three* desiderata. We are unaware of an existing abstraction that satisfies all three desiderata (or something similar to all three) in a satisfactory way. A major objective of this work is to illustrate some of the interplay between each of the desiderata. We stipulate that satisfying all three represents a major goal for future work, which is partially realized via the theory we introduce in the next section.

In the context of state abstraction, the desiderata ask:

- (D1): How much faster is RL or planning using the state abstraction?
- (D2): How much worse is the abstract model’s optimal policy than the true optimal policy?
- (D3): How easy is it to compute or learn the state abstraction?

Each of these questions has a natural formalism:

- (D1): How does the sample or planning complexity of an algorithm $\mathcal{A}$ compare to the abstraction-augmented algorithm, $\mathcal{A}_\phi$?
- (D2): How bad can the abstract policy be? That is, what is the smallest possible $\tau$ such that:
  \[
  \mathbb{E}_{s_{\text{init}} \sim \rho_{\text{init}}} [V^*(s_{\text{init}})] - V^{*\phi}(s_{\text{init}}) \leq \tau.
  \]  (23)
- (D3): How many experiences (or computations, or both) are needed are needed to compute useful state abstractions?

We find similar formalisms for action abstractions.
4 Completed Work

We now turn the newly established results. The central contributions are as follows:

<table>
<thead>
<tr>
<th>Summary of Contributions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>♦ (Toward D2) Framework for approximate state abstraction that preserve near-optimal behavior in arbitrary MDPs [3].</td>
</tr>
<tr>
<td>♦ (Toward D1, D2, D3) Characterizing and computing state abstractions for lifelong RL that preserve near-optimal behavior [5].</td>
</tr>
<tr>
<td>♦ (Toward D1, D2) Information theoretic view for understanding abstraction as compression [6].</td>
</tr>
<tr>
<td>♦ (Toward D3) Characterizing the computational complexity of finding options that minimize planning time, and an approximation algorithm for doing so [66].</td>
</tr>
</tbody>
</table>

All code is publicly available for reproduction of results and extension. Much of the code is based on the Python RL library, simple rl, also developed as part of this thesis. All proofs are found in the original papers or their supplementary material.

4.1 Approximate State Abstraction

Existing methods of state abstraction depend on exact equivalence of states. Thus, our first objective is to prove that approximate abstraction can still preserve near-optimal behavior. Our main result shows that by relaxing state aggregation criteria from equality to similarity, we can still achieve bounded error in the resulting behavior, while offering three benefits. First, approximate abstractions employ the sort of knowledge that we expect a planning or learning algorithm to compute without fully solving the MDP. In contrast, exact abstractions often require solving for optimal behavior, thereby defeating the purpose of abstraction. Second, because of their relaxed criteria, approximate abstractions can achieve greater degrees of compression than exact abstractions. This difference is particularly important in environments where no two states are identical. Third, because the state aggregation criteria are relaxed to near equality, approximate abstractions are able to tune the aggressiveness of abstraction by adjusting what they consider sufficiently similar states.

We support this thesis by describing four different types of approximate abstraction functions that preserve near-optimal behavior by aggregating states on different criteria: \( \phi_{Q^*,\varepsilon} \), on similar optimal \( Q \)-values, \( \phi_{model,\varepsilon} \), on similarity of rewards and transitions, \( \phi_{bolt,\varepsilon} \), on similarity of a Boltzmann distribution over optimal \( Q \)-values, and \( \phi_{model,\varepsilon} \), on similarity of a multinomial distribution over optimal \( Q \)-values.

We now present four classes of approximate state abstraction, and prove that they each satisfy

\[\text{https://github.com/david-abel/rl_abstraction}\]

\[\text{https://github.com/david-abel/simple_rl}\]
near-optimality conditions. The four predicates we study are:

\[ p_{Q^*}(s_1, s_2) = \max_a |Q(s_1, a) - Q(s_2, a)| \leq \varepsilon, \]

\[ p_{\text{model}, \varepsilon}(s_1, s_2) = \max_a |R(s_1, a) - R(s_2, a)| \leq \varepsilon \land \forall_{a, s', \phi^{-1}(3)} |T(s_1, a, s') - T(s_2, a, s')| \leq \varepsilon, \]

\[ p_{\text{mult}}(s_1, s_2) = \max_a \left| \frac{Q(s_1, a) - Q(s_2, a)}{\sum_b Q(s, b)} \right| \leq \varepsilon, \]

\[ p_{\text{bolt}}(s_1, s_2) = \max_a \left| \frac{e^{Q(s_1, a)} - e^{Q(s_2, a)}}{\sum_b e^{Q(s, b)}} \right| \leq \varepsilon. \]

Our main result states that, for each of the four classes of approximate abstraction, for any MDP, the abstracted model preserves near-optimality behavior. More formally:

**Theorem 4.** There exist at least four types of approximate state aggregation functions, \( \phi_{Q^*}, \phi_{\text{model}, \varepsilon}, \phi_{\text{mult}} \) and \( \phi_{\text{bolt}} \), for which the optimal policy in the resulting abstract MDP, applied to the environmental MDP, has suboptimality bounded by a function of \( \varepsilon \):

\[ \forall_{s \in S} V^*(s) - V^{\phi}(s) \leq 2\varepsilon \text{RMax} \eta_p, \quad (24) \]

where \( \eta_p \) depends on the predicate associated with abstraction function types:

\[ \eta_{Q^*} = \frac{1}{(1 - \gamma)^2}, \]

\[ \eta_{\text{model}} = \frac{1 + \gamma(|S| - 1)}{(1 - 1 - \gamma)^3}, \]

\[ \eta_{\text{bolt}} = \frac{|A| + \varepsilon k_{\text{bolt}} + k_{\text{bolt}}}{(1 - \gamma)^2}, \]

\[ \eta_{\text{mult}} = \frac{|A| + k_{\text{mult}}}{(1 - \gamma)^2}. \]

For \( \eta_{\text{bolt}} \) and \( \eta_{\text{mult}} \), we also assume that the difference in the normalizing terms of each distribution is bounded by some non-negative constant, \( k_{\text{mult}}, k_{\text{bolt}} \in \mathbb{R} \), of \( \varepsilon \):

\[ \left| \sum_i Q(s_1, a_i) - \sum_j Q(s_2, a_j) \right| \leq k_{\text{mult}} \times \varepsilon, \]

\[ \left| \sum_i e^{Q(s_1, a_i)} - \sum_j e^{Q(s_2, a_j)} \right| \leq k_{\text{bolt}} \times \varepsilon. \]

Naturally, the value bound of Equation 24 is meaningless for \( 2\varepsilon \eta_p \geq \text{RMax} \frac{1}{1 - \gamma} = \frac{1}{1 - \gamma}, \) since this is the maximum possible value in any MDP. In light of this, observe that for \( \varepsilon = 0 \), all of the above bounds are exactly 0. Any value of \( \varepsilon \) interpolated between these two points achieves different degrees of abstraction, with different degrees of bounded loss. The degree of approximation (choice of \( \varepsilon \)) is changing the compression-value trade-off made by the abstraction—we analyze this trade-off further in Section 4.3.

We now introduce each approximate aggregation family and prove the theorem by proving the specific value bound for each function type.
4.1.1 Optimal Q Function: $\phi^{Q^\ast,\varepsilon}$

We consider an approximate version of Li et al. [90]'s $\phi^{Q^\ast}$. In our abstraction, states are aggregated together when their optimal $Q$-values are within $\varepsilon$.

**Definition 11 ($\phi^{Q^\ast,\varepsilon}$):** An approximate $Q$ function abstraction has the following form:

$$\phi^{Q^\ast,\varepsilon}(s_1) = \phi^{Q^\ast,\varepsilon}(s_2) \implies \forall_a |Q(s_1,a) - Q(s_2,a)| \leq \varepsilon. \quad (25)$$

**Lemma 1.** When a $\phi^{Q^\ast,\varepsilon}$ type abstraction is used to create the abstract MDP:

$$\forall s \in S \ V^\pi^*(s) - V^\phi(s) \leq \frac{2\varepsilon R_{\text{MAX}}}{(1-\gamma)^2}. \quad (26)$$

4.1.2 Model Similarity: $\phi_{\text{model},\varepsilon}$

Now, consider an approximate version of Li et al. [90]'s $\phi_{\text{model}}$, where states are aggregated together when their rewards and transitions are within $\varepsilon$.

**Definition 12 ($\phi_{\text{model},\varepsilon}$):** We let $\phi_{\text{model},\varepsilon}$ define a type of abstraction that, for fixed $\varepsilon$, satisfies:

$$\phi_{\text{model},\varepsilon}(s_1) = \phi_{\text{model},\varepsilon}(s_2) \implies \forall_a |R(s_1,a) - R(s_2,a)| \leq \varepsilon \quad AND \quad \forall s_\phi \in S_\phi \sum_{s' \in G(s_\phi)} \left| T_\phi(s_1,a,s') - T_\phi(s_2,a,s') \right| \leq \varepsilon. \quad (27)$$

**Lemma 2.** When $S_\phi$ is created using a $\phi_{\text{model},\varepsilon}$ type:

$$\forall s \in S \ V^\pi^*(s) - V^\phi(s) \leq R_{\text{MAX}} \frac{2\varepsilon + 2\gamma \varepsilon (|S| - 1)}{(1-\gamma)^3}. \quad (28)$$

4.1.3 Boltzmann over Optimal $Q$

Here, we introduce $\phi_{\text{bolt},\varepsilon}$, which aggregates states with similar Boltzmann distributions on $Q$-values. This type of abstractions is appealing as Boltzmann distributions balance exploration and exploitation [132]. We find this type particularly interesting for abstraction purposes as, unlike $\phi^{Q^\ast,\varepsilon}$, it allows for aggregation when $Q$-value ratios are similar but their magnitudes are different.

**Definition 13 ($\phi_{\text{bolt},\varepsilon}$):** We let $\phi_{\text{bolt},\varepsilon}$ define a type of abstractions that, for fixed $\varepsilon$, satisfies:

$$\phi_{\text{bolt},\varepsilon}(s_1) = \phi_{\text{bolt},\varepsilon}(s_2) \implies \forall_a \left| \frac{e^{Q(s_1,a)}}{\sum_b e^{Q(s_1,b)}} - \frac{e^{Q(s_2,a)}}{\sum_b e^{Q(s_2,b)}} \right| \leq \varepsilon. \quad (29)$$
We also assume that the difference in normalizing terms is bounded by some non-negative constant, \(k_{\text{bolt}} \in \mathbb{R}\), of \(\varepsilon\):

\[
\left| \sum_b e^{Q(s_1,b)} - \sum_b e^{Q(s_2,b)} \right| \leq k_{\text{bolt}} \times \varepsilon. \quad (30)
\]

**Lemma 3.** When \(S_A\) is created using a function of the \(\phi_{\text{bolt},\varepsilon}\) type, for some non-negative constant \(k \in \mathbb{R}\):

\[
\forall s \in S \pi^*(s) - V_{\pi^*}(s) \leq \text{RMax} \frac{2\varepsilon \left( \frac{|A|}{1-\gamma} + \varepsilon k_{\text{bolt}} + k_{\text{bolt}} \right)}{(1-\gamma)^2}. \quad (31)
\]

We use the approximation for \(e^x\), with \(\delta\) error:

\[
e^x \approx 1 + x + \delta \approx 1 + x. \quad (32)
\]

We let \(\delta_1\) denote the error in approximating \(e^{Q(s_1,a)}\) and \(\delta_2\) denote the error in approximating \(e^{Q(s_2,a)}\).

### 4.1.4 Multinomial over Optimal Q: \(\phi_{\text{mult},\varepsilon}\)

We consider approximate abstractions derived from a multinomial distribution over \(Q^*\) for similar reasons to the Boltzmann distribution. Additionally, the multinomial distribution is appealing for its simplicity.

