Robust Option Learning for Compositional Generalization

Anonymous Author
Anonymous Institution

Abstract

Compositional reinforcement learning is a promising approach for training policies to perform complex long-horizon tasks. Typically, a high-level task is decomposed into a sequence of subtasks and a separate policy is trained to perform each subtask. In this paper, we focus on the problem of training subtask policies in a way that they can be used to perform any task; here, a task is given by a sequence of subtasks. We aim to maximize the worst-case performance over all tasks as opposed to the average-case performance. We formulate the problem as a two agent zero-sum game in which the adversary picks the sequence of subtasks. We propose two RL algorithms to solve this game: one is an adaptation of existing multi-agent RL algorithms to our setting and the other is an asynchronous version which enables parallel training of subtask policies. We evaluate our approach on two multi-task environments with continuous states and actions and demonstrate that our algorithms outperform state-of-the-art baselines.

1 Introduction

Reinforcement learning (RL) has proven to be a promising strategy for solving complex control tasks such as walking (Fujimoto et al., 2018), autonomous driving (Ivanov et al., 2021), and dexterous manipulation (Akkaya et al., 2019). However, a key challenge facing the deployment of reinforcement learning in real-world tasks is its high sample complexity—to solve any new task requires a training a new policy designed to solve that task. One promising solution is *compositional reinforcement learning*, where individual *options* (or *skills*) are first trained to solve simple tasks; then, these options can be composed together to solve more complex tasks (Lee et al., 2018, 2021; Ivanov et al., 2021). For example, if a driving robot learns how to make left and right turns and to drive in a straight line, it can then drive along any route composed of these primitives.

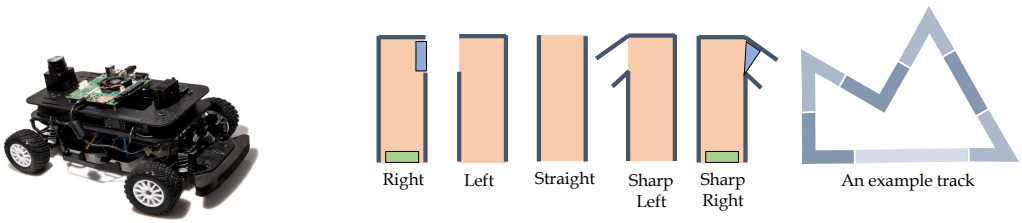
A key challenge facing compositional reinforcement learning is the generalizability of the learned options. In particular, options trained under one distribution of tasks may no longer work well if used in a new task, since the distribution of initial states from which the options are used may shift. An alternate approach is to train the options separately to perform specific subtasks, but options trained this way might cause the system to reach states from which future subtasks are hard to perform. One can overcome this issue by handcrafting rewards to encourage avoiding such states (Ivanov et al., 2021), in which case they generalize well, but this approach relies heavily on human time and expertise.

We propose a novel framework that addresses these challenges by formulating the option learning problem as an adversarial reinforcement learning problem. At a high level, the adversary chooses the task that minimizes the reward achieved by composing the available options. Thus, the goal is to learn a set of *robust options* that perform well across *all* potential tasks. Then, we provide two algorithms for solving this problem. The first adapts existing ideas for using reinforcement learning to solve Markov games to our setting. Then, the second shows how to leverage the compositional structure of our problem to learn options in parallel at each step of a value iteration procedure; in some cases, by enabling such parallelism, we can reduce the computational cost of learning.

We validate our approach on two benchmarks: (i) a rooms environment where a point mass robot must navigate any given sequence of rooms, and (ii) a simulated version of the F1/10th car, where a small racing car must navigate any given racetrack that is composed of several different predefined track segments. In both environments, our empirical results demonstrate that robust options are critical for performing well on a wide variety of tasks.

In summary, our contributions are: (i) a game theoretic formulation of the compositional reinforcement learning problem, (ii) an algorithm for solving this problem in the finite-

Preliminary work. Under review by AISTATS 2023. Do not distribute.



(a) F1/10th Car

(b) Segment Types

Figure 1: F1/10th Environment. The entry and exit regions for the right and sharp right segments are shown in green and blue respectively.

state setting with guaranteed convergence in the limit, (iii) two learning algorithms for continuous-state systems, and (iv) an empirical evaluation demonstrating the effectiveness of our approach.

Motivating example. Let us consider a small scale autonomous racing car shown in Figure 1 (a). We would like to train a controller that can be used to navigate the car through *all* tracks constructed using five kinds of segments; the possible segments are shown in Figure 1 (b) along with an example track. A state of the car is a vector $s = (x, y, v, \theta)$ where (x, y) is its position on the track relative to the current segment, v is its current speed and θ is the heading angle. At any state s, the controller observes the measurements $o(s) \in \mathbb{R}^{1081}$ of a LiDAR sensor and outputs an action $(a, \omega) \in \mathbb{R}^2$ where a is the throttle input and ω is the steering angle. In this environment, completing each segment is considered a subtask and a task corresponds to completing a sequence of segments (which may describe a track)—e.g., `straight` $\rightarrow$ `right` $\rightarrow$ `left` $\rightarrow$ `sharp-right`. Upon completion of a subtask, the car enters the next segment and a change-of-coordinates[1] is applied to the car's state which is now relative to the new segment. The goal here is to learn one option for each subtask such that the agent can perform any task using these options.

If one trains the options independently with the only goal of reaching the end of each segment (e.g., using distance-based rewards), it might (and does) happen that the car reaches the end of a segment in a state that was not part of the initial states used to train the policy corresponding to the next subtask. Therefore, one should make sure that the initial state distribution used during training includes such states as well—either manually or using dataset aggregation (Ross et al., 2011). Furthermore, it is possible that the car reaches a state in the exit region of a segment from which it is challenging to complete the next subtask—e.g., a state in which the car is close to and facing towards a wall. Our algorithm identifies during training that, in order to perform future subtasks, it is better to reach the end of a segment in a configuration where the car is facing straight relative to the next segment. Finally, we want the trained options to be such that they compose well for *every* sequence of segments—i.e., track geometry. Therefore, we are interested in maximizing the worst case performance with respect to the choice of high-level task.

Our framework is broadly applicable in many real-world scenarios. For instance, the F1/10th example can be seen as a miniature version of an autonomous driving scenario where the agent needs to learn to perform maneuvers such as turning left/right, changing lanes etc. Here, the policies for performing these maneuvers need to ensure that the car is in a safe and controllable state for future maneuvers. Another interesting scenario is when a household robot has to perform multiple tasks such as cleaning, cooking etc., but needs to ensure that the policies for performing these tasks leave the house in a favorable state for future tasks—e.g., learning to cook without making the place too dirty (as it might be hard to perform the cleaning task later).

2 Problem Formulation

A *multi-task Markov decision process (MDP)* is a tuple $\mathcal{M} = (S, A, P, \Sigma, R, F, T, \gamma, \eta, \sigma_0)$, where S are the states, A are the actions, $P(s' \mid s, a) \in [0, 1]$ is the probability of transitioning from s to s' on action a, η is the initial state distribution, and $\gamma \in (0, 1)$ is the discount factor. Furthermore, Σ is a set of subtasks and for each subtask $\sigma \in \Sigma$, $R_\sigma : S \times A \rightarrow \mathbb{R}$ is a reward function[2], $F_\sigma \subseteq S$ is a set of final states where the subtask is considered completed and $T_\sigma : F_\sigma \times S \rightarrow [0, 1]$ is the jump probability function; upon reaching a state s in F_σ the system jumps to a new state s' with probability $T_\sigma(s' \mid s)$. For the sake of clarity, we assume[3] that $T_\sigma(s' \mid s) = 0$ for any s' with $s' \in F_{\sigma'}$ for some σ'. Finally, $\sigma_0 \in \Sigma$ is the initial subtask which has to be completed first[4]. A multi-task MDP can

[1] The change-of-coordinates is assumed for simpler modelling and does not affect the LiDAR measurements or the controller.

[2] We can also have a reward function $R_\sigma : S \times A \times S \rightarrow \mathbb{R}$ that depends on the next state but we omit it for clarity of presentation.

[3] This assumption can be removed by adding a fictitious copy of F_σ to S for each $\sigma \in \Sigma$.

[4] When there is no fixed initial subtask, we can add a fictitious initial subtask.

be viewed as a discrete time variant of a hybrid automaton model (Ivanov et al., 2021).

In the case of our motivating example, the set of subtasks is given by $\Sigma = \{\texttt{left}, \texttt{right}, \texttt{straight}, \texttt{sharp-left}, \texttt{sharp-right}\}$ with F_σ denoting the exit region of the segment corresponding to subtask σ. We use the jump transitions T to model the change-of-coordinates performed upon reaching an exit region. The reward function R_σ for a subtask σ is given by $R_\sigma(s, a, s') = -\|s' - c_\sigma\|_2^2 + B \cdot \mathbb{1}(s' \in F_\sigma)$ where c_σ is the center of the exit region and the subtask completion bonus B is a positive constant.