**Definition 14 (\(\phi_{\text{mult},\varepsilon}\)):** We let \(\phi_{\text{mult},\varepsilon}\) define a type of abstraction that, for fixed \(\varepsilon\), satisfies

\[
\phi_{\text{mult},\varepsilon}(s_1) = \phi_{\text{mult},\varepsilon}(s_2) \implies \forall a \left| \frac{Q(s_1,a)}{\sum_b Q(s_1,b)} - \frac{Q(s_1,a)}{\sum_b Q(s_1,b)} \right| \leq \varepsilon. \quad (33)
\]

We also assume that the difference in normalizing terms is bounded by some non-negative constant, \(k_{\text{mult}} \in \mathbb{R}\), of \(\varepsilon\):

\[
\left| \sum_i Q(s_1,a_i) - \sum_j Q(s_2,a_j) \right| \leq k_{\text{mult}} \times \varepsilon. \quad (34)
\]

**Lemma 4.** When \(S_A\) is created using a function of the \(\phi_{\text{mult},\varepsilon}\) type, for some non-negative constant \(k_{\text{mult}} \in \mathbb{R}\):

\[
\forall s \in S_M V_{\pi^*}(s) - V_{\pi^*}(s) \leq \text{RMax} \frac{2\varepsilon \left( \frac{|A|}{1-\gamma} + k_{\text{mult}} \right)}{(1-\gamma)^2}. \quad (35)
\]

Notably, weaker predicates turn out not to preserve value loss. For example, consider the predicate of reward similarity:
Figure 7: We study which state abstraction functions ($\phi$) are useful for lifelong RL, wherein the agent must learn to solve related tasks from the same distribution.

**Definition 15 (Reward Similarity Abstraction):** The $\phi_{R,\varepsilon}$ family of abstractions is defined by the following implication:

$$\phi_{R,\varepsilon}(s_1) = \phi_{R,\varepsilon}(s_2) \implies \max_{a \in A} |R(s_1, a) - R(s_2, a)| \leq \varepsilon.$$  

(36)

**Lemma 5.** Under the reward similarity abstraction, the value loss is upper bounded by:

$$\varepsilon \gamma V_{\text{Max}}.$$  

(37)

All of the results we have introduced thus far are geared toward a single problem. Worse still, actually constructing any of these abstractions in many cases requires approximate knowledge of $Q^*$ – a quantity which, if known, would let us solve the MDP of interest. So, our next set of results focuses on how to estimate abstractions for a new MDP, given knowledge about a set of related MDPs.

4.2 State Abstractions for Lifelong RL

As stated, our next goal is to extend the previously introduced results into Lifelong RL, in which an agent is presented with a continuous stream of MDPs over the course of its lifetime, as in Brunskill and Li [30], Isele et al. [63], Walsh et al. [143] and Wilson et al. [147]. The studied setting is pictured in Figure 7.

We offer the following definition of this setting:
Definition 16 (Lifelong RL): In Lifelong RL, the agent receives $S, A, s_0 \in S$, horizon $H$, discount factor $\gamma$, and a fixed but unknown distribution over reward-transition function pairs, $D$. The agent samples $(R_i, T_i) \sim D$, and interacts with the MDP $(S, A, R_i, T_i, \gamma)$ for $H$ timesteps, starting in state $s_0$. After $H$ timesteps, the agent resamples from $D$ and repeats.

Lifelong RL presently a particularly difficult set of challenges as it forces agents not only to generalize within an MDP, but also across the distribution of MDPs. The setting is practically relevant to any application where an agent must learn to solve a series of related tasks, as is needed when subtle aspects of the world’s causal dynamics or task change over time. Naturally, lifelong RL is closely related to the objectives and models of transfer learning [136, 137] and multitask RL [135, 29]. For more motivation and background on lifelong RL, see Section 1 of Brunskill and Li [28].

We next introduce new theory for computing and using state abstractions in lifelong RL. Our main contribution is the introduction of two new complementary families of state abstraction that possess desirable properties:

1. Transitive state abstractions: A state-abstraction type associated with a transitive predicate defined on pairs of states.

2. PAC state abstractions: A state abstraction that achieves correct clustering with high probability with respect to a distribution over learning problems.

Together, the joint family of transitive PAC state abstractions are efficient to compute, can be estimated from a finite number of sampled and solved tasks, and preserve near-optimal behavior in lifelong RL. They are thus the closest variant of state abstractions we know of that can satisfy the introduced desiderata. We close with a negative result, however: PAC-MDP algorithms [129] such as R-Max [27] are not guaranteed to interact effectively with an abstracted MDP, suggesting that additional work is needed to combine this idea with efficient learning. Finally, we conduct several simple experiments with standard algorithms to empirically corroborate the effects of state abstraction on lifelong learning.

4.2.1 Transitive Abstractions

We first introduce transitive state abstractions, a restricted class of approximate state abstractions that can be computed efficiently. Concretely, this transitivity guarantees that the predicate $p$ associated with the type satisfies the implication $[p(s_1, s_2) \land p(s_2, s_3)] \implies p(s_1, s_3)$. Many existing state-abstraction types are transitive. However, all known approximate abstraction types, such as those introduced in our previous work, are not. To this end, we introduce a transitive modification of the approximate state-abstraction types. Though the technique does extend to many approximate state abstractions, we focus our efforts here on $Q$-value similarity:

Definition 17 ($\phi_{Q^*}$): For a given $d \in [0, V_{MAX}]$, the $\phi_{Q^*_d}$ denotes a state-abstraction type with predicate:

$$p^*_d(s_1, s_2) \equiv \forall a: \left\lfloor Q^*_M(s_1, a) - d \right\rfloor = \left\lfloor Q^*_M(s_2, a) - d \right\rfloor$$

29
Intuitively, the abstraction discretizes the interval from $[0, \text{VMax}]$ by buckets of size $d$. Then, a pair of states satisfy the predicate if the $Q$-values for all actions fall in the same discrete buckets. Note that this predicate is transitive by the transitivity of being-in-the-same-bucket. As we will show in the next section, the above type preserves near optimal behavior as a function of $d$, and MDP-specific parameters like $\gamma$ and RMax. We can use this same track on all of the approximate state abstractions we previously introduced. For example, reward similarity, or any of the distributional variants of $Q^*$ can also be bucketed while enjoying the same near-optimality guarantees of their non-transitive counterpart.

We first show that transitive state abstractions can be computed efficiently.

**Theorem 5 (Efficient Abstractions).** Consider any transitive predicate on state pairs, $p$, that takes computational complexity $c_p$ to evaluate for a given state pair. The state abstraction type $\phi_p$ that induces the smallest abstract state space can be computed in $O(\sqrt[2]{S} \cdot c_p)$.

The intuition here is that we can shave off many computations by leaning heavily on transitivity. Any one query we make of a state pair predicate can yield information about all connected state pairs. Critically, the complexity of $c_p$ dictates the overall complexity of computing $\phi_p$.

From Table 1, note that known approximate state-abstraction types are not transitive. Hence, our next result shows that there exists an approximate state-abstraction type—with a transitive predicate—with bounded value loss:

**Theorem 6.** The $\phi_{Q_d^*}$ abstraction type is a subclass of $\phi_{Q^*_\varepsilon}$, with $d = \varepsilon$, and therefore, for a single MDP:

$$V^*(s_0) - V^{\pi_{\phi_{Q_d^*}}}(s_0) \leq \frac{2d \text{RMax}}{(1 - \gamma)^2}. \quad (38)$$

Thus, the $\phi_{Q_d^*}$ class represents a reasonable candidate for state abstractions as it can be computed efficiently and possess a value loss that scales according to a free parameter, $d$. When $d = 0$, the value loss is zero, and the abstraction collapses to the typical $\phi_{Q^*}$ irrelevance abstraction from Li et al. [90]. We note that predicates defining other existing abstraction types, such as $\phi_{a^*}$ [90], also have natural translations to transitive predicates using the same discretization technique. While most of our main theoretical results are agnostic to choice of predicate, we concentrate on $Q$ based abstractions due to their simplicity and utility. Notably, we never require exact knowledge of $Q^*$: we always approximate it based on knowledge of prior tasks. Our results shed light on when it is possible to employ approximate knowledge of this kind for use in decision making.

Recall, however, that the primary goal of state abstraction is to reduce the size of the agent’s representation over problems of interest. A natural question arises: if one were to solve the full NP-Hard problem of computing the maximally compressing state abstraction of a particular class, how much more compression can be achieved over the transitive approximation? Intuitively: Is the transitive abstraction going to compress the state space? The following result addresses this question.

**Theorem 7 (Abstract State Space Size).** For a given $d$, the function belonging to the transitive abstraction type $\phi_{Q_d^*}$ that induces the smallest possible abstract state space size is at most $2^{2|A|}$ times larger than that of the maximally compressing instance of type $\phi_{Q^*_\varepsilon}$, for $d = \varepsilon$. Thus, letting $S_d$ denote the abstract state space associated with the maximally compressing $\phi_{Q_d^*}$, and letting $S_{\varepsilon}$ denote the abstract state space associated with the maximally compressing $\phi_{Q^*_\varepsilon}$:

$$|S_d| \cdot 2^{2|A|} \geq |S_{\varepsilon}|. \quad (39)$$

30
The above result shows that the non-transitive, maximally compressing state space size can in fact be quite smaller than the transitive approximation (by a factor of $2^{\divides}A\divides\divides$).

### 4.2.2 PAC Abstractions

Our main goal is to bring the tools of state abstractions to bear on lifelong RL. Other work has undertaken this endeavor, such as Guestrin et al. [53], Walsh et al. [143] and Jong and Stone [67], and an earlier version of our work [4]. We here build on the principles established by this work with a new family of abstractions that are guaranteed to hold with high probability over the task distribution $D$, inspired by PAC learning [141]:

**Definition 18 (PAC State Abstraction):** A PAC state abstraction $\phi^\delta_p$ is a state-abstraction function belonging to type $\phi_p$ such that, for a given $\delta \in (0, 1]$, and a given distribution over MDPs $D$, the abstraction groups together nearly all state pairs for which the predicate $p$ holds with high probability over the distribution.

More formally, for an arbitrary state pair $(s_1, s_2)$, let $\rho^p_x$ denote the predicate that is true if and only if $p$ is true over the distribution with probability $1 - x$:

$$\rho^p_x(s_1, s_2) \triangleq \Pr_{M \sim D} \{p_M(s_1, s_2) = 1\} \geq 1 - x.$$  \hspace{1cm} (40)

We say $\phi^\delta_p$ is a PAC state abstraction if there exists a small constant $\varepsilon \in (-\delta, \delta)$ such that, for all state pairs $s_1, s_2$:

$$\Pr\{\rho^p_{\delta + \varepsilon}(s_1, s_2) \equiv \phi^\delta_p(s_1) = \phi^\delta_p(s_2)\} \geq 1 - \delta.$$  \hspace{1cm} (42)

Intuitively, we can’t expect to come up with a single abstraction that captures the changing landscape of state-relations under a given predicate $p$ and task distribution. Instead, we focus on predicates that hold with high probability over the distribution $\rho^p_\delta$, and seek an abstraction that probably approximately captures all of those state-pair relations. Consequently we group most states that can be grouped across most MDPs in the distribution.

Our second collection of results focus on pulling state abstractions out of the single MDP setting and into the lifelong setting via PAC abstractions. We first show that they achieve a probabilistic value loss:

**Corollary 1 (PAC Value Loss).** Consider any state-abstraction type $\phi_p$ with value loss $\tau_p$, that is, in the traditional single task setting:

$$\forall_{s \in S} : V^*_s(s) - V^{\pi_{\phi_p}}(s) \leq \tau_p.$$  \hspace{1cm} (41)

Then, the PAC abstraction $\phi^\delta_p$, in the lifelong setting, has expected value loss:

$$\forall_{s \in S} : \mathbb{E}_{M \sim D} \left[V^*_M(s) - V^{\pi_{\phi^\delta_p}}_M(s)\right] \leq \varepsilon(1 - 3\delta)\tau_p + 3\delta V_{\text{MAX}}.$$  \hspace{1cm} (42)
The value loss is actually quite high, as we can lose $3\delta V_{\text{Max}}$. Accordingly, we must be careful in selection of $\delta$. This bound is not tight, however, so in general the value loss may be lower.

Next, we show how to compute PAC abstractions from a finite number of sampled tasks.

**Theorem 8** (PAC Abstraction Sample Bound). Let $\mathcal{A}_p$ be an algorithm that given an MDP $M = \langle S, A, R, T, \gamma \rangle$ as input can determine if $p(s_1, s_2)$ is true for any pair of states, for any state abstraction type.

Then, for a given $\delta \in (0, 1]$ and $\varepsilon \in (-\delta, \delta)$, we can compute a PAC abstraction $\hat{\phi}_p^\delta$ after $m \geq \ln(\frac{2}{\varepsilon^2})$ sampled MDPs from $D$.

Note that this result assumes oracle access to the true predicate, $p(s_1, s_2)$, during the computation of $\hat{\phi}_p^\delta$. The analogous case in which $p$ can only be estimated via an agent’s interaction with its environment is a natural next step for future work. We have explored this result, but it requires care and attention to detail which we defer for another time.

Given the ability to compute PAC abstractions from a finite number of samples, we now shed some initial light on the interplay between state abstractions and PAC-MDP algorithms for efficient RL.

**Theorem 9.** Consider an MDP $M$ and an instance of the classical model-based algorithm, R-Max [27], that breaks ties using round-robin selection over actions. This algorithm is PAC-MDP in the raw state space. Next, pair a domain with any state-abstraction function $\phi$. If R-Max interacts with $M$ by projecting any received state $s$ through $\phi$, then R-Max is no longer guaranteed to be PAC-MDP in $M$. In fact, the number of mistakes made by R-Max can be arbitrarily large.

The above result is a surprising negative result— it suggests that there is more to the abstraction story than simply projecting states into the abstract. Specifically, it is indicative of future work that sheds light how we can form abstractions that preserve the right kinds of guarantees.

To communicate this piece more directly, we conduct a simple experiment in the 3-chain problem introduced in the proof of Theorem 9. Here we run R-Max and Delayed Q-Learning with and without $\phi$. Each agent is given 250 steps to interact with the MDP. The results are shown in Figure 8. R-Max, paired with abstraction $\phi$, fails to learn a anywhere close to a near-optimal policy. In fact, we can control a parameter in the MDP such that R-Max performs arbitrarily bad. It remains an open question as to whether $\phi$ preserves the PAC-MDP property for Delayed $Q$ [128].

To explicate this point further, we next show that projecting an MDP to the abstract state space via $\phi$ and learning with $M_\phi$ is non-identical to learning with $M$ and projecting states through $\phi$:

**Corollary 2.** For any RL algorithm $\mathcal{A}$ whose policy updates during learning and an arbitrary state abstraction $\phi$.

Let $\mathcal{A}_\phi$ denote the algorithm yielded by projecting all incoming states to $\phi(s)$ before presenting them to $\mathcal{A}$, and let $M_\phi = \langle S_\phi, A, T_\phi, R_\phi, \gamma \rangle$, denote the abstract MDP induced by $\phi$ on $M$, where:

$$S_\phi = \{ \phi(s) : \forall s \in S \},$$

$$R_\phi(\phi(s), a) = \sum_{g \in \delta^{-1}(\phi(s))} w(g)R(g, a),$$

$$T_\phi(s, a, s') = \sum_{g \in G(s)} \sum_{g' \in G(s')} T_\phi(g, a, g') w(g),$$
with \( w(s) \) is a fixed weighting function and \( G(s) = \phi^{-1}(\phi(s)) \). That is, \( G(s) \) gets all of the true environmental states in the same cluster as \( s \).

The process yielded by \( \mathcal{A}_\phi \) interacting with \( M \) is not identical to \( \mathcal{A} \) interacting with \( M_\phi \). That is, the expected trajectory taken by the agent is not the same in the two situations. Formally:

\[
E_{\mathcal{A}}[s_t | s_0, \pi] \neq E_{\mathcal{A}_\phi}[s_t | s_0, \pi],
\]

where \( s_t \) is the state the agent arrives in after \( t \) time steps.

Again, we find a peculiarity to the abstraction story: abstracting during interaction is distinct from offline abstraction. This result is reminiscent of parts of Theorem 4 and Theorem 5 from Li et al. [90]. In the future, we aim to provide a cohesive framework that preserves both PAC and convergence guarantees, whether the abstractions are used offline or during interaction.

To summarize our theorems about state abstraction in lifelong RL: any state abstraction that belongs to both the transitive class and the PAC class is: (1) efficient to compute, (2) can be estimated from a polynomial number of sampled and solved problems, (3) and preserves near-optimal behavior in the lifelong RL setting. The identification of such a class of desirable state abstractions for lifelong RL is the main contribution of this paper. We further uncover a peculiar shortcoming of state abstractions in the final two results, raising open questions about how to generalize state abstractions to work well with PAC-MDP algorithms.

4.2.3 Experiments

We conduct two sets of simple experiments with the goal of illuminating how state abstractions impact learning and decision making.

- **Learning with and without \( \phi_\delta \):** We investigate the impact of different types of abstractions on Q-Learning [144] and Delayed Q-Learning [128] in a lifelong RL task distribution.

- **Planning with and without \( \phi_\delta \):** Second, we explore the impact of planning via Value Iteration [?] with and without a state abstraction, as suggestive of the potential to accelerate model-based algorithms with good state abstractions.

![Cumulative Reward: 3 Chain](image)

Figure 8: Results averaged over 50 runs on the pathological 3 chain MDP introduced in the proof of Theorem 9.
In each case, we compute various types of $\phi_\delta$ according to the sample bound from Theorem 8, with $\delta = 0.1$, and the PAC parameter $\varepsilon = 0.1$ (the worst case $\varepsilon$). We experiment with ($\phi_{Q^*}$), approximate ($\phi_{Q^*_A}$), and transitive ($\phi_{Q^*_d}$) state abstractions from the $Q$ similarity classes across each of the above algorithms. We experiment with probably approximate $Q$ based abstractions because their value loss bound is known, tight, and a small function of the approximation parameter, and (2) They have known transitive variants and are thus simple to compute, as we show in Theorem 5. Further, if a $Q^*$ based abstraction presents no opportunity to abstract (the reward or transition function change too dramatically across tasks), then Theorem 8 tells us that we will abstain from abstracting.

Each learning experiment proceeds as follows: for each agent, at timestep zero, sample a reward function from the distribution. Then, let the agent interact with the resulting MDP for 100 episodes. When the last episode finishes, reset the agent to the start state $s_0$, and repeat. All learning curves are averaged over samples from the distribution. Thus, improvements to learning from each $\phi$ are improvements averaged over the task distribution. In all learning plots we report 95% confidence intervals over the sampling process (both samples from the distribution and runs of each agent in the resulting MDP).

![Figure 9: Cumulative reward averaged over 100 task samples from the Colored Four Rooms task distribution. Learning algorithms were given 100 episodes of 250 steps to learn, paired with a variety of different state-abstraction types.](image)

**Color Room:** We first conduct experiments testing Q-Learning and Delayed Q Learning on an 11×11 Four Room variant, adapted from Sutton et al. [134]. In the task distribution, goal states can appear in exactly one of the furthest corner of each of the three non-starting rooms (that is, there are three possible goal locations) uniformly at random. Transitions into a goal state yield +1 reward with all other transitions providing +0. Goal states are set to terminal. To explore the impact of abstraction, we augment the problem representation by introducing an irrelevant dimension: color. Specifically, each cell in the grid can have a color red, blue, green, or yellow. All cells are initially red. The agent is given another action, paint, that paints the entire set of cells to one of the four colors uniformly at random. No other action can change the color of a cell. The color has no impact on either reward or transitions, and so is fundamentally irrelevant in decision making. We are thus testing the hypothesis as to how effectively the sample based PAC abstractions can pick up on the irrelevant characteristics and still support efficient but high performance learning. Given the inherent structure of the Four Rooms domain we also experiment with an intuitively useful hand-coded state abstraction ($\phi_h$) that assigns an abstract state to each
room (for a total of four abstract states). The agents all start in the bottom left cell.

Figure 9 shows results for algorithms run on the Colored Four Rooms task distribution. First, notice that \( \phi_h \), the hand coded abstraction, is disastrous for both learning algorithms. Despite employing a seemingly reasonable decomposition of the state space, the agent fails to come close to the performance of the baseline agent. We draw a parallel between these results and those presented in our previous work [4], where we identify the existence of cases where the benefits of generalization that come with state abstractions are accompanied by more challenging exploration and credit-assignment problems. Conversely, for Q-learning we find that all three PAC abstractions achieve statistically significant improvement in cumulative reward, averaged across 100 task samples. Notably, the slope of the learning curves is roughly equivalent. We conclude that all of the algorithms are learning policies of roughly similar value (except Q-Learning), but the abstraction learners find these policies more quickly. In the case of Q-Learning, the PAC abstractions find even further improvement over the baseline learner, both in terms of learning speed and the value of the policy used near the end of learning.

Planning To give further evidence of the potential benefits offered by state abstraction we conduct two simple planning experiments. Motivated by the work on planning with state abstractions [8, 7, 65], we here show the impact of giving Value Iteration a state abstraction in two simple problems. The first is the 10 \times 30 Upworld grid problem from Abel et al. [3]. The second is the Trench problem from Abel et al. [2], in which the agent must seek out a block, pick it up, carry it to a trench, place the block in the trench, and walk across the block to the goal. In each case, we vary the size of the state space by changing the width of the grid. In Upworld, this range is from 3 to 20, while in Trench the range is from 3 to 6.

Figure 10 shows the planning time taken compared to the size of the problem for Value Iteration. The results are expected: in both Upworld and the Trench problem, there are opportunities to abstract aggressively, thereby significantly lowering the computational burden of planning. These results suggest that future model-based RL algorithms employing the appropriate abstractions can plan efficiently.

Note that in all of the approximate abstractions we have here introduced, the parameter \( \varepsilon \) dictates how much to prioritize compression of the MDP vs. preservation of value. However, our
results thus far have not explicitly addressed this fact – a natural question still stands: how much can an MDP be compressed by a state abstraction such that an agent can still learn to solve it? Our next section seeks a principled answer to this question.

\section*{4.3 State Abstraction as Compression}

Information Theory offers foundational results about the limits of compression \cite{119}. The core of the theory clarifies how to communicate in the presence of noise, culminating in seminal results about the nature of communication and compression that helped establish the science and engineering practices of computation. Of particular relevance to our agenda is Rate-Distortion theory, which studies the trade-off between a code’s ability to compress (rate) and represent the original signal (distortion) \cite{119,23}. Cognitive neuroscience has suggested that perception is tied to efficient compression \cite{15}, termed the “efficient coding hypothesis” by Barlow \cite{19}.

The goal of this work is to understand the role compression plays in state abstraction for sequential decision making. We draw a parallel between \textit{state abstraction} as used in reinforcement learning and \textit{compression} as understood in information theory. We build heavily on the seminal work of Shannon \cite{119}, Blahut \cite{25}, Arimoto \cite{12} and Tishby et al. \cite{139}, and draw inspiration from related work on understanding the relationship between abstraction and compression \cite{26,126}. While the perspective we introduce is intended to be general, we focus our study in two ways. First, we investigate only \textit{state abstraction}, deferring discussion of temporal \cite{134}, action \cite{54}, and hierarchical abstraction \cite{36,40} to future work. Second, we devote attention to the learning problem when a \textit{demonstrator} is available, as in Apprenticeship Learning \cite{14,1,11}, which simplifies several important aspects of our model.

Concretely, we introduce a new objective function that explicitly trades off state-compression and performance. Our main result proves this objective is upper bounded by a variant of the Information Bottleneck objective adapted to sequential decision making. We introduce \textsc{DiBS}, an algorithm that outputs a \textit{lossy} state abstraction optimizing the trade-off between compressing the state space and preserving the capacity for performance in that state space. We show this algorithm converges to a local minima of the Information Bottleneck objective, thereby minimizing an upper bound on our true objective. We conduct experiments to showcase the trade-off between compression and performance captured by the algorithm in a traditional grid world and present an extension to high-dimensional observations via experimentation with the Atari game Breakout. We close by suggesting concrete paths toward realizing a general abstraction-as-compression learning paradigm for sequential decision making.