A task τ is defined to be an infinite sequence[5] of subtasks $\tau = \sigma_0\sigma_1 \ldots$, and $\mathcal{T}$ denotes the set of all tasks. For any task $\tau \in \mathcal{T}$, $\tau[i]$ denotes the i^{th} subtask σ_i in τ. In our setting, the task is chosen by the environment nondeterministically. Given a task τ, a configuration of the environment is a pair $(s, i) \in S \times \mathbb{Z}_{\geq 0}$ with $s \notin F_{\tau[i]}$ denoting that the system is in state s and the current subtask is $\tau[i]$. The initial distribution over configurations $\Delta : S \times \mathbb{Z}_{\geq 0} \to [0, 1]$ is given by $\Delta(s, i) = \eta_{\tau[0]}(s)$ if $i = 0$ and 0 otherwise. The probability of transitioning from (s, i) to (s', j) on an action a is given by $\Pr((s', j) \mid (s, i), a) =$

$$\begin{cases} \sum_{s'' \in F_{\tau[i]}} P(s'' \mid s, a) T_{\tau[i]}(s' \mid s'') & \text{if } j = i + 1 \\ P(s' \mid s, a) & \text{if } j = i \quad \& \; s' \notin F_{\tau[i]} \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, the system transitions to the next subtask if the current subtask is completed and stays in the current subtask otherwise. A (deterministic) policy is a function $\pi : S \to A$, where $a = \pi(s)$ is the action to take in state s. Our goal is to learn one policy π_σ for each subtask σ such that the discounted reward over the worst-case task τ is maximized. Formally, given a set of policies $\Pi = \{\pi_\sigma \mid \sigma \in \Sigma\}$ and a task τ, we can define a Markov chain over configurations with initial distribution Δ and transition probabilities given by $P_\Pi((s', j) \mid (s, i)) = \Pr((s, j') \mid (s, i), \pi_{\tau[i]}(s))$. We denote by $\mathcal{D}_\tau^\Pi$ the distribution over infinite sequences of configurations $\rho = (s_0, i_0)(s_1, i_1) \ldots$ generated by τ and Π. Then, we define the objective function as

$$J(\Pi) = \inf_{\tau \in \mathcal{T}} \mathbb{E}_{\rho \sim \mathcal{D}_\tau^\Pi} \Big[\sum_{t=0}^{\infty} \gamma^t R_{\tau[i_t]}(s_t, \pi_{\tau[i_t]}(s_t)) \Big].$$

These definitions can be naturally extended to stochastic policies as well. In our motivating example, choosing a large enough completion bonus B guarantees the discounted reward to be higher for runs in which more subtasks are completed. Our aim is to compute a set of policies $\Pi^* \in \arg\max_\Pi J(\Pi)$. Each subtask policy π_σ defines an option (Sutton et al., 1999) $o_\sigma = (\pi_\sigma, I_\sigma, \beta_\sigma)$ where $I_\sigma = S \setminus F_\sigma$ and $\beta_\sigma(s) = \mathbb{1}(s \in F_\sigma)$. Here, the choice of which option to trigger is made by the environment rather than the agent.

[5] A finite sequence can be appended with an infinite sequence of fictitious subtasks with zero reward.

3 Reduction to Stagewise Markov Games

The problem statement naturally leads to a game theoretic view in which the environment is the adversary. We can formally reduce the problem to a two-agent zero-sum Markov game $\mathcal{G} = (\bar{S}, A_1, A_2, \bar{P}, \bar{R}, \bar{\gamma}, \bar{\eta})$ where $\bar{S} = S \times \Sigma$ is the set of states, $A_1 = A$ are the actions of agent 1 (the agent representing the options) and $A_2 = \Sigma$ are the actions of agent 2 (the adversary). The transition probability function $\bar{P} : \bar{S} \times A_1 \times A_2 \times \bar{S} \to [0, 1]$ is given by $\bar{P}((s', \sigma') \mid (s, \sigma), a_1, a_2) =$

$$\begin{cases} P(s' \mid s, a_1) & \text{if } s \notin F_\sigma \;\&\; \sigma = \sigma' \\ T_\sigma(s' \mid s) & \text{if } s \in F_\sigma \;\&\; \sigma' = a_2 \\ 0 & \text{otherwise.} \end{cases}$$

We observe that the states are partitioned into two sets $\bar{S} = S_1 \cup S_2$ where $S_1 = \{(s, \sigma) \mid s \notin F_\sigma\}$ is the set of states where agent 1 acts (causing a step in $\mathcal{M}$) and $S_2 = \{(s, \sigma) \mid s \in F_\sigma\}$ is the set of states where agent 2 takes actions (causing a change of subtask); this makes $\mathcal{G}$ a stagewise game. The reward function $\bar{R} : \bar{S} \times A_1 \to \mathbb{R}$ is given by $\bar{R}((s, \sigma), a) = R_\sigma(s, a)$ if $s \notin F_\sigma$ and 0 otherwise. The discount factor depends on the state and is given by $\bar{\gamma}(s, \sigma) = \gamma$ if $s \notin F_\sigma$ and 1 otherwise; this is because a change of subtask does not invoke a step in $\mathcal{M}$. The initial state distribution $\bar{\eta}$ is given by $\bar{\eta}(s, \sigma) = \eta(s)\mathbb{1}(\sigma = \sigma_0)$. A run of the game is a sequence $\bar{\rho} = \bar{s}_0 a_0^1 a_0^2 \bar{s}_1 a_1^1 a_1^2 \ldots$ where $\bar{s}_t \in \bar{S}$ and $a_t^i \in A_i$.

A (deterministic) policy for agent i is a function $\pi_i : \bar{S} \to A_i$. Given policies π_1 and π_2 for agents 1 and 2, respectively and a state $\bar{s} \in \bar{S}$ we denote by $\mathcal{D}_{\bar{s}}^{\mathcal{G}}(\pi_1, \pi_2)$ the distribution over runs generated by π_1 and π_2 starting at $\bar{s}$. Then, the value of a state $\bar{s}$ is defined by

$$V^{\pi_1, \pi_2}(\bar{s}) = \mathbb{E}_{\bar{\rho} \sim \mathcal{D}_{\bar{s}}^{\mathcal{G}}(\pi_1, \pi_2)} \Big[\sum_{t=0}^{\infty} \big(\prod_{k=0}^{t-1} \bar{\gamma}(\bar{s}_k) \big) \bar{R}(\bar{s}_t, a_t^1) \Big].$$

We are interested in computing a policy π_1^* maximizing

$$J_{\mathcal{G}}(\pi_1) = \mathbb{E}_{\bar{s} \sim \bar{\eta}}[\min_{\pi_2} V^{\pi_1, \pi_2}(\bar{s})].$$

Given a policy π_1 for agent 1, we can construct a policy π_σ for any subtask σ given by $\pi_\sigma(s) = \pi_1(s, \sigma)$; we denote by $\Pi(\pi_1)$ the set of subtask policies constructed this way. The following theorem connects the objective of the game with our multi-task learning objective; all proofs are in Appendix A.

Theorem 3.1. *For any policy π_1 for agent 1 in $\mathcal{G}$, we have $J(\Pi(\pi_1)) \geq J_{\mathcal{G}}(\pi_1)$.*

Algorithm 1 Asynchronous value iteration algorithm for computing optimal subtask policies.

function AsyncValueIteration($\mathcal{M}, V$)
  while stopping criterion is met **do**
    for $\sigma \in \Sigma$ **do** // in parallel
      Compute $\mathcal{W}_\sigma(V)$
    $V \leftarrow \mathcal{F}_{\texttt{async}}(V)$ // using Equation 3

Therefore, $J_{\mathcal{G}}(\pi_1)$ is a lower bound on the objective $J(\Pi(\pi_1))$ which we seek to maximize. Now, let us define the optimal value of a state $\bar{s}$ by $V^*(\bar{s}) = \max_{\pi_1} \min_{\pi_2} V^{\pi_1,\pi_2}(\bar{s})$. The following theorem shows that it is possible to construct a policy π_1^* that maximizes $J_{\mathcal{G}}(\pi_1)$ from the optimal value function V^*.

Theorem 3.2. *For any policy π_1^* such that for all $(s,\sigma) \in S_1$, $\pi_1^*(s,\sigma) \in \arg\max_{a\in A} \left\{\bar{R}((s,\sigma),a) + \gamma \cdot \sum_{s'\in S} P(s' \mid s,a)V^*(s',\sigma)\right\}$, we have that $\pi_1^* \in \arg\max_{\pi_1} J_{\mathcal{G}}(\pi_1)$.*

3.1 Value Iteration

In this section, we briefly look at two value iteration algorithms to compute V^* which we later adapt in Section 4 to obtain learning algorithms. Let $\mathcal{V} = \{V : S_1 \to \mathbb{R}\}$ denote the set of all value functions over S_1. Given a value function $V \in \mathcal{V}$ we define its extension to all of $\bar{S}$ using $[\![V]\!](s,\sigma) =$

$$\begin{cases} \min_{\sigma'\in\Sigma} \sum_{s'\in S} T_\sigma(s' \mid s)V(s',\sigma') & \text{if } s \in F_\sigma \\ V(s,\sigma) & \text{otherwise.} \end{cases} \quad (1)$$

For a state $s \in F_\sigma$, $[\![V]\!](s,\sigma)$ denotes the worst-case value (according to V) with respect to the possible choices of next subtask σ'. Now, we consider the Bellman operator $\mathcal{F} : \mathcal{V} \to \mathcal{V}$ defined by $\mathcal{F}(V)(s,\sigma) =$

$$\max_{a\in A} \left\{\bar{R}((s,\sigma),a) + \gamma \cdot \sum_{s'\in S} P(s' \mid s,a)[\![V]\!](s',\sigma)\right\} \quad (2)$$

for all $(s,\sigma) \in S_1$. Let us denote by $V^* \downarrow_{S_1}$ the restriction of V^* to S_1. The following lemma follows straightforwardly giving us our first value iteration procedure.

Theorem 3.3. *$\mathcal{F}$ is a contraction mapping with respect to the ℓ_∞-norm on $\mathcal{V}$ and $V^* \downarrow_{S_1}$ is the unique fixed point of $\mathcal{F}$ with $\lim_{n\to\infty} \mathcal{F}^n(V) = V^* \downarrow_{S_1}$ for all $V \in \mathcal{V}$.*

Asynchronous VI. Next we consider an *asynchronous* value iteration procedure which allows us to parallelize computing subtask policies for different subtasks. Given a subtask σ and a value function $V \in \mathcal{V}$, we define a *subtask MDP* $\mathcal{M}_\sigma^V$ which behaves like $\mathcal{M}$ until reaching a final state $s \in F_\sigma$ after which it transitions to a dead state $\perp$ achieving a reward of $[\![V]\!](s,\sigma)$. Formally, $\mathcal{M}_\sigma^V = (S_\sigma, A, P_\sigma, R_\sigma^V, \gamma)$ where $S_\sigma = S \sqcup \{\perp\}$ with $\perp$ being a special dead state. The transition probabilities are given by

$$P_\sigma(s' \mid s,a) = \begin{cases} P(s' \mid s,a) & \text{if } \perp \neq s \notin F_\sigma \\ \mathbb{1}(s' = \perp) & \text{otherwise.} \end{cases}$$

The reward function is given by

$$R_\sigma^V(s,a) = \begin{cases} R_\sigma(s,a) & \text{if } \perp \neq s \notin F_\sigma \\ [\![V]\!](s,\sigma) & \text{if } \perp \neq s \in F_\sigma \\ 0 & \text{otherwise.} \end{cases}$$

We denote by $\mathcal{W}_\sigma(V)$ the optimal value function of the MDP $\mathcal{M}_\sigma^V$. We then define the asynchronous value iteration operator $\mathcal{F}_{\texttt{async}} : \mathcal{V} \to \mathcal{V}$ using

$$\mathcal{F}_{\texttt{async}}(V)(s,\sigma) = \mathcal{W}_\sigma(V)(s). \quad (3)$$

We can show that repeated application of $\mathcal{F}_{\texttt{async}}$ leads to the optimal value function V^* of the game $\mathcal{G}$.

Theorem 3.4. *For any $V \in \mathcal{V}$, $\lim_{n\to\infty} \mathcal{F}_{async}^n(V) \to V^* \downarrow_{S_1}$.*

Since each $\mathcal{W}_\sigma(V)$ can be computed independently, we can parallelize the computation of $\mathcal{F}_{\texttt{async}}$ giving us the algorithm in Algorithm 1. We can also show that it is not necessary to compute $\mathcal{W}_\sigma(V)$ exactly. Let $\mathcal{V}_\sigma = \{\bar{V} : S_\sigma \to \mathbb{R}\}$ be the set of all value functions over S_σ. For a fixed $V \in \mathcal{V}$, let $\mathcal{F}_{\sigma,V} : \mathcal{V}_\sigma \to \mathcal{V}_\sigma$ denote the usual Bellman operator for the MDP $\mathcal{M}_\sigma^V$ given by $\mathcal{F}_{\sigma,V}(\bar{V})(s) =$

$$\max_{a\in A} \left\{R_\sigma^V(s,a) + \gamma \cdot \sum_{s'\in S_\sigma} P_\sigma(s' \mid s,a)\bar{V}(s')\right\}$$

for all $\bar{V} \in \mathcal{V}_\sigma$ and $s \in S_\sigma$. Now for any $V \in \mathcal{V}$ and $\sigma \in \Sigma$, we define a corresponding $V_\sigma \in \mathcal{V}_\sigma$ using $V_\sigma(s) = [\![V]\!](s,\sigma)$ if $s \in S$ and $V_\sigma(\perp) = 0$. Then, for any integer $m > 0$ and $V \in \mathcal{V}$, we can use $\mathcal{F}_{\sigma,V}^m(V_\sigma)$ as an approximation to $\mathcal{W}_\sigma(V)$. Let us define $\mathcal{F}_m : \mathcal{V} \to \mathcal{V}$ using

$$\mathcal{F}_m(V)(s,\sigma) = \mathcal{F}_{\sigma,V}^m(V_\sigma)(s).$$

Intuitively, $\mathcal{F}_m$ corresponds to performing m steps of value iteration in each subtask MDP $\mathcal{M}_\sigma^V$ (which can be parallelized) starting at V_σ. The following theorem guarantees convergence when using $\mathcal{F}_m$ instead of $\mathcal{F}_{\texttt{async}}$.

Theorem 3.5. *For any $V \in \mathcal{V}$ and $m > 0$, $\lim_{n\to\infty} \mathcal{F}_m^n(V) \to V^* \downarrow_{S_1}$.*

4 Learning Algorithms

In this section, we present RL algorithms for solving the game $\mathcal{G}$. We first consider the finite MDP setting for which we can obtain a modified Q-learning algorithm with a convergence guarantee. We then present two algorithms based on Soft Actor Critic (SAC) for the continuous state setting.

4.1 Finite MDP

Assuming finite states and actions, we can obtain a Q-learning variant for solving $\mathcal{G}$ which we call *Robust Option Q-learning*. We assume that jump transitions T are known to the learner; this is usually the case since jump transitions are used to model subtask transitions and (potential) change-of-coordinates within the controller. However, we believe that the algorithm can be easily extended to the scenario where T is unknown.

We maintain a function $Q : S_1 \times A \to \mathbb{R}$ with $Q(s, \sigma, a)$ denoting $Q((s, \sigma), a)$. The corresponding value function V_Q is defined using $V_Q(s, \sigma) = \max_{a \in A} Q(s, \sigma, a)$ and is extended to all of $\bar{S}$ as $[\![V_Q]\!]$. Note that, given a Q-function, the extended value function $[\![V_Q]\!]$ can be computed exactly. Robust Option Q-learning is an iterative process—in each iteration t, it takes a step $((s, \sigma), a_1, a_2, (s', \sigma))$ in $\mathcal{G}$ with $(s, \sigma) \in S_1$ and updates the Q-function using

$$Q_{t+1}(s, \sigma, a_1) \leftarrow (1-\alpha_t) Q_t(s, \sigma, a_1) + \alpha_t(\bar{R}((s, \sigma), a_1) + \gamma [\![V_{Q_t}]\!](s', \sigma)).$$

where Q_t is the Q-function in iteration t and $[\![V_{Q_t}]\!]$ is the corresponding extended value function.

Under standard assumptions on the learning rates α_t, similar to Q-learning, we can show that Robust Option Q-learning converges to the optimal Q-function almost surely. Here, the optimal Q-function is defined by $Q^*(s, \sigma, a) = \bar{R}((s, \sigma), a) + \gamma \sum_{s' \in S} P(s' \mid s, a) V^*(s', \sigma)$ for all $(s, \sigma) \in S_1$. Let $\alpha_t(s, \sigma, a)$ denote the learning rate used in iteration t if $Q_t(s, \sigma, a)$ is updated and 0 otherwise. Then, we have the following theorem.

Theorem 4.1. *If $\sum_t \alpha_t(s, \sigma, a) = \infty$ and $\sum_t \alpha_t^2(s, \sigma, a) < \infty$ for all $(s, \sigma) \in S_1$ and $a \in A$, then $\lim_{t \to \infty} Q_t = Q^*$ almost surely.*

4.2 Continuous States and Actions

In the case of continuous states and actions, we can adapt any Q-function based RL algorithm such as Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2016) or Soft Actor Critic (SAC) (Haarnoja et al., 2018) to our setting. Here we present an SAC based algorithm that we call Robust Option SAC (ROSAC) which is outlined in Algorithm 2. This algorithm, like SAC, adds an entropy bonus to the reward function to improve exploration.

We maintain two Q-functions for each subtask σ, $Q_{\psi_\sigma} : S \to \mathbb{R}$ parameterized by ψ_σ and a target function $Q_{\psi_\sigma^{\texttt{targ}}}$ parameterized by $\psi_\sigma^{\texttt{targ}}$. We also maintain a stochastic subtask policy $\pi_{\theta_\sigma} : S \to \mathcal{D}(A)$ for each subtask σ where $\mathcal{D}(A)$ denotes the set of distributions over A. Given a step (s, a, s') in $\mathcal{M}$ and a subtask σ with $s \notin F_\sigma$, we define the target value by

$$y_\sigma(s, a, s') = R_\sigma(s, a) + \gamma [\![V]\!](s', \sigma)$$

Algorithm 2 Robust Option Soft Actor Critic.

Inputs: Learning rates α_ψ, α_θ, entropy weight β and Polyak rate δ.

function ROSAC(α_ψ, α_θ, β, δ)
  Initialize params $\{\psi_\sigma\}_{\sigma \in \Sigma}$, $\{\psi_\sigma^{\texttt{targ}}\}_{\sigma \in \Sigma}$, $\{\theta_\sigma\}_{\sigma \in \Sigma}$
  Initialize replay buffer $\mathcal{B}$
  for each iteration **do**
    for each episode **do**
      $s_0 \sim \eta$
      $\sigma_0 \leftarrow$ `InitialSubtask`
      for each step t **do**
        $a_t \sim \pi_{\theta_{\sigma_t}}(\cdot \mid s_t)$
        $s_{t+1} \sim P(\cdot \mid s, a)$
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, s_{t+1})\}$
        if $s_{t+1} \in F_{\sigma_t}$ **then**
          $s_{t+1} \sim T_{\sigma_t}(\cdot \mid s_{t+1})$
          $\sigma_{t+1} \leftarrow \texttt{Greedy}(\varepsilon, \arg\min_\sigma \tilde{V}(s_{t+1}, \sigma), \Sigma)$
        else
          $\sigma_{t+1} \leftarrow \sigma_t$
    for each gradient step **do**
      Sample batch $B \sim \mathcal{B}$
      for $\sigma \in \Sigma$ **do**
        $\psi_\sigma \leftarrow \psi_\sigma - \alpha_\psi \nabla_{\psi_\sigma} \mathcal{L}_Q(\psi_\sigma, B)$
        $\theta_\sigma \leftarrow \theta_\sigma - \alpha_\theta \nabla_{\theta_\sigma} \mathcal{L}_\pi(\theta_\sigma, B)$
        $\psi_\sigma^{\texttt{targ}} \leftarrow \delta \psi_\sigma + (1 - \delta) \psi_\sigma^{\texttt{targ}}$

where the value $[\![V]\!](s', \sigma)$ is estimated using $\tilde{V}(s', \sigma) = Q_{\psi_\sigma^{\texttt{targ}}}(s', \tilde{a}) - \beta \log \pi_{\theta_\sigma}(\tilde{a} \mid s')$ with $\tilde{a} \sim \pi_{\theta_\sigma}(\cdot \mid s')$ if $s' \notin F_\sigma$. If $s' \in F_\sigma$, we estimate $[\![V]\!](s', \sigma)$ using $\tilde{V}(s', \sigma) = \min_{\sigma' \in \Sigma} \tilde{V}(s'', \sigma')$ where $\tilde{V}(s'', \sigma') = Q_{\psi_{\sigma'}^{\texttt{targ}}}(s'', \tilde{a}) - \beta \log \pi_{\theta_{\sigma'}}(\tilde{a} \mid s'')$ with $\tilde{a} \sim \pi_{\theta_{\sigma'}}(\cdot \mid s'')$ and $s'' \sim T_\sigma(\cdot \mid s')$. Now, given a batch $B = \{(s, a, s')\}$ of steps in $\mathcal{M}$ we update ψ_σ using one step of gradient descent corresponding to the loss $\mathcal{L}_Q(\psi_\sigma, B) =$

$$\frac{1}{|B|} \sum_{(s,a,s') \in B} (Q_{\psi_\sigma}(s, a) - y_\sigma(s, a, s'))^2$$

and the subtask policy π_{θ_σ} is updated using the loss $\mathcal{L}_\pi(\theta_\sigma, B) =$

$$-\frac{1}{|B|} \sum_{(s,a,s') \in B} \mathbb{E}_{\tilde{a} \sim \pi_{\theta_\sigma}(\cdot \mid s)} \big[Q_{\psi_\sigma}(s, \tilde{a}) - \beta \log \pi_{\theta_\sigma}(\tilde{a} \mid s) \big].$$

The gradient $\nabla_{\theta_\sigma} \mathcal{L}_\pi(\theta_\sigma, B)$ can be estimated using the reparametrization trick if, for example, $\pi_{\theta_\sigma}(\cdot \mid s)$ is a Gaussian distribution whose parameters are differentiable w.r.t. θ_σ. We use Polyak averaging to update the target Q-networks $\{Q_{\psi_\sigma^{\texttt{targ}}} \mid \sigma \in \Sigma\}$.