\subsection*{4.3.1 Information Theory Background}

\textbf{Information Theory} studies communication in the presence of noise \cite{119}. In Shannon’s words: “The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.” Information Theory typically investigates coder-decoder pairs and their capacity to faithfully communicate messages with zero or low error, even in the presence of noise. The seminal results all center around the definition of \textit{entropy}:
Figure 11: The usual Rate-Distortion setting (left) and the lower bound on the Rate-Distortion trade-off (right)

Figure 12: The basic quantities of information theory and their relations.

**Definition 19** (Entropy): The **Entropy** of a discrete random variable $X$, with alphabet $\mathcal{X}$, is given by:

\[
H(X) := - \sum_{x \in \mathcal{X}} \Pr(X = x) \log_2 \Pr(X = x).
\]  

(44)

An additional quantity of relevance is the Mutual Information between two discrete random variables:

**Definition 20** (Mutual Information): The **mutual information** of two discrete random variables $X$ and $Y$ is given by:

\[
I(X;Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}
\]  

(45)

Together with the joint and conditional entropy, we find a surprisingly elegant set of relations between the basic quantities of the theory, pictured in the venn diagram in Figure 12.
**Rate-Distortion (RD) Theory** studies the trade-off between a coder-decoder pair’s ability to compress a signal and the pair’s ability to faithfully reproduce the original signal. The typical RD setting is pictured in Figure 11a: an information source generates \( x \in \mathcal{X} \), which is coded via \( p(z \mid x) \) for \( z \in \mathcal{Z} \), and decoded via a deterministic function \( f : \mathcal{Z} \rightarrow \tilde{\mathcal{X}} \). Distortion is defined with respect to some chosen distortion metric, \( d : \mathcal{X} \times \tilde{\mathcal{X}} \rightarrow \mathbb{R}_{\geq 0} \), where typically \( \mathcal{X} = \tilde{\mathcal{X}} \). The information Rate \( R \) denotes the number of bits in each code word. So, with a coding alphabet \( \mathcal{Z} = \{0, 1\}^n \), the rate is \( n \). Shannon and Kőnigov (see Berger [23] for more background) offer a lower bound on the trade-off between Rate and Distortion: given a level of distortion, \( D \), the following function defines the smallest rate that achieves distortion of at most \( D \):

\[
R(D) = \min_{p(\tilde{x} \mid x) : d(x, \tilde{x}) \leq D} I(X; \tilde{X}),
\]

(46)

where \( I(X; \tilde{X}) \) is the mutual information between random variables \( X \) and \( \tilde{X} \). Intuitively, as we add bits to our code, we can more faithfully reconstruct our source message. The curve displayed in Figure 11b shows an example lower bound of the trade-off between Rate and Distortion expressed by Equation 46.

![Figure 13: Our framework for trading off compression and value with state abstraction.](image)

For a given information source, it is natural to consider how one might compute the coder-decoder pair that achieves one of the minimal points defined by the Rate-Distortion function. Finding this point presents the following optimization problem:

\[
\min_{p(\tilde{x} \mid x)} \underbrace{I(X; \tilde{X})}_{\text{Rate}} + \beta \underbrace{\mathbb{E}[d(x, \tilde{x})]}_{\text{Distortion}}.
\]

(47)

Blahut-Arimoto (BA) is a simple iterative algorithm that converges to the global optimum of this optimization problem [12, 25]. BA alternates between the following two steps, for a given \( \beta \in \mathbb{R}_{\geq 0} \):

\[
p_{t+1}(x) = \sum_{x} p_t(x) p_t(\tilde{x} \mid x) \quad \text{and} \quad p_{t+1}(\tilde{x} \mid x) = \frac{p_t(\tilde{x}) \exp \{-\beta d(x, \tilde{x})\}}{\sum_{x'} p_t(x') \exp \{-\beta d(x', \tilde{x})\}}.
\]

When \( \beta \) is large, the algorithm places larger preference on lowering the distortion in Equation 47. When \( \beta \) is closer to 0, BA prioritizes compression (at the cost of more distortion). BA is known to converge to the global optimum with convergence rate:

\[
O\left(\frac{\|\mathcal{X}||\tilde{\mathcal{X}}|\sqrt{\log(|\tilde{\mathcal{X}}|)} / \varepsilon}{\varepsilon}\right),
\]

(48)

for \( \varepsilon \) error tolerance [12], That is, letting \( R \) be the rate and \( \text{hat} R^n \) be the estimated rate given by the \( n \)-th iteration of BA, the convergence rate is \( |R - \text{hat} R^n| \leq O\left(\frac{\log(N)}{n}\right) \), where \( n \) is the number of
iterations [12]. The computational complexity of finding the exact solution for discrete memoryless channel is unknown. For continuous memoryless channel the problem is an infinite dimensional convex optimization which is known to be NP-Hard [131].

The Information Bottleneck (IB) Method extends RD theory to prediction, by recasting relevant information in terms of a code’s ability to predict another variable, $Y$. By contrast, Traditional RD theory defines “relevant” information by choice of a distortion function—codes are said to capture relevant information if they achieve low distortion. In IB, we define relevant information according to how well a random variable $Y$ can be predicted from each $\tilde{x} \in \tilde{X}$. For the IB to make sense, we must suppose that $I(X; Y) > 0$, and that the coder–decoder scheme has access to the joint probability mass function (pmf) $p(X, Y)$. IB then recasts the RD lower bound in Equation 46 in terms of prediction of $Y$ given $X$. The optimal assignment to the distribution $p(\tilde{x} | x)$ is then given by minimizing:

$$
\mathcal{L} [p(\tilde{x} | x)] = I(\tilde{X}; X) - \beta I(\tilde{X}; Y),
$$

(49)

where $\beta \in \mathbb{R}_{\geq 0}$ is a Lagrange multiplier attached to the meaningful information. Like BA, choice of $\beta$ determines the relative preference between compression (rate) and predicting $Y$ (distortion); when $\beta = 0$, the coder can ignore $Y$ entirely, and so is free to compress arbitrarily. Conversely, as $\beta \to \infty$, the coder must prioritize prediction of $Y$, requiring more bits in the coding alphabet.

Tishby et al. [139] offer a convergent algorithm for solving the above optimization problem.

**Theorem 10.** [139] Equation 49 yields the following optimization problem:

$$
\min_{p(\tilde{x} | x)} \mathcal{L}_{IB} [p(\tilde{x} | x); p(\tilde{x}); p(y | \tilde{x})] =
$$

$$
\min_{p(\tilde{x} | x)} \left( I(X; \tilde{X}) + \beta \mathbb{E}_{p(x, \tilde{x})} [D_{KL}(p(y | x) \| p(y | \tilde{x})] \right).
$$

The following three steps, repeated, converge to a minima of the above optimization problem, with $Z(s, \beta)$ a normalizing term:

$$
\begin{cases}
    p_t(\tilde{x} | x) \leftarrow \frac{Z(s, \beta)}{Z(s, \beta)} \exp\{-\beta D_{KL}(p(y | x) \| p_t(y | \tilde{x}))\}, \\
    p_{t+1}(\tilde{x}) \leftarrow \sum_x p(x) p_t(\tilde{x} | x), \\
    p_{t+1}(y | \tilde{x}) \leftarrow \sum_y p(y | x) p_t(\tilde{x} | x).
\end{cases}
$$

Critically, the algorithm only presents a locally optimal solution to the above optimization problem. To the best of our knowledge, there is no known efficient algorithm for computing the global optimum. Mumey and Gedeon [104] show that a closely related problem to finding the global optimum to the above is in fact NP-Hard, suggesting that approximation is likely our best option. Like BA, choice of $\beta$ determines the relative preference between compression (rate) and predicting $Y$ (distortion).

The Deterministic IB (DIB) extends the IB by focusing on deterministic coding functions where $p(\tilde{x} | x) \equiv f : \mathcal{X} \to \tilde{\mathcal{X}}$ [130]. Given the equality $I(X; Y) = H(X) - H(X | Y)$, note that when
the coder is a deterministic function $f$, we can replace the mutual information term in the objective by the entropy of the latent space:

$$\min_{f(x)} \mathcal{L}_{\text{DIB}}[f(x); p(\tilde{x}); p(y \mid x)] = \min_{f(x)} \left( H(\tilde{X}) + \beta \mathbb{E}_{p(x, \tilde{x})} \left[ D_{\text{KL}}(p(y \mid x) \| p(y \mid \tilde{x})) \right] \right). \quad (51)$$

Given that state abstractions tend to be deterministic, we will primarily be focused on this extension.

### 4.3.2 State Abstraction as Compression

Our model of compression is pictured in Figure 13. Our goal is to answer the following question: *How many abstract states are needed for an agent to faithfully make similar decisions to a demonstrator?* We cast the Rate-Distortion trade-off as a trade-off between (1) the size of the abstract state space $|\mathcal{S}_\phi|$, and (2) the value of the best policy representable using $\mathcal{S}_\phi$ compared to $\pi_d$. More formally, we introduce and study the following objective:

**Definition 21** (Compression-Value Abstraction or CVA Objective): The objective function, $J$, for a given Lagrange multiplier $\beta \in [0, \infty)$, is defined as:

$$\min_{\phi} J[\phi] := \min_{\phi} \left( |\mathcal{S}_\phi| + \beta \mathbb{E}_{\rho_d(s)} \left[ V^{\pi_d}(s) - V^{\pi_\phi}(s) \right] \right),$$

where $V^{\pi_\phi}(s)$ is shorthand for $V^{\pi_\phi}(\phi(s))$.

Our goal is to define an algorithm that solves the above objective in finite time, with at least local convergence guarantees. However, to the best of our knowledge, there is no known efficient method to solve the above objective. Instead, to achieve our goal, we next introduce an IB-like objective that serves as an upper bound on $J$.

We use the following definition, denoting the size of the non-negligibly used portion of an alphabet under a given pmf:

**Definition 22** (pmf-Used Alphabet Size): The pmf-used alphabet size of $X$ is the number of elements whose probability under $p(x)$ is greater than some negligibility threshold $\delta_{\text{min}} \in (0, 1)$:

$$|X|_{p(x)}^{\delta_{\text{min}}} := \min \{|\{x \in \mathcal{X} : p(x) > \delta_{\text{min}}\}|, |\mathcal{X}|\}.$$  

This notion of alphabet size generalizes the usual method of measuring the size of a state space. When we think about the CVA objective, the state space size will typically be thought of in relation to this notion of state space size, under a given state distribution.

One might wonder why such a question cannot be answered by assigning one abstract state to each action, as is captured by the $\pi^*$-irrelevance abstractions studied by Li et al. [90]. First, if the demonstrator policy is stochastic, no such abstraction exists. Second, we are ultimately interested in state abstractions that facilitate effective *learning*; if the abstraction were given to an arbitrary
RL algorithm, we would like learning to be made easier. Highly aggressive abstraction types like $\pi^*$ destroy guarantees and make aspects of learning harder [90, 4]. Moreover, by better understanding the limits of compression in MDPs, we may open the door to a general abstraction-as-compression paradigm, without a demonstrator policy entirely.

4.3.3 IB as an Upper Bound on $J$

In our setup, the MDP paired with the fixed control policy $\pi_d$ define an information-generating source. That is, the source distribution is defined, for each $s \in S$, as:

$$\rho_d(s) := \sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s \mid s_0, \pi_d).$$

(52)

At each time step, a state is sampled from $\rho_d$ and given to a learning agent through a probabilistic state-abstraction function, $\phi : S \rightarrow \Pr(S_\phi)$, which projects each state to each abstract state $s_\phi$ with some probability. Our core simplifying assumption is: There exists a demonstrator policy $\pi_d$ that controls the MDP. The agent’s goal is to perform as well as the demonstrator using as small of a state space as possible, as reflected by $J$.