Note that we do not train a separate policy for the adversary. During exploration, we use the ε-greedy strategy to select subtasks. We first estimate the "worst" subtask for a state s using $\tilde{\sigma} = \arg\min_\sigma \tilde{V}(s, \sigma)$ where $\tilde{V}(s, \sigma)$ is estimated as before. Then the function $\texttt{Greedy}(\varepsilon, \tilde{\sigma}, \Sigma)$ chooses $\tilde{\sigma}$

Algorithm 3 Asynchronous Robust Option SAC.
Inputs: Learning rates α, entropy weight β, Polyak rate δ and number of SAC iterations N.

1: **function** AROSAC(α, β, δ, N)
2: Initialize params $\Psi = \{\psi_\sigma\}_{\sigma\in\Sigma}$, $\Theta = \{\theta_\sigma\}_{\sigma\in\Sigma}$
3: Initialize target params $\Psi^{\texttt{targ}} = \{\psi_\sigma^{\texttt{targ}}\}_{\sigma\in\Sigma}$
4: Initialize replay buffers $\{\mathcal{B}_\sigma\}_{\sigma\in\Sigma}$
5: Initialize $D = \{\}$
6: **for** each iteration **do**
7: $\tilde{V} \leftarrow$ OBTAINVALUEESTIMATOR(Ψ, Θ)
8: **for** $\sigma \in \Sigma$ **do** // in parallel
9: Run SAC($\mathcal{M}_\sigma^{\tilde{V}}, D, \psi_\sigma, \psi_\sigma^{\texttt{targ}}, \theta_\sigma, \alpha, \beta, \delta, N$)
10: Update $\psi_\sigma, \psi_\sigma^{\texttt{targ}}, \theta_\sigma$ and X_σ to new values
11: **for** $\sigma \in \Sigma$ **do**
12: **for** $s \in X_\sigma$ **do**
13: $s' \sim T_\sigma(\cdot \mid s)$ and $D \leftarrow D \cup \{s'\}$

with probability $1-\varepsilon$ and picks a subtask uniformly at random from Σ with probability ε. Furthermore, we can easily impose constraints on the sequences of subtasks considered (e.g., only consider realistic tracks in F1/10th) by restricting the adversary to pick the next subtask σ_{t+1} from a subset $\Sigma_{t+1} \subseteq \Sigma$ of possible substasks—i.e., by performing $\arg\min$ (in Line 14) over Σ_{t+1} instead of Σ.

Asynchronous ROSAC. We can also obtain an asynchronous version of the above algorithm which lets us train subtask policies in parallel. Asynchronous Robust Option SAC (AROSAC) is outlined in Algorithm 3. Here we use one replay buffer for each subtask. We maintain an initial state distribution $\tilde{\eta}$ over S to be used for training every subtask policy $\{\pi_\sigma\}_{\sigma\in\Sigma}$. $\tilde{\eta}$ is represented using a finite set of states D from which a state is sampled uniformly at random. The value function $\tilde{V} : S \times \Sigma \to \mathbb{R}$ is estimated as before. To be specific, in each iteration, an estimate of any value $\tilde{V}(s, \sigma)$ is obtained on the fly using the Q-functions and the subtask policies from the previous iteration.

The SAC subroutine runs the standard Soft Actor Critic algorithm for N iterations on the subtask MDP $\mathcal{M}_\sigma^{\tilde{V}}$ (defined in Section 3)[6] with initial state distribution $\tilde{\eta}$ (defaults to η if $D = \emptyset$). It returns the updated parameters along with states X_σ visited during exploration with $X_\sigma \subseteq F_\sigma$. The states in X_σ are used to update the initial state distribution for the next iteration following the Dataset Aggregation principle (Ross et al., 2011).

5 Experiments

We evaluate our algorithms ROSAC and AROSAC on two multi-task environments; a rooms environment and an F1/10th racing car environment (F110).

Rooms environment. We consider the environment shown in Figure 2 which depicts a room with walls and exits. Initially the robot is placed in the green triangle. The L-shaped obstacles indicate walls within the room that the robot cannot cross. A state of the system is a position $(x, y) \in \mathbb{R}^2$ and an action is a pair (v, θ) where v is the speed and θ is the heading angle to follow during the next time-step. There are three exits: left (blue), right (yellow) and up (grey) reaching each of which is a subtask. Upon reaching an exit, the robot enters another identical room where the exit is identified (via change-of-coordinates) with the bottom entry region of the current room. A task is a sequence of directions—e.g., `left` $\to$ `right` $\to$ `up` $\to$ `right` indicating that the robot should reach the left exit followed by the right exit in the subsequent room and so on. Although the dynamics are simple, the obstacles make learning challenging in the adversarial setting.

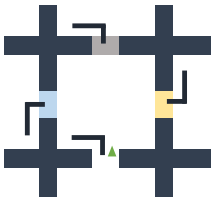
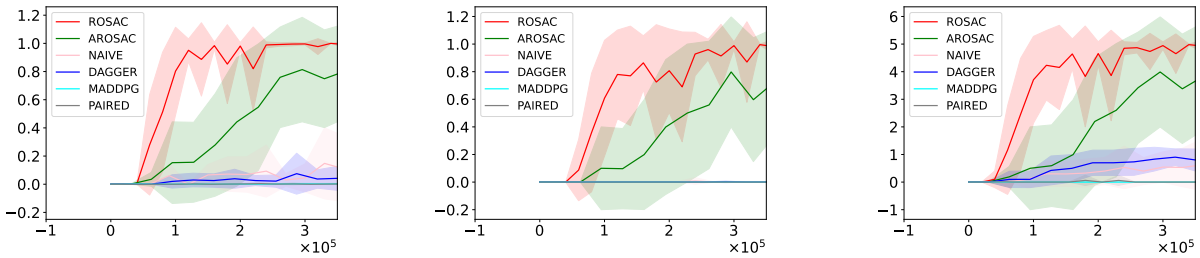


Figure 2: Rooms environment

F1/10th environment. This is the environment in the motivating example. A publicly available simulator (F110) of the F1/10th car is used for training and testing. The policies use the LiDAR measurements from the car as input (as opposed to the state) and we assume that the controller can detect the completion of each segment; as shown in prior work (Ivanov et al., 2021), one can train a separate neural network to predict subtask completion.

Baselines. We compare our approach to four baselines. The baseline NAIVE trains one controller for each subtask with the only aim of completing the subtask, similar to the approach used by Ivanov et al. (2021), using a manually designed initial state distribution. DAGGER is a similar approach which, instead of using a manually designed initial state distribution for training, infers the initial state distribution using the Dataset Aggregation principle (Ross et al., 2011). In other words, DAGGER is similar to AROSAC except that $\mathcal{M}$ is used instead of $\mathcal{M}_\sigma^{\tilde{V}}$ (Line 9 of Algorithm 3) for training the options in each iteration. The MADDPG baseline solves the game $\mathcal{G}$ using the multi-agent RL algorithm proposed by Lowe et al. (2017) for solving concurrent Markov games with continuous states and actions. We use the open-source implementation of MADDPG by the authors. The PAIRED baseline refers to the unsupervised environment design approach proposed by Dennis et al. (2020) in which the multi-task RL problem is viewed as a two-agent zero-sum game similar to our approach (but is not designed for long-horizon and compositional tasks).

[6]Note that it is possible to obtain samples from $\mathcal{M}_\sigma^{\tilde{V}}$ as long can one can obtain samples from $\mathcal{M}$ and membership in F_σ can be decided.

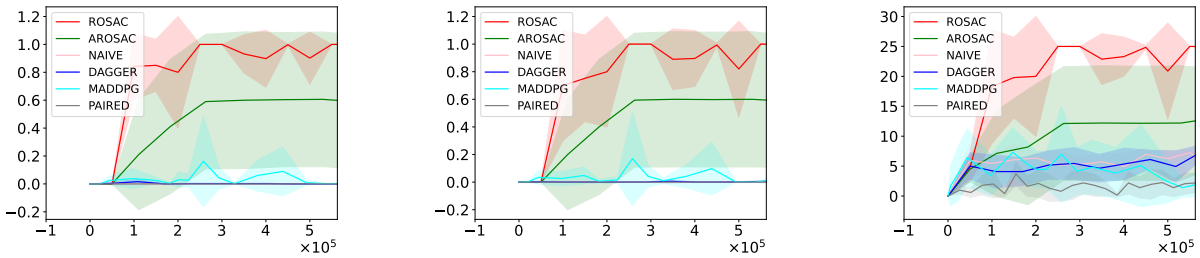


(a) Success probability against random adversary

(b) Success probability against MCTS adversary

(c) Number of subtasks completed against MCTS adversary

Figure 3: Plots for the Rooms environment. x-axis denotes the number of sample steps and y-axis denotes the either the average number of subtasks completed or the probability of completing 5 subtasks. Results are averaged over 10 runs. Error bars indicate $\pm$ standard deviation.



(a) Success probability against random adversary

(b) Success probability against MCTS adversary

(c) Number of subtasks completed against MCTS adversary

Figure 4: Plots for the F1/10th environment. x-axis denotes the number of sample steps and y-axis denotes the either the average number of subtasks completed or the probability of completing 25 subtasks. Results are averaged over 5 runs. Error bars indicate $\pm$ standard deviation.

We implemented PAIRED with PPO as the base algorithm to train the policies of the protagonist and the antagonist which consist of one NN policy per subtask. The adversary is parameterized by the probabilities (logits) of choosing various subtasks at different timesteps and is trained using REINFORCE updates.