To construct the IB/DIB analogue objective, we let $I(S; S_\phi)$ denote the rate and let $D_{KL}(\pi_d(a \mid s) \parallel \pi_\phi(a \mid s_\phi))$ denote the distortion. We further suppose there exists a fixed, deterministic mapping from $S_\phi$ to $\tilde{S}$, with $\tilde{S} = S$. Thus, the distribution $p(\tilde{x} \mid x)$ is simply $p(s_\phi \mid s)$, which we abbreviate as $\phi$. Consequently, we find the following alignments between our objects of study (abstractions, policies) and those studied by IB:

$$p(\tilde{x} \mid x) \sim \phi, \quad p(\tilde{x}) \sim p_\phi, \quad p(y \mid \tilde{x}) \sim \pi_\phi,$$

(53)

where $p_\phi$ is the steady state distribution over abstract states induced by $\pi_\phi$ and $\phi$. Using these alignments, per Theorem 1, we construct an objective function $\hat{J}$ based on the IB:

$$\min_{\phi} \hat{J} [\phi; p_\phi; \pi_\phi] :=$$

$$\min_{\phi} \left( I(S; S_\phi) + \beta \mathbb{E}_{\rho_d(s)} \left[ D_{KL}(\pi_d(a \mid s) \parallel \pi_\phi(a \mid s)) \right] \right).$$

(54)

Note that $\pi_\phi(a \mid s)$ is shorthand for $\pi_\phi(a \mid \phi(s))$. If we choose to use the DIB instead, the above mutual information term is replaced by $H(S_\phi)$.

4.3.4 Relating KL and $V, H$ and $|S_\phi|$

We now present the main result of the paper, which relates $\hat{J}$ to $J$. To illustrate this relationship, we present two key lemmas that relate each of KL with $V$, and $H(S_\phi)$ with $|S_\phi|$. These will help in proving the following theorem:

**Theorem 11.** A function $f$ of the IB objective $\hat{J}$ is an upper bound for the CVA Objective, $J$, where state space size is treated as $|S_\phi|_{\rho_d(s)}$:

$$\forall \phi : J[\phi] \leq f(\hat{J}[\phi]).$$

(55)
To prove the theorem, we require two lemmas. The first relates the entropy of a pmf to the maximum size of the alphabet used by that pmf:

**Lemma 6.** Consider a discrete random variable \( X \), with alphabet \( \mathcal{X} \) and some pmf \( p(x) \). For a given threshold \( \delta_{\text{min}} \in (0, 1] \), the pmf-used alphabet size of the alphabet is bounded:

\[
|\mathcal{X}|_{p(x) \delta_{\text{min}}} \leq \frac{H(X)}{\delta_{\text{min}} \log \left( \frac{1}{\delta_{\text{min}}} \right)}.
\]  

(56)

This bound is relatively loose: We know \( H(X) \leq \log_2 |\mathcal{X}| \). Thus, in the worst case, the bound can be up to \( \tilde{O}(1/\delta_{\text{min}}) \) times larger than the true alphabet. Still, this result allows us to relate the entropy of a random variable with its used alphabet size. Further, by definition, the entropy \( H(S_{\phi}) \) already gives us a lower bound on the number of bits needed to represent the used parts of \( S_{\phi} \). So, the entropy as a measure of compression is exploiting the fact that the most probable state can be written as 0, the second most probable state as 10, and so on. Thus, a lower entropy is already indicative of compressing \( |S_{\phi}| \). Further, in experiments, we find this upper bound is loose relative to the size of the abstract state space our algorithm produces.

Next, we introduce a second lemma that relates the expected KL divergence between two policies to the difference in value achieved by the policies, in expectation under some state distribution. To the best of our knowledge, this result is novel:

**Lemma 7.** Consider two stochastic policies, \( \pi_1 \) and \( \pi_2 \) on state space \( S \), and a fixed probability distribution over \( S \), \( p(s) \). If, for some \( k \in \mathbb{R}_{\geq 0} \):

\[
\mathbb{E}_{p(s)} \left[ D_{\text{KL}}(\pi_1(a \mid s) || \pi_2(a \mid s)) \right] \leq k,
\]

then:

\[
\mathbb{E}_{p(s)} \left[ V^{\pi_1}(s) - V^{\pi_2}(s) \right] \leq \sqrt{2k} V_{\text{Max}},
\]

(58)

where \( V_{\text{Max}} \) is a bound such that all value-function components must fall in \( [-V_{\text{Max}}, V_{\text{Max}}] \).

Note that this bound is vacuous for values of \( k \geq \frac{1}{2} \). However, we find in experiments that the gap in KL between the demonstrator and the computed policy ends up being quite low (typically well below 1/2).

The optimization problem presented by \( \hat{J} \) can be solved by the usual IB method. We thus introduce Deterministic Information Bottleneck for State abstractions (DIBS, presented in Algorithm 5), a simple iterative algorithm that adapts the DIB to Apprenticeship Learning with state abstractions. DIBS outputs a state-abstraction–policy pair in finite time that computes a local minimum of \( \hat{J} \), which we know from Theorem 11 is an upper bound on \( J \). The pseudocode presented is for the deterministic variant of the IB, as often state abstraction functions are treated as deterministic aggregation functions [90]. The stochastic variant, which we call SIBS, will also be of interest, as soft state aggregation has been explored as well [124].

**4.3.5 Experiments**

To explore the utility of DIBS for solving the CVA objective, we next describe two experiments that showcase its power for trading off compression and value. In the first case, we study the
Figure 14: The RD curve computed by DIBS (a), the value of the state-abstraction–policy combination \((\phi, \pi_\phi)\) found by DIBS for different values of \(\beta\) (b), and the state abstractions found when \(\beta = 2\) (c) and \(\beta = 100\) (d). In (c) and (d), all grid cells with the same color are grouped together in the same abstract state.

traditional Four Rooms domain introduced by Sutton et al. [134] to explore the usefulness of DIBS for trading off compression and value in a simple MDP. Second, we present a simple extension to SIBS that scales to high-dimensional observation spaces. We evaluate this extension in the Atari game Breakout.

In Four Rooms, we run DIBS and SIBS to convergence and compare the value of \(\pi_\phi, \text{DIBS}\) and \(\pi_\phi, \text{SIBS}\) to the value of the demonstrator policy for \(\beta\) between 0 and 4. Results averaged over 20 runs (with 95% confidence intervals) are presented in Figure 14b, which showcases the value achieved by the abstraction-policy pair for each setting of \(\beta\). Notably, as \(\beta\) increases past one, both \(\pi_\phi, \text{DIBS}\) and \(\pi_\phi, \text{SIBS}\) are able to almost exactly match the value of the demonstrator. Figures 14c and 14d show the state abstractions found by DIBS for \(\beta = 2\) and \(\beta = 100\), respectively. For the abstraction in Figure 14c, there are three abstract states, which is sufficient for nearly representing the demonstrator policy (“move right” in blue, and so on). As we increase \(\beta\), we find the abstraction gives rise to far more states. We also construct a Rate-Distortion curve based on the experiments, presented in Figure 14a, which shows the trade-off made between the size of used \(S_\phi\) and the expected value loss.

**Breakout** We next show how to translate our algorithmic framework into domains with high-dimensional observations. To do so, we turn to variational autoencoders (VAEs) [76]. In the VAE setting, we are concerned with learning a compact latent data representation, \(z\), that captures a high-dimensional observation space, \(x\) through the use of two parameterized functions \(q_\psi(z|x)\) and \(p_\theta(x|z)\). The pair represent a probabilistic encoder and decoder, typically captured by two separate neural networks, where the former maps data to a latent representation and the latter maps from \(z\) to the original observation. Traditionally, the two models are trained jointly to optimize the evidence lower bound objective (ELBO), which maximizes \(\mathbb{E}_{q_\psi(z|x)}[\log p_\theta(x|z)]\) to facilitate reconstruction of the original data and minimizes \(D_{\text{KL}}(q_\psi(z|x) \parallel p_\theta(z))\) to keep \(q_\psi(z|x)\) close to some prior \(p_\theta(z)\) over latent codes. Both \(q_\psi(z|x)\) and \(p_\theta(z)\) are commonly treated as Gaussian to use the Gaussian reparameterization trick [76]. Since the ELBO is optimized in expectation over the data distribution, \(p(x)\), we can leverage a known result regarding the KL-divergence term.
is compromised under a low setting of $\beta$ that the quality of state reconstruction (each state is a row of four consecutive game screens) and the nature of the resulting abstraction. In the visualizations of the abstraction, we observe $\pi$ of 0.0001. During training, the demonstrator’s policy (mirroring that of Figure 14) between choice of $\beta$, success in approximating the demonstrator policy, and the nature of the resulting abstraction. The model is trained via Equation 60 for 2000 episodes using the Adam optimizer [75] with a learning rate of 0.0001. During training, the demonstrator’s policy ($\pi_d$) controls the MDP. We present results showcasing the effect of $\beta$ on compression and performance in Figure 15; we find a relationship (mirroring that of Figure 14) between choice of $\beta$, success in approximating the demonstrator policy, and the nature of the resulting abstraction. In the visualizations of the abstraction, we observe that the quality of state reconstruction (each state is a row of four consecutive game screens) is compromised under a low setting of $\beta$ (prioritizing compression), whereas a higher value of $\beta$

\[ 13: \text{return } \phi_{t+1}, \pi_{\phi,t+1} \]

\[ \mathbb{E}_{p(x)}[D_{KL}(q_{\phi}(z|x) \| p_0(z))] = \]

\[ I(X;Z) + D_{KL}(q(z) \| p(z)) \geq I(X;Z). \]

We treat $x$ as the ground state representation $S$ and $z$ as the abstract state $S_\phi$. We derive a new objective function that serves as a variational upper bound to the stochastic IB (SIBS) objective derived in Equation 54:

\[ \min_{\phi} \mathbb{E}_{p_d(s)}[D_{KL}(q_{\phi}(S_\phi|S) \| p_0(S_\phi))] + \]

\[ \beta D_{KL}(\pi_d(a \| s) \| \pi_\phi(a \| s))], \]

\[ \geq \min_{\phi} \left( I(S;S_\phi) + \beta \mathbb{E}_{p_d(s)}[D_{KL}(\pi_d(a \| s) \| \pi_\phi(a \| s))] \right), \]

where the upper bound follows directly from Equation 59.

To make use of this upper bound, we first create a demonstrator policy $\pi_d$ for the Atari game Breakout [21] using A2C [102]. A Gaussian VAE agent is then trained with a separate architecture that has the same first four layers as the A2C agent before mapping out to a mean and covariance (in $\mathbb{R}^{25}$). Instead of reconstructing states, our decoder serves as an abstract policy network, mapping to a final distribution over the primitive actions, $\pi_\phi(a \| s)$, which is really $\pi_\phi(a \| z)$. The model is trained via Equation 60 for 2000 episodes using the Adam optimizer [75] with a learning rate of 0.0001. During training, the demonstrator’s policy ($\pi_d$) controls the MDP. We present results showcasing the effect of $\beta$ on compression and performance in Figure 15; we find a relationship (mirroring that of Figure 14) between choice of $\beta$, success in approximating the demonstrator policy, and the nature of the resulting abstraction. In the visualizations of the abstraction, we observe that the quality of state reconstruction (each state is a row of four consecutive game screens) is compromised under a low setting of $\beta$ (prioritizing compression), whereas a higher value of $\beta$
Figure 15: The mean reward over 100 evaluation episodes of state-abstraction–policy \((\phi, \pi_\phi)\) combinations found by our VAE-approximation to Sibs for different values of \(\beta\) (a), and attempted state (a stack of four consecutive game screens) reconstructions using fixed state abstractions found when \(\beta = 2\) and \(\beta = 2048\) (b). Notice the poorer reconstruction quality in light of a low \(\beta\) (encouraging more compression).

preserves significantly more information (paddle position and shape of bricks), leading to higher-quality reconstruction. We suspect that increasing the size of our latent bottleneck could help close the gap between the abstract and demonstrator policies.

4.3.6 Related Work

Follow-up work on the Information Bottleneck explores connections between the IB and RL. Rubin et al. [117] introduce the control information of a policy for a given state, which leads to the trade-off between policy-information and policy value, expressed as follows:

\[
F_\pi(s_0; \beta) = I_\pi(s_0) - \beta V_\pi(s_0). \tag{61}
\]

Using dynamic programming, they compute the policy that makes this trade off. Tishby and Polani [138] introduce the “information-to-go” function, which denotes the informational regret of choosing one policy over another. They then construct Bellman-like equations for deploying this quantity. Both of their methods and objectives are similar to ours, with one major departure: we consider the abstractions that achieve the appropriate trade-off, not policies, though the two are connected.