Evaluation. We evaluate the performance of these algorithms against two adversaries. One adversary is the random adversary which picks the next subtask uniformly at random from the set of all subtasks. The other adversary estimates the worst sequence of subtasks for a given set of options using Monte Carlo Tree Search algorithm (MCTS) (Kocsis and Szepesvári, 2006). The MCTS adversary is trained by assigning a reward of 1 if it selects a subtask which the corresponding policy is unable to complete within a fixed time budget and a reward of 0 otherwise. For the Rooms environment, we consider subtask sequences of length atmost 5 whereas for the F1/10th environment, we consider sequences of subtasks of length at most 25. We evaluate both the average number of subtasks completed as well as the probability of completing the set maximum number of subtasks.

Results. The plots for the rooms environment are shown in Figure 3 and plots for the F1/10th environment are shown in Figure 4. We can observe that ROSAC and AROSAC outperform other approaches and learn robust options. AROSAC performs worse as compared to ROSAC; however, it has the added benefit of being parallelizable. DAGGER and NAIVE baselines are unable to learn policies that can be used to perform long sequences of subtasks; this is mostly due to the fact that they learn options that cause the system to reach states from which future subtasks are difficult to perform—e.g., in the rooms environment, the agent sometimes reaches the left half of the exits from where it is difficult to reach the right exit in the subsequent room. Although MADDPG uses the same reduction to two-player games as ROSAC, it ignores all the structure in the game and treats it as a generic Markov game. As a result, it learns a separate NN policy for each player which leads to the issue of unstable training, primarily due to the non-stationary nature of the environment observed by either agent. As shown in the plots, this leads to poor performance when applied to the problem of learning robust options. Similarly, PAIRED is unable to learn good options and is likely due to two reasons. Firstly, it does not

use a compositional or a hierarchical approach for training the options, which causes it to scale poorly to long-horizon tasks. Secondly, it relies on "on policy" algorithms (such as PPO) for training which are less efficient than SAC for our environments.

6 Related Work

Options and Hierarchical RL. The options framework (Sutton et al., 1999) is commonly used to model subtask policies as temporally extended actions. In hierarchical RL (Nachum et al., 2018, 2019; Kulkarni et al., 2016; Dietterich, 2000; Bacon et al., 2017; Tiwari and Thomas, 2019), options are trained along with a high-level controller that chooses the sequence of options (plan) to execute in order to complete a specific high-level task. In contrast, our approach aims to train options that work for multiple high-level plans. There is also work on discovering options—e.g., using intrinsic motivation (Machado et al., 2017), entropy maximization (Eysenbach et al., 2018), semi-supervised RL (Finn et al., 2017), skill chaining (Konidaris and Barto, 2009; Bagaria and Konidaris, 2019; Bagaria et al., 2021a), expectation maximization (Daniel et al., 2016) and subgoal identification (Stolle and Precup, 2002). Our approach is complementary to option discovery methods since our algorithm can be used to learn robust policies corresponding to the options which can be used in multiple contexts. There has also been a lot of research on planning using learned options (Abel et al., 2020; Jothimurugan et al., 2021; Precup et al., 1998; Theocharous and Kaelbling, 2004; Konidaris et al., 2014).

Some hierarchical RL algorithms have also been shown to enable few-shot generalization (Jothimurugan et al., 2021) to unseen tasks. Closely related to our work is the work on compositional RL in the multi-task setting (Ivanov et al., 2021) in which the subtask policies are trained using standard RL algorithms in a naive way without guarantees regarding worst-case performance.

Multi-task RL. There has been some work on RL for zero-shot generalization (Vaezipoor et al., 2021; Oh et al., 2017; Sohn et al., 2018; Kuo et al., 2020; Andreas et al., 2017); however, in these works, the learning objective is to maximize average performance with respect to a fixed distribution over tasks as opposed to the worst-case. Most closely related to our work is the line of work on minimax/robust RL (Pinto et al., 2017; Campero et al., 2020; Dennis et al., 2020), which aims to train policies to maximize the worst-case performance across multiple tasks/environments. However, there are a few key differences between existing work and our approach. Firstly, existing methods train a single NN policy to perform multiple tasks as opposed to training composable options for subtasks. As shown in the experiments, training a single policy does not scale well to complex long-horizon tasks. Secondly, these approaches do not provide strong convergence guarantees whereas we provide convergence guarantees in the finite-state setting. Finally, we utilize the structure in the problem to infer an adversary directly from the value functions instead of training a separate adversary as done in previous approaches.

Skill Chaining. Research on skill chaining (Bagaria and Konidaris, 2019; Bagaria et al., 2021b,a; Lee et al., 2021) has led to algorithms for discovering and training options so that they compose well. However, in these approaches, the aim is to compose the options to perform a *specific* task which corresponds to executing the options in specific order(s). In contrast, our objective is independent of specific tasks and seeks to maximize performace across all tasks. Also, such approaches, at best, only provide hierarchical optimality guarantees whereas our algorithm converges to the optimal policy with respect to our game formulation. There has also been work on skill composition using transition policies (Lee et al., 2018); this method assumes that the subtask policies are fixed and learns one transition policy per subtask which takes the system from an end state to a "favourable" initial state for the subtask. However, poorly trained subtask policies can lead to situations in which it is not possible to achieve such transitions. In contrast, our approach trains subtask policies which compose well without requiring additional transition policies.

Multi-agent RL. There has been a lot of research on multi-agent RL algorithms (Lowe et al., 2017; Hu and Wellman, 2003; Hu et al., 1998; Littman, 2001; Perolat et al., 2017; Prasad et al., 2015; Akchurina, 2008) including algorithms for two-agent zero-sum games (Bai and Jin, 2020; Wei et al., 2017; Littman, 1994). In this paper, we utilize the specific structure of our game to obtain a simple algorithm that neither requires solving matrix games nor trains a separate policy for the adversary. Furthermore, we show that we can obtain an asynchronous RL algorithm which enables training options in parallel.

7 Conclusions

We have proposed a framework for training robust options which can be used to perform arbitrary sequences of subtasks. In our framework, we first reduce the problem to a two-agent zero-sum stagewise Markov game which has a unique structure. We utilized this structure to design two algorithms, namely ROSAC and AROSAC, and demonstrated that they outperform existing approaches for training options with respect to multi-task performance. One potential limitation of our approach is that the set of subtasks is fixed and has to be provided by the user. An interesting direction for future work is to address this limitation by combining our approach with option discovery methods.

References

David Abel, Nathan Umbanhowar, Khimya Khetarpal, Dilip Arumugam, Doina Precup, and Michael L. Littman. Value preserving state-action abstractions. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2020.

Natalia Akchurina. Multi-agent reinforcement learning algorithm with variable optimistic-pessimistic criterion. In *ECAI*, volume 178, pages 433–437, 2008.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pages 166–175. PMLR, 2017.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Akhil Bagaria and George Konidaris. Option discovery using deep skill chaining. In *International Conference on Learning Representations*, 2019.

Akhil Bagaria, Jason K Senthil, and George Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *International Conference on Machine Learning*, pages 521–531. PMLR, 2021a.

Akhil Bagaria, Jason K Senthil, Matthew Slivinski, and George Konidaris. Robustly learning composable options in deep reinforcement learning. In *IJCAI*, pages 2161–2169, 2021b.

Yu Bai and Chi Jin. Provable self-play algorithms for competitive reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming.* Athena Scientific, Belmont, MA, 1996.

Andres Campero, Roberta Raileanu, Heinrich Küttler, Joshua B Tenenbaum, Tim Rocktäschel, and Edward Grefenstette. Learning with amigo: Adversarially motivated intrinsic goals. *arXiv preprint arXiv:2006.12122*, 2020.

Christian Daniel, Herke Van Hoof, Jan Peters, and Gerhard Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104 (2):337–357, 2016.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

Thomas G Dietterich. State abstraction in maxq hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 994–1000, 2000.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *ICLR*, 2018.

F110. F1/10 Autonomous Racing Competition. http://f1tenth.org.

Chelsea Finn, Tianhe Yu, Justin Fu, Pieter Abbeel, and Sergey Levine. Generalizing skills with semi-supervised reinforcement learning. In *ICLR*, 2017.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Junling Hu and Michael P Wellman. Nash q-learning for general-sum stochastic games. *Journal of machine learning research*, 4(Nov):1039–1069, 2003.

Junling Hu, Michael P Wellman, et al. Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML*, volume 98, pages 242–250. Citeseer, 1998.

Radoslav Ivanov, Kishor Jothimurugan, Steve Hsu, Shaan Vaidya, Rajeev Alur, and Osbert Bastani. Compositional learning and verification of neural network controllers. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s):1–26, 2021.

Kishor Jothimurugan, Osbert Bastani, and Rajeev Alur. Abstract value iteration for hierarchical reinforcement learning. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 1162–1170. PMLR, 13–15 Apr 2021.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

George Konidaris and Andrew G Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in neural information processing systems*, pages 1015–1023, 2009.

George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. Constructing symbolic representations for high-level planning. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

Yen-Ling Kuo, Boris Katz, and Andrei Barbu. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of ltl formulas. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5604–5610, 2020.

Youngwoon Lee, Shao-Hua Sun, Sriram Somasundaram, Edward S Hu, and Joseph J Lim. Composing complex skills by learning transition policies. In *International Conference on Learning Representations*, 2018.

Youngwoon Lee, Joseph J Lim, Anima Anandkumar, and Yuke Zhu. Adversarial skill chaining for long-horizon robot manipulation via terminal state regularization. *arXiv preprint arXiv:2111.07999*, 2021.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

Michael L Littman. Friend-or-foe q-learning in general-sum games. In *ICML*, volume 1, pages 322–328, 2001.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6382–6393, 2017.

Marios C Machado, Marc G Bellemare, and Michael Bowling. A laplacian framework for option discovery in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2295–2304. JMLR. org, 2017.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3303–3313, 2018.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. In *ICLR*, 2019.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning*, pages 2661–2670. PMLR, 2017.

Stephen David Patek. *Stochastic and shortest path games: theory and algorithms*. PhD thesis, Massachusetts Institute of Technology, 1997.

Julien Perolat, Florian Strub, Bilal Piot, and Olivier Pietquin. Learning Nash Equilibrium for General-Sum Markov Games from Batch Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, pages 2817–2826. PMLR, 2017.

HL Prasad, Prashanth LA, and Shalabh Bhatnagar. Two-timescale algorithms for learning nash equilibria in general-sum stochastic games. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1371–1379, 2015.