Slonim and Tishby [125] extends the IB to build a hierarchy of clusters on the input space such that each cluster can predict a target \(Y\). We differ in that we focus on state abstraction for sequential decision making, though the two kinds of clustering are related. We instead build our proxy objective from IB/DIB as they naturally scale to more general settings, such as our VAE formulation and proposed extensions (Section 5.3).

RD theory has been used in the context of optimal control [86, 114, 107]. Our main departures consist of our formalism (RL vs. control), our use of a general, discrete transition matrix to model the environment (instead of linear-Gaussian dynamics), and our objective of maximizing value (or fidelity to the demonstrator policy) instead of properties like asymptotic system stability.

Further work has explored learning abstractions or state representations from a demonstrator [34, 98, 88]. We differ in our close attachment to Rate-Distortion theory, through which we can
target new lower bounds on the trade-off between an abstraction’s induced compression and ability to make useful decisions. To our knowledge, no other work has presented results on this trade-off about abstraction for RL.

Our extension to high-dimensional observation spaces draws inspiration from Higgins et al. [56], who introduced the $\beta$-VAE model for controlling the capacity of the learned latent representation via a hyperparameter. We show how to extend our methods to high-dimensional domains using a VAE for learning abstract state spaces over high-dimensional observations. We leverage a variational upper bound to a proxy objective, similarly to the $\beta$-VAE.

Many questions remain open. Most notably, we are interested in a fully realized version of the introduced paradigm in which a demonstrator policy is not required. We return to this point in Section 5.

What about action abstractions? Thus far many of our results focus on how state abstractions might satisfy the three desiderata. We next investigate how hard it is to find action abstractions that accelerate planning.

4.4 Action Abstractions that Minimize Planning Time

The right set of options allows planning to probe more deeply into the search space with a single computation. Thus, if options are chosen appropriately, planning algorithms can find good plans with less computation.

Indeed, previous work has offered substantial support that abstract actions can accelerate planning. However, little is known about how to find the right set of options. Prior work often seeks to codify an intuitive notion of what underlies an effective option, such as identifying relatively unusual states [121], identifying bottleneck states or high-betweenness states [123, 122, 17, 103], finding repeated policy fragments [112], or finding states that often occur on successful trajectories [97, 18]. While such intuitions often capture important aspects of the role of options in planning, the resulting algorithms are somewhat heuristic in that they are not based on optimizing any precise performance-related metric; consequently, their relative performance can only be evaluated empirically.

In this work, we aim to formalize what it means to find the set of options that is optimal for planning, and to use the resulting formalization to develop a practical algorithm with a principled theoretical foundation. Specifically, we consider two settings that describe the problem of finding the right set of options: 1) finding the set of $k$ options that minimize the number of VI iterations until convergence, and 2) computing the smallest set of options so that planning converges in less than a given maximum of $\ell$ VI iterations. We show that both problems are NP-hard, even for a deterministic MDP, and therefore harder than directly solving the MDP, which takes polynomial time [92]. We then provide a polynomial-time approximation algorithm for a subclass of each problem, A-MIMO and A-MOMI, that computes approximately optimal options for MDPs with bounded return and goal states. We prove that both algorithms have bounded suboptimality for deterministic tasks. These algorithms are not practical for speeding up run-time performance, as they are computationally harder than solving the MDP itself. The purpose of the algorithm is to analyze and evaluate the utility of options generated by heuristic methods. Finally, we empirically evaluate the performance of two heuristic approaches for option discovery, betweenness options
Figure 16: A single option can encode multiple unrelated behaviors. The dark circles indicate where the option can be initiated ($s_1$ & $s_6$) and terminated ($s_2$ & $s_9$), whereas the lighter circles denote the states visited by the option policy when applied in the respective initiating state.

[122] and eigenoptions [93], against A-MIMO, A-MOMI and the optimal options in standard grid domains.

The problem we consider is to find a set of options to add to the set of primitive actions that seek to minimize the number of iterations required for VI to converge:

\[ L(O) \]

\[ \text{The number of iterations} \quad L(O) \quad \text{of a value-iteration algorithm using option set} \quad A \cup O, \quad \text{with} \quad O \quad \text{a non-empty set of options, is the smallest} \quad b \quad \text{at which} \quad |V_b(s) - V^*(s)| < \epsilon \quad \text{for all} \quad s \in S, \quad \text{where} \quad \epsilon \quad \text{is a tolerance to the suboptimality of the plan quality.} \]

4.4.1 Point options.

The options formalism is immensely general. Due to its generality, a single option can actually encode several completely unrelated sets of different behaviors. Consider the nine-state example MDP pictured in Figure 16; a single option can in fact initiate, make decisions in, and terminate along entirely independent trajectories. As we consider more complex MDPs (which, as discussed earlier, is often a motivation for introducing options), the number of independent behaviors that can be encoded by a single option increases further still.

As a result, it can be difficult to reason about the impact of adding a single option, in the traditional sense. As the MDP grows larger, a combinatorial number of different behaviors can emerge from “one” option. Consequently, it is difficult to address the question: which single option helps planning? As MDPs grow large, one option can in fact encode a huge number of possible, independent behaviors. Thus, we instead introduce and study “point options”, which only allow for a single continuous stream of behavior:

\[ \text{We can ensure} \quad |V^*(s) - V_i(s)| < \epsilon \quad \text{by running VI until} \quad |V_{i+1}(s) - V_i(s)| < \epsilon(1 - \gamma)/2\gamma \quad \text{for all} \quad s \in S \quad [146]. \]
Definition 24 (Point option): A point option is any option whose initiation set and termination set are each true for exactly one state each:

\[
\begin{align*}
|\{s \in S : \mathcal{I}(s) = 1\}| &= 1, \quad (62) \\
|\{s \in S : \beta(s) > 0\}| &= 1, \quad (63) \\
|\{s \in S : \beta(s) = 1\}| &= 1. \quad (64)
\end{align*}
\]

We let \( \mathcal{O}_p \) denote the set containing all point options.

For simplicity, we denote the initiation state as \( \mathcal{I}_o \) and the termination state as \( \beta_o \) for a point option \( o \).

To plan with a point option from state \( s \), the agent runs value iteration using a model \( Q(s, o) = R(s, o) + \gamma(o)V(s') \) in addition to the backup operations by primitive actions. We assume that the model of each option is given to the agent and ignore the computation cost for computing the model for the options.

Point options are a useful subclass to consider for several reasons. First, a point option is a simple model for a temporally extended action. Second, the policy of the point option can be calculated as a path-planning problem for deterministic MDPs. Third, any other options with a single termination state with termination probability 1 can be represented as a collection of point options. Forth, a point option has a fixed amount of computational overhead per iteration.

4.4.2 Complexity Results

Our main results focus on two computational problems:

1. \( \text{MINITERMAXOPTION (MIMO)} \): Which set of \( k \) or fewer options minimizes the number of iterations to convergence?

2. \( \text{MINOPTIONMAXITER (MOMI)} \): Which set of options let value iteration converge in at most \( \ell \) iterations?

More formally, MIMO is defined as follows.

**Definition 25 (MIMO):** The MINITERMAXOPTION problem:

Given an MDP \( M \), a non-negative real-value \( \epsilon \), and an integer \( k \), return \( \mathcal{O} \) that minimizes \( L(\mathcal{O}) \), subject to \( \mathcal{O} \subseteq \mathcal{O}_p \) and \( |\mathcal{O}| \leq k \).

We then consider the complementary optimization problem: compute the smallest set of point options such that planning time is guaranteed to fall below some threshold. Motivated by this scenario, the second problem we study is MINOPTIONMAXITER (MOMI).

**Definition 26 (MOMI):** The MINOPTIONMAXITER problem:

Given an MDP \( M \), a non-negative real-value \( \epsilon \), and an integer \( \ell \), return \( \mathcal{O} \) that minimizes \( |\mathcal{O}| \) subject to \( \mathcal{O} \subseteq \mathcal{O}_p \) and \( L(\mathcal{O}) \leq \ell \).

We now introduce our main result, which shows that both MIMO and MOMI are NP-hard.
Theorem 12. MIMO and MOMI are NP-hard.

In light of the computational difficulty of both problems, the appropriate approach is to find tractable approximation algorithms. However, from the reduction from the set-cover optimization problem, we can show that MOMI cannot be approximated to a constant factor. More precisely:

Theorem 13. MOMI cannot be approximated to a constant factor in polynomial time unless P = NP.

4.4.3 Generalizations of MIMO and MOMI

A natural question is whether Theorem 12 extends to more general option-construction settings. We consider two possible extensions, which we believe offer significant coverage of finding optimal options for planning in general.

We first consider the case where the options are not necessarily point options. Using the space of all options $O_{\text{all}}$ we generalize MIMO as follows:

**Definition 27 (MIMO\textsubscript{gen}):**

Given an MDP $M$, a non-negative real-value $\epsilon$, $O' \subseteq O_{\text{all}}$, and an integer $\ell$, return $O$ minimizing $L(O)$ such that $O \subseteq O'$ and $|O| \leq k$.

**Theorem 14.** MIMO\textsubscript{gen} and MOMI\textsubscript{gen} are NP-hard.

The proof follows from the fact that MIMO\textsubscript{gen} is a superset of MIMO MOMI\textsubscript{gen} is a superset of MOMI.

We next consider the multi-task generalization, where we aim to minimize the average number of iterations to solve a problem $M$ sampled from a distribution of MDPs, $D$:

**Definition 28 (MIMO\textsubscript{multi}):**

Given a distribution of MDPs $D$, $O' \subseteq O_{\text{all}}$, a non-negative real-value $\epsilon$, and an integer $\ell$, return $O$ that minimizes $E_{M \sim D}[L_M(O)]$ such that $O \subseteq O'$ and $|O| \leq k$.

**Theorem 15.** MIMO\textsubscript{multi} and MOMI\textsubscript{multi} are NP-hard.

The proof follows from the fact that MIMO\textsubscript{multi} is a superset of MIMO\textsubscript{gen} and MOMI\textsubscript{multi} is a superset of MOMI\textsubscript{gen}.

In summary, the problem of computing optimal behavioral abstractions for planning is intractable.

4.4.4 Approximation Algorithms

We now provide polynomial-time approximation algorithms, A-MIMO and A-MOMI, to solve MIMO and MOMI, respectively. Both algorithms have bounded suboptimality slightly worse than a constant factor for deterministic MDPs. We assume that (1) there is exactly one absorbing state $g \in S$ with $T(g,a,g) = 1$ and $R(g,a) = 0$, and every optimal policy eventually reaches $g$ with probability 1, (2) there is no cycle with a positive reward involved in the optimal policy’s trajectory. That is, $V_\pi^+(s) := E[\sum_{t=0}^{\infty} \max\{0, R(s,a)\}] < \infty$ for all policies $\pi$. Note that we can
A-MIMO

4.4.5 A-MIMO

The outline of the approximation algorithm for MIMO, A-MIMO, is as follows.

1. Compute an asymmetric distance function \( d_\epsilon(s,s'): S \times S \to \mathbb{N} \) representing the number of iterations for a state \( s \) to reach its \( \epsilon \)-optimal value if we add a point option from a state \( s' \) to a goal state \( g \).

2. Using this distance function, solve an asymmetric \( k \)-center problem, which finds a set of center states that minimizes the maximum number of iterations for every state to converge.

3. Generate point options with initiation states set to the center states in the solution of the asymmetric \( k \)-center, and termination states set to the goal.

(1) First, we compute a distance function \( d_\epsilon: S \times S \to \mathbb{N} \), defined as follows:

**Definition 29** (Distance \( d_\epsilon(s_i,s_j) \)): \( d_\epsilon(s_i,s_j) \) is the number of iterations for \( s_i \) to reach \( \epsilon \)-optimal if we add a point option from \( s_j \) to \( g \) minus one.