Doina Precup, Richard S Sutton, and Satinder Singh. Theoretical results on reinforcement learning with temporally abstract options. In *European conference on machine learning*, pages 382–393. Springer, 1998.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

Sungryull Sohn, Junhyuk Oh, and Honglak Lee. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. *Advances in Neural Information Processing Systems*, 31, 2018.

Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer, 2002.

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

Georgios Theocharous and Leslie P Kaelbling. Approximate planning in pomdps with macro-actions. In *Advances in Neural Information Processing Systems*, pages 775–782, 2004.

Saket Tiwari and Philip S Thomas. Natural option critic. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5175–5182, 2019.

Pashootan Vaezipoor, Andrew C Li, Rodrigo A Toro Icarte, and Sheila A. Mcilraith. Ltl2action: Generalizing ltl instructions for multi-task rl. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10497–10508. PMLR, 18–24 Jul 2021.

Chen-Yu Wei, Yi-Te Hong, and Chi-Jen Lu. Online reinforcement learning in stochastic games. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4994–5004, 2017.

A Proofs

In this section, we prove the theorems presented in the paper through a series of lemmas. The proofs here are adaptations of proofs of similar results for zero-sum concurrent games with a single discount factor (Patek, 1997).

A.1 Definitions

We first introduce some notation and definitions. Recall that we defined $S_1 = \{(s,\sigma) \mid \sigma \notin F_\sigma\}$, $S_2 = \{(s,\sigma) \mid \sigma \in F_\sigma\}$ and $\bar{S} = S_1 \cup S_2$. $\mathcal{V} = \{V : S_1 \to \mathbb{R}\}$ denotes the set of value functions over S_1 and $\bar{\mathcal{V}} = \{V : \bar{S} \to \mathbb{R}\}$ denotes the set of value functions over $\bar{S}$. We use $\|\cdot\|$ to denote the ℓ_∞-norm. For any $V \in \mathcal{V}$, $(s,\sigma) \in S_2$ and any $\sigma' \in \Sigma$, define

$$[\![V]\!]_{\sigma'}(s,\sigma) = \sum_{s' \in S} T_\sigma(s' \mid s) V(s',\sigma').$$

Note that for any $(s,\sigma) \in S_2$, we have $[\![V]\!](s,\sigma) = \min_{\sigma' \in \Sigma} [\![V]\!]_{\sigma'}(s,\sigma)$. Similarly, for any $V \in \mathcal{V}$, $(s,\sigma) \in S_1$ and $a \in A$, let us define

$$\mathcal{F}_a(V)(s,\sigma) = \bar{R}((s,\sigma),a) + \gamma \cdot \sum_{s' \in S} P(s' \mid s,a)[\![V]\!](s',\sigma)$$

with $\mathcal{F}(V)(s,\sigma) = \max_{a \in A} \mathcal{F}_a(V)(s,\sigma)$. Given any policy $\pi_1 : \bar{S} \to A_1$ for agent 1 in $\mathcal{G}$, we define the resulting MDP $\mathcal{G}(\pi_1) = (\bar{S}, A_2, P_{\pi_1}, R_{\pi_1}, \gamma)$ with states $\bar{S}$ and actions $\mathcal{A}_2 = \Sigma$ as follows. The transition probability function is given by

$$P_{\pi_1}((s',\sigma') \mid (s,\sigma), a_2) = \begin{cases} \bar{P}((s',\sigma') \mid (s,\sigma), \pi_1(s,\sigma), a_2) & \text{if } (s,\sigma) \in S_1 \\ \sum_{s'' \in S} T_\sigma(s'' \mid s) \bar{P}((s',\sigma') \mid (s'',a_2), \pi_1(s'',a_2), a_2) & \text{if } (s,\sigma) \in S_2 \end{cases}$$

and the reward function is given by

$$R_{\pi_1}((s,\sigma),a_2) = \begin{cases} -\bar{R}((s,\sigma), \pi_1(s,\sigma)) & \text{if } (s,\sigma) \in S_1 \\ -\sum_{s' \in S} T_\sigma(s' \mid s) \bar{R}((s',a_2), \pi_1(s',a_2)) & \text{if } (s,\sigma) \in S_2. \end{cases}$$

Intuitively, the MDP $\mathcal{G}(\pi_1)$ merges every step of $\mathcal{G}$ in which a change of subtask occurs with the subsequent step in the environment, while using π_1 to choose actions for agent 1. For any $\bar{s} \in \bar{S}$, let $\mathcal{D}_{\bar{s}}^{\mathcal{G}(\pi_1)}(\pi_2)$ denote the distribution over infinite trajectories generated by π_2 starting at $\bar{s}$ in $\mathcal{G}(\pi_1)$. Then we define the value function for the MDP $\mathcal{G}(\pi_1)$ using

$$V_{\mathcal{G}(\pi_1)}^{\pi_2}(\bar{s}) = \mathbb{E}_{\rho \sim \mathcal{D}_{\bar{s}}^{\mathcal{G}(\pi_1)}(\pi_2)} \Big[\sum_{t=0}^{\infty} \gamma^t R_{\pi_1}(\bar{s}_t, a_t) \Big]$$

for all $\pi_2 : \bar{S} \to A_2$ and $\bar{s} \in \bar{S}$.

A.2 Necessary Lemmas

We need a few intermediate results in order to prove the main theorems. We begin by analyzing the operators $[\![\cdot]\!] : \mathcal{V} \to \bar{\mathcal{V}}$ and $\mathcal{F} : \mathcal{V} \to \mathcal{V}$ defined in Section 3.1.

Lemma A.1. *For any $V_1, V_2 \in \mathcal{V}$, we have $\|[\![V_1]\!] - [\![V_2]\!]\| = \|V_1 - V_2\|$.*

Proof. For any $(s,\sigma) \in S_1$, we have $|[\![V_1]\!](s,\sigma) - [\![V_2]\!](s,\sigma)| = |V_1(s,\sigma) - V_2(s,\sigma)|$. For any $(s,\sigma) \in S_2$ and $\sigma' \in \Sigma$ we have

$$\begin{aligned} |[\![V_1]\!]_{\sigma'}(s,\sigma) - [\![V_2]\!]_{\sigma'}(s,\sigma)| &= \|\sum_{s' \in S} T_\sigma(s' \mid s) V_1(s',\sigma') - \sum_{s' \in S} T_\sigma(s' \mid s) V_2(s',\sigma')\| \\ &\leq \sum_{s' \in S} T_\sigma(s' \mid s) |V_1(s',\sigma') - V_2(s',\sigma')| \\ &\leq \|V_1 - V_2\|. \end{aligned}$$

Now we have $|[\![V_1]\!](s,\sigma) - [\![V_2]\!](s,\sigma)| = |\min_{\sigma'} [\![V_1]\!]_{\sigma'}(s,\sigma) - \min_{\sigma'} [\![V_2]\!]_{\sigma'}(s,\sigma)| \leq \|V_1 - V_2\|$ which concludes the proof. □

Now we are ready to show that $\mathcal{F}$ is a contraction.

Lemma A.2. *$\mathcal{F} : \mathcal{V} \to \mathcal{V}$ is a contraction mapping w.r.t the norm $\|\cdot\|$.*

Proof. Let $V_1, V_2 \in \mathcal{V}$. Then for any $(s, \sigma) \in S_1$ and $a \in A$,

$$\begin{aligned}
|\mathcal{F}_a(V_1)(s,\sigma) - \mathcal{F}_a(V_2)(s,\sigma)| &= |\gamma \sum_{s' \in S} P(s' \mid s, a)[\![V_1]\!](s', \sigma) - \gamma \sum_{s' \in S} P(s' \mid s, a)[\![V_2]\!](s', \sigma)| \\
&\leq \gamma \sum_{s' \in S} P(s' \mid s, a)|[\![V_1]\!](s', \sigma) - [\![V_2]\!](s', \sigma)| \\
&\leq \gamma \|V_1 - V_2\|
\end{aligned}$$

where the last inequality followed from Lemma A.1. Therefore, for any $(s, \sigma) \in S_1$, we have $|\mathcal{F}(V_1)(s,\sigma) - \mathcal{F}(V_2)(s,\sigma)| = |\max_a \mathcal{F}_a(V_1)(s,\sigma) - \max_a \mathcal{F}_a(V_2)(s,\sigma)| \leq \gamma \|V_1 - V_2\|$ showing that $\mathcal{F}$ is a contraction. □

Now we connect the value function of the game $\mathcal{G}$ with that of the MDP $\mathcal{G}(\pi_1)$.

Lemma A.3. *For any $\pi_1 : \bar{S} \to A_1$, $\pi_2 : \bar{S} \to A_2$ and $\bar{s} \in \bar{S}$, $V^{\pi_1,\pi_2}(\bar{s}) = -V^{\pi_2}_{\mathcal{G}(\pi_1)}(\bar{s})$.*

Proof. Given an infinite trajectory $\bar{\rho} = \bar{s}_0 a_0^1 a_0^2 \bar{s}_1 a_1^1 a_1^2 \dots$ in $\mathcal{G}$ we define a corresponding trajectory $\bar{\rho}_2 = \bar{s}_{i_0} a_{i_0}^2 \bar{s}_{i_1} a_{i_1}^2$ in $\mathcal{G}(\pi_1)$ as a subsequence where $i_0 = 0$ and $i_{t+1} = i_t + 1$ if $\bar{s}_{i_t} \in S_1$ and $i_{t+1} = i_t + 2$ if $\bar{s}_{i_t} \in S_2$. Then for any $\bar{s} \in \bar{S}$ we have

$$\begin{aligned}
V^{\pi_1,\pi_2}(\bar{s}) &= \mathbb{E}_{\bar{\rho} \sim \mathcal{D}^{\mathcal{G}}_{\bar{s}}(\pi_1,\pi_2)}\Big[\sum_{t=0}^{\infty}\big(\prod_{k=0}^{t-1} \bar{\gamma}(\bar{s}_k)\big)\bar{R}(\bar{s}_t, a_t^1)\Big] \\
&\overset{(1)}{=} \mathbb{E}_{\bar{\rho} \sim \mathcal{D}^{\mathcal{G}}_{\bar{s}}(\pi_1,\pi_2)}\Big[\sum_{t=0}^{\infty} \gamma^t R_{\pi_1}(\bar{s}_{i_t}, a^2_{i_t})\Big] \\
&\overset{(2)}{=} -\mathbb{E}_{\rho \sim \mathcal{D}^{\mathcal{G}(\pi_1)}_{\bar{s}}(\pi_2)}\Big[\sum_{t=0}^{\infty} \gamma^t R_{\pi_1}(\bar{s}_t, a_t)\Big] \\
&= -V^{\pi_2}_{\mathcal{G}(\pi_1)}(\bar{s})
\end{aligned}$$

where (1) followed from the definitions of $\bar{\gamma}$ and R_{π_1} and the fact that $\bar{R}(\bar{s}_t, a_t^1) = 0$ if $\bar{s}_t \in S_2$, and (2) followed from the fact that sampling a trajectory ρ by first sampling $\bar{\rho}$ from $\mathcal{D}^{\mathcal{G}}_{\bar{s}}(\pi_1, \pi_2)$ and then constructing the subsequence $\bar{\rho}_2$ is the same as sampling an infinite trajectory ρ from $\mathcal{D}^{\mathcal{G}(\pi_1)}_{\bar{s}}(\pi_2)$[7]. □