More formally, let \( d'_\epsilon(s_i) \) denote the number of iterations needed for the value of state \( s_i \) to satisfy \( |V(s_i) - V^*(s_i)| < \epsilon \), and let \( d''_\epsilon(s_i,s_j) \) be an upper bound of the number of iterations needed for the value of \( s_i \) to satisfy \( |V(s_i) - V^*(s_i)| < \epsilon \), if the value of \( s_j \) is initialized such that \( |V(s_j) - V^*(s_j)| < \epsilon \). We define \( d_\epsilon(s_i,s_j) := \min(d'_\epsilon(s_i)-1,d''_\epsilon(s_i,s_j)) \). For simplicity, we use \( d \) to denote the function \( d_\epsilon \). Consider the following example.

**Example 1.** Table 2 is a distance function for the MDP shown in Figure 17. For a deterministic MDP, \( d_0(s) \) corresponds to the number of edge traversals from state \( s \) to \( g \), where we have edges only for those that corresponds to the state transition by the optimal actions. The quantity \( d_0(s,s') - 1 \) is the minimum of \( d_0(s) \) and one plus the number of edge traversals from \( s \) to \( s' \).

Note that we only need to solve the MDP once to compute \( d \). \( d(s,s') \) can be computed once you solved the MDP without any options and store all value functions \( V_i \) for \( i = 1, \ldots, b \) until convergence as a function of \( V_i \): \( V_i(s) = f(V_i(s_0), V_i(s_1), \ldots) \). If we add a point option from \( s' \) to \( g \), then \( V_i(s') = V^*(s') \). Thus, \( d(s,s') \) is the smallest \( i \) where \( V_i(s) \) reaches \( \epsilon \)-optimal if we replace \( V_i(s') \) with \( V^*(s') \) when computing \( V_i(s) \) as a function of \( V_i \).

(2) We exploit this characteristic of \( d \) and solve the asymmetric \( k \)-center problem [108] on \((U,d,k)\) to get a set of centers, which we use as initiation states for point options. The asymmetric \( k \)-center problem is a generalization of the metric \( k \)-center problem where the function \( d \) obeys the triangle inequality, but is not necessarily symmetric.

50
**Definition 30 (AsymKCenter):**
Given a set of elements $U$, a function $d:U \times U \rightarrow \mathbb{N}$, and an integer $k$, return $C$ that minimizes $P(C) = \max_{s \in U} \min_{c \in C} d(s,c)$ subject to $|C| \leq k$.

We solve the problem using a polynomial-time approximation algorithm proposed by Archer [10]. The algorithm has a suboptimality bound of $O(\log^* k)$ where $k < |U|$ and it is proven that the problem cannot be solved within a factor of $\log^* |U| - \theta(1)$ unless $P=NP$ [33]. As the procedure by Archer [10] often finds a set of options smaller than $k$, we generate the rest of the options by greedily adding $\log k$ options at once. See the supplementary material for details.

(3) We generate a set of point options with initiation-states set to one of the centers and the termination state set to the goal state of the MDP. That is, for every $c$ in $C$, we generate a point option starting from $c$ to the goal state $g$.

![Diagram](image)

Figure 17: Example for A-MIMO with $k = 2$. Discovered options are denoted by the dashed lines.

<table>
<thead>
<tr>
<th>$s \backslash s'$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
<th>$s_5$</th>
<th>$s_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$s_2$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$s_3$</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$s_4$</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$s_5$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$s_6$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: $d_0(s, s')$ for Figure 17.

**Theorem 16.** A-MIMO has the following properties:

1. A-MIMO runs in polynomial time.
2. If the MDP is deterministic, it has a bounded suboptimality of $\log^* k$.
3. The number of iterations to solve the MDP using the option set acquired is upper bounded by $P(C)$.

### 4.4.6 A-MOMI

We now describe a polynomial-time approximation algorithm, A-MIMO, based on using set cover to solve MOMI. The overview of the procedure is as follows.
1. Compute \( d \) for every state pair.

2. For every state \( s_i \), compute a set of states \( X_{s_i} \) within \( \ell - 1 \) distance of reaching \( s_i \). The set \( X_{s_i} \) represents the states that converge within \( \ell \) steps if we add a point option from \( s_i \) to \( g \).

3. Let \( \mathcal{X} \) be a set of \( X_{s_i} \) for every \( s_i \in S \setminus X_g^+ \), where \( X_g^+ \) is a set of states that converges within \( \ell \) without any options (thus can be ignored).

4. Solve the set-cover optimization problem to find a set of subsets that covers the entire state set using the approximation algorithm by Hochbaum [57]. This process corresponds to finding a minimum set of subsets \( \{X_{s_i}\} \) that makes every state in \( S \) converge within \( \ell \) steps.

5. Generate a set of point options with initiation states set to one of the center states in the solution of the asymmetric \( k \)-center, and termination states set to the goal.

**Theorem 17.** A-MOMI has the following properties:

1. A-MOMI runs in polynomial time.

2. It guarantees that the MDP is solved within \( \ell \) iterations using the option set acquired by A-MOMI \( O \).

3. If the MDP is deterministic, the option set is at most \( \max_{s \in S} X_s \) times larger than the smallest option set possible to solve the MDP within \( \ell \) iterations.

Note that the approximation bound for deterministic MDP may be improved by improving the approximation algorithm for the set cover. It is proven to be NP-hard to approximate up to a factor of \((1 - o(1)) \log n \) [41], thus there may be an improvement on the approximation ratio for the set cover problem, which will also improve the approximation ratio of A-MIMO.

### 4.4.7 Experiments

We evaluate the performance of the value-iteration algorithm using options generated by the approximation algorithms on several grid-based simple domains. Our code is freely available online.

We ran the experiments on an \( 11 \times 11 \) four-room domain and a \( 9 \times 9 \) grid world with no walls. In both domains, the agent’s goal is to reach a specific square. The agent can move in the usual four directions but cannot cross walls.

First, we visualize a variety of option types, including the optimal point options, those found by our approximation algorithms, and several option types proposed in the literature. We computed the optimal set of point options by enumerating every possible set of point options and picking the best. We are only able to find optimal solutions up to four options within 10 minutes, while the approximation algorithm could find any number of options within a few minutes. Both betweenness options and eigenoptions are discovered by a polynomial time algorithm, thus able to discover within a few minutes. Figure 18 shows the optimal and bounded suboptimal set of options computed by A-MIMO. See the supplementary material for visualizations for the \( 9 \times 9 \) grid domain.

Figure 18c shows the four bottleneck states with highest shortest-path betweenness centrality in the state-transition graph [122]. Interestingly, the optimal options are quite close to the bottleneck states in the four-room domain, suggesting that bottleneck states are also useful for planning as a heuristic to find important subgoals.
Figure 18: Comparison of the optimal point options with options generated by the approximation algorithm A-MIMO. The green square represents the termination state and the blue squares the initiation states. Observe that the approximation algorithm is similar to that of optimal options. Note that the optimal option set is not unique: there can be multiple optimal option sets, and we are visualizing just one returned by the solver.

Figure 18f shows the set of subgoals discovered by graph Laplacian analysis following the method of Machado et al. [93]. While they proposed to generate options to travel between subgoals for reinforcement learning, we generate a set of point options from each subgoal to the goal state as that is a better use of the subgoals for planning setting.

Next, we run Value Iteration using the set of options generated by A-MIMO and A-MOMI. Figures 19a and 19b show the number of iterations on the four-room and the 9 × 9 grids using a set of options of size k. The experimental results suggest that the suboptimal algorithm finds set of options similar to, but not quite as good as, the optimal ones. For betweenness options and eigenoptions, we evaluated every subset of options among the four and present results for the best subset found. Because betweenness options are placed close to the optimal options, the performance is close to optimal especially when the number of options are small.

In addition, we used A-MOMI to find a minimum option set to solve the MDP within the given number of iterations. Figures 19c and 19d show the number of options generated by A-MOMI compared to the minimum number of options.
Figure 19: MIMO and MOMI evaluations. Parts (a)–(c) show the number of iterations for VI using options generated by A-MIMO. Parts (d)–(f) show the number of options generated by A-MOMI to ensure the MDP is solved within a given number of iterations. OPT: optimal set of options. APPROX: a bounded suboptimal set of options generated by A-MIMO an A-MOMI. BET: betweenness options. EIG: eigenoptions.

We now discuss the road ahead.
5 Future Work

Many questions remain. With my final projects, I propose three core directions to pursue:

1. Better understand the relationship between state, action, and hierarchical abstraction.
   - Project One: Construct simple hierarchical abstractions from state abstraction.
   - Project Two: Development of a new, simpler option model.

2. Develop algorithms for novel kinds of state abstraction where the current theory fails.
   - Project One: Learn state abstractions for environments with continuous state.
   - Project Two: Flesh out theory for dynamic state abstractions. Better understand whether dynamic abstraction can help, in principle, and if so, show why and how.

3. Follow up to existing work
   - Project One: Abstraction-as-Compression 2.0, where we remove the demonstrator policy.

Nearly all of these are already underway in some capacity. I next discuss each of these in more detail.

5.1 Simple Hierarchical RL

The majority of the results introduced so far are focused on state abstraction, except for the last set of results discussed, which target action abstraction. A natural question still stands: can the same desirable properties we have shown can obtain of state abstractions, also obtain of hierarchies of abstractions?

As my first project (which is already ongoing, joint with Nate Umbanhowar and Michael Littman), I plan to explore the following simple recipe for constructing abstraction hierarchies, for a given aggregation predicate $p$:

1. Compute $\phi_p$ via Algorithm 2.
2. Compute $\omega$ via Algorithm 3, which effectively computes options to transition between adjacent abstract states.
3. Compute the abstract MDP, $M_{\phi,\omega}$, via Algorithm 4.

The above three steps can be repeated arbitrarily until no compression takes place at step 1. The result is a hierarchy of MDPs, resulting simply from a choice of an abstraction predicate.

The research will involve analyzing the structures that result from different aggregation predicates, both through analysis of the value loss of the induced hierarchies, their planning efficiency, and their empirical performance compared to baseline algorithms. Ultimately, I view this recipe as a simple means of constructing hierarchies that satisfy the desiderata.
Algorithm 2 Compute $\phi$

**INPUT:** $M, p$

**OUTPUT:** $\phi_p$

1. $EdgeHash = \text{hash}(s: \{s': p_M(s, s')\})$
2. $PredicateGraph = (S, E)$
3. $\{c_1, \ldots, c_n\} = \text{clique\_cover}(G)$
4. for $s \in S$ do
5. $\phi_p(s) = \text{get\_clique\_number}(s, \{c_1, \ldots, c_n\})$
6. end for
7. return $\phi_p$

Algorithm 3 Compute Options

**INPUT:** $M, \phi$

**OUTPUT:** $O$

1. $O = \{ \}$
2. for $(s_{\phi, 1}, s_{\phi, 2}) \in S_\phi \times S_\phi$ do ▷ Compute clique.
3. $I(s) \doteq 1 \{ \phi(s) = s_{\phi, 1} \}$
4. $\beta(s) \doteq 1 \{ \phi(s) = s_{\phi, 2} \} \cdot (1 - \gamma)$
5. $R_{1, 2}(s) \doteq 1 \{ \phi(s) = s_{\phi, 2} \}$
6. $s_0 = S_\phi \text{get\_state\_in\_cluster}(s_{\phi, 1})$
7. $\pi_1^* = \text{arg max}_\pi V^\pi_{R_{1, 2}}(s_0)$
8. $o_{1, 2} = \text{Option}(I, \beta, \pi)$
9. $O.\text{add}(o_{1, 2})$
10. end for
11. for $o \in O$ do ▷ Remove redundant options.
12. if is_redundant($o, O, M, \phi$) then
13. $O.\text{remove}(o)$
14. end if
15. end for
16. return $O$

Algorithm 4 Compute $M_{i+1}$

**INPUT:** $M_i, \phi, \omega$

**OUTPUT:** $M_{i+1}$

1. $S_\phi = \{ \phi(s) : \forall s \in S \}$
2. $A_\omega = O$
3. $\mathcal{R}_s^\omega = \sum_{s_{\phi} \in S_\phi} \mathcal{R}_\omega(s, o)w(s)$
4. $\mathcal{T}_s^\omega = \sum_{x_{\phi} \in S_\phi} \sum_{s_{\phi} \in S_\phi} (s_x) T_{s_{\phi}, x}^\omega w(s)$
5. $M = \left( S_\phi, A_\omega, \mathcal{R}_s^\omega, \mathcal{T}_s^\omega \right)$
6. return $M$

Our earlier work suggests that this recipe does in fact work, and that the resulting hierarchies
5.2 New Multi-Time Model

As we form abstraction hierarchies, the transition and reward models of our abstract actions become difficult to compute. Notably, the multi-time model is a highly general model for an option, allowing for coordination with planning and RL algorithms. Consequently, the main focus of this work will be to introduce an alternate model for an option. Our hope is that these new option models are easier to learn and compute, while still retaining sufficient information to be useful for decision making.

New Option Model. Joint with John Winder, we investigate one assumption that may lead to more effective models: suppose that each option can self report its expected number of time steps (in terms of the level $i - 1$ actions, for an option at level $i$):

$$
\eta_o^s = \mathbb{E}_{o,s,M}[t : \gamma(s_t) | s].
$$

Using this model, we can write down a simplified variant of the multi-time model:

$$
\mathcal{T}(s' | s, o) = \gamma^h \sum_{t=1}^{h} \mathbb{Pr}(s_t = s', \beta(s_t) | s, o)
$$

Using this new model, we hope to:

1. Prove how many data are required to estimate $\kappa$.
2. Prove the value loss of using $\mathcal{T}_\kappa$ in place of $\mathcal{T}_\gamma$.
3. Prove the value loss of using the estimated $\mathcal{T}_\kappa$ instead of $\mathcal{T}_\kappa$.

We first show how many data are required to estimate $\kappa$.
Figure 20: Comparison of the learning performance of a hierarchical RL algorithm using our $\kappa$ variant of the option model compared to the typical option model.

**Theorem 18.** For a given $\delta$, $\varepsilon$, a level $i$ option $o$, a max horizon $h \leq \frac{1}{1-\gamma}$, and state $s$, after $m \geq \frac{h^2 \ln(\frac{4}{\varepsilon})}{2\varepsilon^2}$ executions of $o$ in $s$, we can produce an empirical estimate of the number of $i-1$ time steps taken by the option that is $\varepsilon$ close to the true expected step number with high probability:

$$\Pr\{|\kappa^s_o - \widehat{\kappa}^s_o| < \varepsilon\} > 1 - \delta.$$  (67)

The remaining results define the core of the research project. We have also conducted preliminary experiments on the Taxi domain from [40] comparing our $\kappa$ based option model to regular option models in the context of a hierarchy of MDPs. Results are presented in Figure 20.

### 5.3 Abstraction-as-Compression 2.0

The abstraction-as-compression paradigm presented in Section 4.3 offers the first attempt to characterize the limits of compression in abstraction and RL. While these results represent a major first step toward understanding state abstraction as compression in sequential decision making, we hope to extend this work into a more general learning paradigm for RL. To this end, we now describe two concrete avenues for realizing this broader goal. This work joint with Dilip Arumugam, Kavosh Asadi, Yuu Jinnai, Michael Littman, and Lawson Wong.

**Agent Has Control.** Relaxing the assumption that $\pi_d$ controls the MDP is essential for extending this work to the typical RL setting. We here propose a path toward removing this restriction by focusing on an intermediate goal: define an algorithm with the same properties as DIBS, but with the learning agent’s *non-stationary* policy controlling the underlying MDP instead of $\pi_d$. Ultimately, we seek an algorithm that, after $T < \infty$ iterations, can produce an abstraction–policy pair such that, for some state distribution $p(s)$:

$$\mathbb{E}_{p(s)}\left[V^d(s) - V^{\pi^T}(s)\right] \leq f(\beta, T).$$  (68)

The most challenging aspect of this setup is that the source distribution is no longer fixed since the agent’s policy will change over time as the agent learns and updates both $\phi$ and $\pi_d$. To this end, we highlight one assumption and prove one new result that, together, yield a convergent algorithm for the case where the agent controls the MDP:
Assumption 1. For $t$, the number of calls made to DIBS, let:

$$ X_s^{(t)} = D_{KL}(\pi_d(a \mid s) \parallel \pi_{\phi}^{(t)}(a \mid \phi^{(t)}(s))). $$

(69)

Then:

$$ \mathbb{E}_{s \sim \rho_{\phi}^{(t)}}[X_s^{(t)}] \leq \mathbb{E}_{s \sim \rho_{\phi}^{(t-1)}}[X_s^{(t)}]. $$

(70)

This assumption states that the policy computed via one call to DIBS will yield a new stationary distribution, $\rho_{\phi}^{t}$, such that the current abstract policy and demonstrator policy are not farther apart in expectation under $\rho_{\phi}^{t}$. The thought is that the convergence guarantee of DIBS may ensure that the policy is getting better, for certain values of $\beta$. Along with this assumption, we introduce the following result:

Lemma 8. Given two policies $\pi_1$ and $\pi_2$, if:

$$ \sup_s \sum_a |\pi_1(a \mid s) - \pi_2(a \mid s)| \leq \Delta, $$

(71)

then:

$$ \sum_s |\rho_{\pi_1,\pi_0}(s) - \rho_{\pi_2,\pi_0}(s)| \leq \frac{\Delta \gamma}{1 - \gamma}. $$

(72)

The above proposition suggests that two policies that deviate by a bounded amount are guaranteed to share similar stationary distributions. We can then construct a convergent algorithm for the agent-in-control setting, which we call agent-controlled DIBS (AC-DIBS):

Theorem 19. If Assumption 1 holds, AC-DIBS converges.

We conduct a Four Rooms experiment similar to that of the previous section. Here, we now run the entire process of DIBS for $T$ rounds, each time to convergence, but letting the agent’s initial policy for that round define the stationary state distribution. We call this variant of the algorithm AC-DIBS, for Agent-Controlled-DIBS. Results are presented in Figure 21. Surprisingly, we find AC-DIBS always converges quickly to a point where $\rho_{\phi}^{T}$ is sufficiently close to $\rho_d$, when $\beta > 1$. This finding supports our supposition that Assumption 1, or something similar to it, is true. The pseudocode for AC-DIBS is presented in Algorithm 2.

Following through on a full relaxation of the above assumption is the primary goal for this work going forward.

Figure 21: Agent Controlled DIBS.
Algorithm 2 AC-Dibs

**Input:** $\pi_d, M, \beta, \Delta, \epsilon, \text{rounds}, \text{iters}$

**Output:** $\phi, \pi_\phi$

1: $\forall s : \phi^{(0)}(s) = \text{random.choice}([1, |S|])$  
\hspace*{1cm} $\triangleright$ Initialize
2: $\forall s : \pi_\phi^{(0)}(a | s_\phi) \sim \text{Unif}(A)$
3: $\rho_\phi^{(0)} \leftarrow \text{get\_stationary\_pmf}(M, \pi_\phi^{(0)}, \phi^{(0)})$
4: \hspace{0.5cm} for $i = 1$ to \text{rounds} do
5: \hspace{1cm} $\pi^{(i)}_\phi, \phi^{(i)} = \text{Dibs}(\pi_d, \rho^{(i-1)}_\phi, M, \beta, \Delta, \text{iters})$
6: \hspace{1cm} $\rho_\phi^{(i)}(s) = \text{get\_stationary\_pmf}(M, \pi^{(i)}_\phi, \phi^{(i)})$
7: \hspace{1cm} if $L_1(\rho^{(i-1)}_\phi, \rho^{(i)}_\phi) \leq \epsilon$ then
8: \hspace{1.5cm} break  
\hspace{0.5cm} $\triangleright$ Converged
9: \hspace{0.5cm} end if
10: \hspace{0.5cm} end for
11: \hspace{0.5cm} return $\phi_{t+1}, \pi_{\phi_{t+1}}$

**Multiple MDPs.** Second, we are also interested in using our framework to compute a single abstraction sufficient for representing the demonstrator policy across different, but potentially related tasks, as was explored in Section 4.2. We suppose we are given a set of MDPs $\mathcal{M}$, each sharing a state and action space, but are allowed to vary in $T, R, \gamma$, taking inspiration from the work of Thrun [137], Guestrin et al. [53], Jong and Stone [67].

We conduct an experiment in which $|\mathcal{M}| = 4$, each with a goal in one of the four corners of the world. We run Dibs for each MDP in $\mathcal{M}$, for a fixed $\beta$, and form a master abstraction $\phi_\mathcal{M}$ by taking the intersection across each computed state abstraction. That is, for any state pair $(s_1, s_2)$, for $\phi_i$ computed by Dibs on each MDP, we define $\phi_\mathcal{M}$:

$$
\phi_\mathcal{M}(s_1) = \phi_\mathcal{M}(s_2) = \bigcap_{i=1}^{|\mathcal{M}|} \{ \phi_i(s_1) = \phi_i(s_2) \}. 
$$

Figure 22 shows $\phi_\mathcal{M}$ for different values of $\beta$. Note that the abstraction becomes far more detailed as $\beta$ increases: when $\beta$ is close to 0, the algorithm prioritizes compression, as is reflected by Figure 22a, which only has a single state. Conversely, as $\beta$ increases, we find the algorithm adds more distinctions between states, only grouping those that are close to one another or the near the same wall. (In the diagram, white cells are clusters containing only a singleton state.)

**5.4 State Abstraction with Continuous State**

While many of the formalisms introduced can accommodate representation of continuous state, much of the core theory does not generalize to MDPs that have continuous state. For instance, all of the value loss results presented must assume a countably infinite state space in order for the inductive arguments to go through. In this project, joint with Kavosh Asadi, we explore whether we can develop algorithms for computing useful state abstractions even when the true environmental state space is continuous.
5.5 Dynamic State Abstraction

In this final project, joint with Mark Ho and Daniel Reichman, we hope to explore the potential of a dynamic abstraction for improving RL and planning. We draw inspiration from construal level theory [140]. Computationally bounded agents cannot represent every possibility in perfect detail at every point in time. Rather, they must prioritize how to represent different possible states of the world based on actual state of the world. Inspired by findings in cognitive science, we introduce the notion of ego-centric state representations, in which an agent’s representation of the state space depends on their current state, building off of the earlier work by Konidaris [85], Konidaris and Barto [82]. Such representations are dynamic and can be selected in concert with the planning algorithm that operates over them, providing a principled way for agents to “plan on replanning.” Our hypothesis is that ego-centric abstractions of this form can considerably the planning time needed in order to produce a good policy.
5.6 Timeline

I propose the following timeline for completing each project:

<table>
<thead>
<tr>
<th>Project</th>
<th>Publication Target</th>
<th>Submission Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstraction-as-Compression 1.0</td>
<td>AAAI 2019</td>
<td>September 2018</td>
</tr>
<tr>
<td>Abstraction Cognitive Sci Paper</td>
<td>Behavioral Sciences</td>
<td>November 2018</td>
</tr>
<tr>
<td>Philosophy Thesis</td>
<td>IACAP</td>
<td>January 2019</td>
</tr>
<tr>
<td>Options that Minimize Planning Time</td>
<td>ICML 2019</td>
<td>Winter 2018</td>
</tr>
<tr>
<td>New Option Models</td>
<td>ICML 2019</td>
<td>Winter 2019</td>
</tr>
<tr>
<td>Continuous State Abstraction</td>
<td>ICML 2019</td>
<td>Winter 2019</td>
</tr>
<tr>
<td>State-Action Abstraction Hierarchies</td>
<td>NIPS 2019</td>
<td>Spring 2019</td>
</tr>
<tr>
<td>Dynamic State Abstractions</td>
<td>NIPS 2019</td>
<td>Spring 2019</td>
</tr>
<tr>
<td>Abstraction-as-Compression 2.0</td>
<td>NIPS 2019</td>
<td>Spring 2019</td>
</tr>
<tr>
<td>State Abstraction Survey</td>
<td>Journal</td>
<td>2019</td>
</tr>
<tr>
<td>Thesis Defense</td>
<td></td>
<td>May 2020</td>
</tr>
</tbody>
</table>
References