Lemma A.2 shows that for any $V \in \mathcal{V}$ we have

$$\lim_{n \to \infty} \mathcal{F}^n(V) = V_{\lim}$$

where $V_{\lim} \in \mathcal{V}$ is the unique fixed point of $\mathcal{F}$. Now we define two policies π_1^* and π_2^* for agents 1 and 2 respectively, as follows. For $(s, \sigma) \in S_1$ we have

$$\pi_1^*(s,\sigma) \in \arg\max_{a \in A} \mathcal{F}_a(V_{\lim})(s,\sigma) \tag{4}$$

and for $(s, \sigma) \in S_2$, we have

$$\pi_2^*(s,\sigma) \in \arg\min_{\sigma'} [\![V_{\lim}]\!]_{\sigma'}(s,\sigma). \tag{5}$$

Note that the actions taken by π_1^* in S_2 and π_2^* in S_1 can be arbitrary since they do not affect the transitions of the game $\mathcal{G}$. Now we show that for any $\bar{s} \in \bar{S}$, π_1^* maximizes $V^{\pi_1,\pi_2^*}(\bar{s})$ and π_2^* minimizes $V^{\pi_1^*,\pi_2}(\bar{s})$.

Lemma A.4. *For any $\bar{s} \in \bar{S}$, $V^{\pi_1^*,\pi_2^*}(\bar{s}) = \min_{\pi_2} V^{\pi_1^*,\pi_2}(\bar{s}) = [\![V_{\lim}]\!](\bar{s})$.*

[7] This can be shown formally by analyzing the probabilities assigned by the two distributions on cylinder sets.

Proof. Let $\mathcal{G}(\pi_1^*) = (\bar{S}, \mathcal{A}_2, P_{\pi_1^*}, R_{\pi_1^*}, \gamma)$. For any $(s, \sigma) \in S_2$, we have

$$\begin{aligned}
[\![V_{\lim}]\!](s,\sigma) &= \min_{\sigma' \in \Sigma} \sum_{s' \in S} T_\sigma(s' \mid s)[\![V_{\lim}]\!](s', \sigma') \\
&\overset{(3)}{=} \min_{\sigma' \in \Sigma} \sum_{s' \in S} T_\sigma(s' \mid s)\Big(\bar{R}((s', \sigma'), a) + \gamma \cdot \sum_{s'' \in S} P(s' \mid s, a)[\![V_{\lim}]\!](s'', \sigma')\Big)\Big|_{a = \pi_1^*(s', \sigma')} \\
&\overset{(4)}{=} \min_{a_2 \in A_2} \Big(- R_{\pi_1^*}((s,\sigma), a_2) + \gamma \sum_{\bar{s} \in \bar{S}} P_{\pi_1^*}(\bar{s} \mid (s,\sigma), a_2)[\![V_{\lim}]\!](\bar{s})\Big)
\end{aligned}$$

where (3) followed from the definitions of $V_{\lim}$ and π_1^* and (4) followed from the definitions of $R_{\pi_1^*}$ and $P_{\pi_1^*}$. Since $-[\![V_{\lim}]\!]$ satisfies the Bellman equations for the MDP $\mathcal{G}(\pi_1^*)$, the optimal value function for $\mathcal{G}(\pi_1^*)$ is given by $V^*_{\mathcal{G}(\pi_1^*)} = -[\![V_{\lim}]\!]$. Now, from the definition of π_2^* we can conclude that π_2^* is an optimal policy for $\mathcal{G}(\pi_1^*)$. Therefore, Lemma A.3 implies that for any $\bar{s} \in \bar{S}$,

$$\min_{\pi_2} V^{\pi_1^*, \pi_2}(\bar{s}) = \min_{\pi_2} -V^{\pi_2}_{\mathcal{G}(\pi_1^*)}(\bar{s}) = -V^{\pi_2^*}_{\mathcal{G}(\pi_1^*)}(\bar{s}) = V^{\pi_1^*, \pi_2^*}(\bar{s}).$$

Hence, we have proved the desired result. □

The following lemma can be shown using a similar argument and the proof is omitted.

Lemma A.5. *For any $\bar{s} \in \bar{S}$, $V^{\pi_1^*, \pi_2^*}(\bar{s}) = \max_{\pi_1} V^{\pi_1, \pi_2^*}(\bar{s}) = [\![V_{\lim}]\!](\bar{s})$.*

The following lemma shows that it does not matter which agent picks its policy first.

Lemma A.6. *For any policies π_1^* and π_2^* satisfying Equations 4 and 5 respectively, for all $\bar{s} \in \bar{S}$,*

$$V^{\pi_1^*, \pi_2^*}(\bar{s}) = \min_{\pi_2} \max_{\pi_1} V^{\pi_1, \pi_2}(\bar{s}) = \max_{\pi_1} \min_{\pi_2} V^{\pi_1, \pi_2}(\bar{s}) = V^*(\bar{s}).$$

Proof. We have, for any $\bar{s} \in \bar{S}$,

$$\begin{aligned}
V^*(\bar{s}) &= \max_{\pi_1} \min_{\pi_2} V^{\pi_1, \pi_2}(\bar{s}) \\
&\geq \min_{\pi_2} V^{\pi_1^*, \pi_2}(\bar{s}) \\
&\overset{(1)}{=} V^{\pi_1^*, \pi_2^*}(\bar{s}) \\
&\overset{(2)}{=} \max_{\pi_1} V^{\pi_1, \pi_2^*}(\bar{s}) \\
&\geq \min_{\pi_2} \max_{\pi_1} V^{\pi_1, \pi_2}(\bar{s}) \\
&\geq \max_{\pi_1} \min_{\pi_2} V^{\pi_1, \pi_2}(\bar{s}) \\
&= V^*(\bar{s})
\end{aligned}$$

where the (1) followed from Lemma A.4 and (2) followed from Lemma A.5. □

A.3 Proof of Theorem 3.1

Let $\Pi(\pi_1) = \{\pi_\sigma \mid \sigma \in \Sigma\}$ be the set of subtask policies defined by π_1. Let $\tau = \sigma_0 \sigma_1 \ldots$ be a task. Then we define a history-dependent policy π_2^τ in $\mathcal{G}(\pi_1)$ which maintains an index i denoting the current subtask and picks σ_{i+1} upon

reaching any state in S_2 while simultaneously updating the index to $i+1$. Then we have

$$\begin{aligned}
J(\Pi(\pi_1)) &= \inf_{\tau\in\mathcal{T}} \mathbb{E}_{\rho\sim\mathcal{D}_\tau^\Pi}\Big[\sum_{t=0}^{\infty}\gamma^t R_{\tau[i_t]}(s_t,\pi_{\tau[i_t]}(s_t))\Big] \\
&\overset{(1)}{=} \inf_{\tau\in\mathcal{T}} \mathbb{E}_{\bar{s}\sim\bar{\eta}}\Big[\mathbb{E}_{\rho\sim\mathcal{D}_{\bar{s}}^{\mathcal{G}(\pi_1)}(\pi_2^\tau)}\big[-\sum_{t=0}^{\infty}\gamma^t R_{\pi_1}(\bar{s}_t,a_t)\big]\Big] \\
&= \inf_{\tau\in\mathcal{T}} \mathbb{E}_{\bar{s}\sim\bar{\eta}}[-V_{\mathcal{G}(\pi_1)}^{\pi_2^\tau}(\bar{s})] \\
&\geq \mathbb{E}_{\bar{s}\sim\bar{\eta}}[-\sup_{\tau\in\mathcal{T}} V_{\mathcal{G}(\pi_1)}^{\pi_2^\tau}(\bar{s})] \\
&\overset{(2)}{\geq} \mathbb{E}_{\bar{s}\sim\bar{\eta}}[-\max_{\pi_2} V_{\mathcal{G}(\pi_1)}^{\pi_2}(\bar{s})] \\
&\overset{(3)}{=} \mathbb{E}_{\bar{s}\sim\bar{\eta}}[\min_{\pi_2} V^{\pi_1,\pi_2}(\bar{s})] \\
&= J_{\mathcal{G}}(\pi_1)
\end{aligned}$$

where (1) followed from the definitions of π_2^τ and $\mathcal{G}(\pi_1)$, (2) followed from the fact that there is an optimal stationary policy maximizing $V_{\mathcal{G}(\pi_1)}^{\pi_2}(\bar{s})$ and (3) followed from Lemma A.4.

A.4 Proof of Theorem 3.2

Since $V^* = [\![V_{\lim}]\!]$, for all $(s,\sigma)\in\bar{S}$ we have $\pi_1^*(s,\sigma)\in\arg\max_{a\in A}\mathcal{F}_a(V_{\lim})(s,\sigma)$. Now for any π_2^* satisfying Equation 5, we can conclude from Lemma A.6 that, for any $\bar{s}\in\bar{S}$,

$$\begin{aligned}
J_{\mathcal{G}}(\pi_1^*) &= \mathbb{E}_{\bar{s}\sim\bar{\eta}}[\min_{\pi_2} V^{\pi_1^*,\pi_2}(\bar{s})] \\
&= \mathbb{E}_{\bar{s}\sim\bar{\eta}}[V^{\pi_1^*,\pi_2^*}(\bar{s})] \\
&= \mathbb{E}_{\bar{s}\sim\bar{\eta}}[\max_{\pi_1}\min_{\pi_2} V^{\pi_1,\pi_2}(\bar{s})] \\
&\geq \max_{\pi_1}\mathbb{E}_{\bar{s}\sim\bar{\eta}}[\min_{\pi_2} V^{\pi_1,\pi_2}(\bar{s})] \\
&= \max_{\pi_1} J_{\mathcal{G}}(\pi_1)
\end{aligned}$$

which shows that π_1^* maximizes $J_{\mathcal{G}}(\pi_1)$. □

A.5 Proof of Theorem 3.3

From Lemma A.2, we can conclude that $\mathcal{F}$ is a contraction over $\mathcal{V}$ w.r.t. the ℓ_∞-norm. Lemmas A.6 and A.4 gives us that $V^*\downarrow_{S_1} = V_{\lim}$. Now the definition of $V_{\lim}$ implies that $\lim_{n\to\infty}\mathcal{F}^n(V) = V^*\downarrow_{S_1}$ for all $V\in\mathcal{V}$. □

A.6 Proof of Theorems 3.4 and 3.5

This proof is similar to the proof of convergence of asynchronous value iteration for MDPs presented in the book by Bertsekas and Tsitsiklis (1996). It is easy to see that, for any $V\in\mathcal{V}$ and $\sigma\in\Sigma$, the operators $[\![\cdot]\!]$, $\mathcal{F}$, $\mathcal{F}_{\texttt{async}}$, and $\mathcal{F}_{\sigma,V}$ are monotonic. Recall that, for any $V\in\mathcal{V}$ and $\sigma\in\Sigma$, we defined the corresponding $V_\sigma\in\mathcal{V}_\sigma$ using $V_\sigma(s) = [\![V]\!](s,\sigma)$ if $s\in S$ and $V_\sigma(\bot)=0$. Also, we have $\mathcal{F}(V)(s,\sigma) = \mathcal{F}_{\sigma,V}(V_\sigma)(s) = \mathcal{F}_1(V)(s,\sigma)$ for all $(s,\sigma)\in S_1$.

Now let $V\in\mathcal{V}$ be a value function such that $\mathcal{F}(V)\leq V$. Then we have $\mathcal{F}_{\sigma,V}(V_\sigma)\leq V_\sigma$ for all $\sigma\in\Sigma$. Therefore, using monotonicity of $\mathcal{F}_{\sigma,V}$, we get that $\mathcal{F}_{\sigma,V}^m(V_\sigma)\leq\mathcal{F}_{\sigma,V}^{m-1}(V_\sigma)\leq V_\sigma$ for all $m>0$ which implies $\mathcal{F}_m(V)\leq\mathcal{F}_{m-1}(V)\leq V$. Hence, for any $(s,\sigma)\in S_1$,

$$\begin{aligned}
\mathcal{F}_{\texttt{async}}(V)(s,\sigma) = \mathcal{W}_\sigma(V)(s) &= \lim_{m\to\infty}\mathcal{F}_{\sigma,V}^m(V_\sigma)(s) \\
&\leq \mathcal{F}_{\sigma,V}(V_\sigma)(s) = \mathcal{F}(V)(s,\sigma).
\end{aligned}$$

Furthermore, letting $V^m = \mathcal{F}_m(V)$ we get that $[\![V^m]\!]\leq[\![V]\!]$ and hence $V_\sigma^m\leq\mathcal{F}_{\sigma,V}^m(V_\sigma)$ for all $\sigma\in\Sigma$. Also, for $(s,\sigma)\in S_1$, $\mathcal{F}(V^m)(s,\sigma) = \mathcal{F}_{\sigma,V^m}(V_\sigma^m)(s)\leq\mathcal{F}_{\sigma,V}(V_\sigma^m)(s)\leq\mathcal{F}_{\sigma,V}^{m+1}(V_\sigma)(s) = V^{m+1}(s,\sigma)$. Therefore, using

continuity of $\mathcal{F}$, we have $\mathcal{F}(\mathcal{F}_{\texttt{async}}(V)) = \mathcal{F}(\lim_{m\to\infty} V^m) = \lim_{m\to\infty} \mathcal{F}(V^m) \leq \lim_{m\to\infty} V^{m+1} = \mathcal{F}_{\texttt{async}}(V)$. Now we can show by induction on n that, for any $V \in \mathcal{V}$ with $\mathcal{F}(V) \leq V$ and $n \geq 1$, we have $\mathcal{F}(\mathcal{F}^n_{\texttt{async}}(V)) \leq \mathcal{F}^n_{\texttt{async}}(V)$ and

$$V_{\text{lim}} \leq \mathcal{F}^n_{\texttt{async}}(V) \leq \mathcal{F}^n(V).$$

Taking the limit as $n \to \infty$ gives us that $\lim_{n\to\infty} \mathcal{F}^n_{\texttt{async}}(V) = V_{\text{lim}}$ if $\mathcal{F}(V) \leq V$. Using a symmetric argument, we get that $\lim_{n\to\infty} \mathcal{F}^n_{\texttt{async}}(V) = V_{\text{lim}}$ if $\mathcal{F}(V) \geq V$.

Let $I \in \mathcal{V}$ be defined by $I(s,\sigma) = 1$ for all $(s,\sigma) \in S_1$. For a general $V \in \mathcal{V}$, we can find a $\delta > 0$ such that we have $V^- = V_{\text{lim}} - \delta I \leq V \leq V_{\text{lim}} + \delta I = V^+$ and $\mathcal{F}(V^-) \geq V^-$ and $\mathcal{F}(V^+) \leq V^+$. Therefore, using monotonicity of $\mathcal{F}_{\texttt{async}}$ we get

$$\mathcal{F}^n_{\texttt{async}}(V^-) \leq \mathcal{F}^n_{\texttt{async}}(V) \leq \mathcal{F}^n_{\texttt{async}}(V^+)$$

for all $n \geq 0$. Taking the limit as n tends to ∞ gives us the required result. Theorem 3.5 follows from a similar argument. □

A.7 Proof of Theorem 4.1

Given a function $Q : S_1 \times A \to \mathbb{R}$ we define a new function $\mathcal{H}(Q)$ using

$$\mathcal{H}(Q)(s,\sigma,a) = \bar{R}((s,\sigma),a) + \gamma \sum_{s' \in S} P(s' \mid s,a)[\![V_Q]\!](s',\sigma)$$

for all $(s,\sigma) \in S_1$ and $a \in A$. Then, Robust Option Q-learning is of the form

$$Q_{t+1}(s,\sigma,a) = (1-\alpha_t(s,\sigma,a))Q_t(s,\sigma,a) + \alpha_t(s,\sigma,a)\Big(H(Q_t)(s,\sigma,a) + w_t(s,\sigma,a)\Big)$$

where the noise factor is defined by

$$w_t(s,\sigma,a) = \gamma[\![V_{Q_t}]\!](\tilde{s},\sigma) - \gamma \sum_{s' \in S} P(s' \mid s,a)[\![V_Q]\!](s',\sigma)$$

with $\tilde{s} \sim P(\cdot \mid s,a)$ being the observed sample. Let $\mathcal{X}_t$ denote the measure space generated by the set of random vectors $\{Q_0, Q_1, \ldots, Q_t, w_0, \ldots, w_{t-1}, \alpha_0, \ldots, \alpha_t\}$. Then, for all $(s,\sigma) \in S_1$, $a \in A$ and $t \geq 0$, we have

$$\mathbb{E}[w_t(s,\sigma,a) \mid \mathcal{X}_t] = 0$$

and

$$\mathbb{E}[w_t^2(s,\sigma,a) \mid \mathcal{X}_t] \leq 4\gamma^2 \max_{s' \in S} \Big\{[\![V_{Q_t}]\!]^2(s',\sigma)\Big\} \leq 4\gamma^2 \max_{(s',\sigma') \in S_1, a' \in A} \Big\{Q_t^2(s',\sigma',a')\Big\}.$$

Furthermore, using Lemmas A.1 and A.2 and the definition of V_Q we can conclude that $\mathcal{H}$ is a contraction w.r.t the ℓ_∞-norm and Q^* is the unique fixed point of $\mathcal{H}$. Therefore, the random sequence of Q-functions $\{Q_t\}_{t\geq 0}$ satisfies all assumptions in Proposition 4.4 of Bertsekas and Tsitsiklis (1996) implying that $Q_t \to Q^*$ as $t \to \infty$ with probability 1. □

B Experimental Details

All experiments were run on a 48-core machine with 512GB of memory and 8 GPUs. In all approaches (ours and baselines) except for MADDPG, the policy consists of one fully-connected NN per subtask, each with two hidden layers. MADDPG consists of two policies, one for the agent and one for the adversary, each with two hidden layers. In the case of MADDPG, the subtask is encoded in the observation using a one-hot vector. All hyperparameters were computed by grid search over a small set of values.

Rooms environment. The hidden dimension used is 64 for all approaches except MADDPG for which we use 128 dimensional hidden layers. For DAGGER, NAIVE and AROSAC we run SAC with Adam optimizer (learning rate of $\alpha = 0.01$), entropy weight $\beta = 0.05$, Polyac rate 0.005 and batch size of 100. In each iteration of AROSAC and DAGGER, SAC is run for $N = 10000$ steps. Similarly, ROSAC is run with Adam optimizer (learning rates $\alpha_\psi = \alpha_\theta = 0.01$), entropy weight $\beta = 0.05$, Polyac rate 0.005 and batch size of 300. The MADDPG baseline uses a learning rate of 0.0003 and batch size of 256. PAIRED uses PPO with a learning rate of 0.02, batch size of 512, minibatch size of 128 and 4 epochs for each policy update. The adversary is trained using REINFORCE with a learning rate of 0.003.

F1/10th environment. The hidden dimension used is 128 for all approaches. For DAGGER, NAIVE and AROSAC we run SAC with Adam optimizer (learning rate of $\alpha = 0.001$), entropy weight $\beta = 0.03$, Polyac rate 0.005 and batch size of 128. In each iteration of AROSAC and DAGGER, SAC is run for $N = 10000$ steps. Similarly, ROSAC is run with Adam optimizer (learning rates $\alpha_\psi = \alpha_\theta = 0.001$), entropy weight $\beta = 0.03$, Polyac rate 0.005 and batch size of 5×128. The MADDPG baseline uses a learning rate of 0.0003 and batch size of 256. PAIRED uses PPO with a learning rate of 0.001, batch size of 512, minibatch size of 128 and 4 epochs for each policy update. The adversary is trained using REINFORCE with a learning rate of 0.003.