Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)

Abstraction-Guided Policy Recovery from Expert Demonstrations

Canmanie T. Ponnambalam, [1] **Frans A. Oliehoek,** [2] **Matthijs T. J. Spaan** [1]
[1] Department of Software Technology
[2] Department of Intelligent Systems
Delft University of Technology
Delft, The Netherlands
{c.t.ponnambalam, m.t.j.spaan, f.a.oliehoek}@tudelft.nl

Abstract

Behavior cloning is a method of automated decision-making that aims to extract meaningful information from expert demonstrations and reproduce the same behavior autonomously. It is unlikely that demonstrations will exhaustively cover the potential problem space, compromising the quality of automation when out-of-distribution states are encountered. Our approach RECO jointly learns both an imitation policy and recovery policy from expert data. The recovery policy steers the agent from unknown states back to the demonstrated states in the data set. While there is, per definition, no data available to learn the recovery policy, we exploit abstractions to generalize beyond the available data and simulate the recovery problem. When the most appropriate abstraction for the given data is unknown, our method selects the best recovery policy from a set generated by several candidate abstractions. In tabular domains, where we assume an agent must call to a human supervisor for help if it is in an unknown state, we show how RECO results in drastically fewer calls without compromising solution quality and with relatively few trajectories provided by an expert. We also introduce a continuous adaptation of our method and demonstrate the ability of RECO to recover an agent from states where its supervised learning-based imitation policy would otherwise fail.

Introduction

Reinforcement learning is an attractive approach to automating decision-making in increasingly complex systems. Unfortunately, learning methods often fall short of meeting the requirements of real-world problems. In several domains, particularly safety-critical applications, it is not practical to take random actions. However, learning algorithms get much of their power from this ability to explore. One way to assuage the limitations of such algorithms is to incorporate demonstrations by an expert and train agents to copy and deploy exemplar behavior with as little human supervision required as possible. In behavioral cloning (Bain and Sammut 1996), the agent is expected to directly replicate the demonstrated behavior rather than make insights about the underlying reasoning. We are interested in this case, where an expert has provided examples of correct behavior that we seek to automate without any exploration or further learning done in the real environment.

In a typical behavior cloning scenario, a predictive model can be used to supply expert actions trained on the example data provided by the expert (Ross and Bagnell 2010). As with any predictive model, the ability to generalize over data never seen before is not guaranteed. If the model is used for planning, the error will accumulate over every planning step. It may be infeasible for a human to define the correct behavior for every state even in relatively small problems. When presented with a state far from its expert data set, an agent can either follow its potentially very incorrect model or take random actions, both of which may be undesirable. Another option is to facilitate human intervention whenever the agent is lost; this is effective but compromises the overall autonomy of the system. All of these issues limit the applicability of behavior cloning.

Our method aims to remedy this by producing a robust behavior clone that can recover from parts of the state space not covered by its imitation policy. We apply state abstraction to expert trajectories, effectively multiplying the available data and using this new data set to generate a simulator for the recovery problem. Optimizing for expected reward in the simulator results in a recovery policy which returns the agent to states covered by its behavior policy. Instead of generalizing over the entire decision-making task, our agent focuses on the simpler problem of getting back to states covered by the demonstrations, thus minimizing the effect of model error. Even with a biased data set (containing only successful trajectories) and without making potentially dangerous conclusions about never-before-seen states, we can recover a lost agent back to a known policy.

RECO boosts the ability of behavior cloning agents and makes efficient use of offline data. Our method is aimed at problems where expert demonstrations have limited coverage of the problem space and where un-modelled effects may take an agent to a state outside of the given data. For example, an autonomous vehicle (AV) may encounter construction on the road and be forced to a state outside of its planned route. A RECO AV will adjust its goal to first aim to get back on track, using data from previous routes to do so, and then continue to execute the original plan.

In this paper, we first discuss related methods in the literature. Then, we formally define the policy recovery problem

and show how we use data from expert trajectories to specify and solve it. This includes a discussion on criteria for suitable abstraction selection and how to use several candidate abstractions to generate the best recovery policy. We assess the performance of RECO against several baselines in tabular and continuous problems, demonstrating its ability to make use of fewer trajectories while maintaining good performance in the environment. We conclude with suggestions for expanding this work.

Related Work

In behavior cloning or imitation learning, the offline data is produced by an expert who is assumed to follow an optimal policy. We focus on agents that, once deployed, cannot execute random actions (exploration) in the environment nor learn from their online interactions. All offline reinforcement learning techniques must deal with the issue of what to do when the agent encounters an out-of-distribution or anomalous state during execution. This issue of *distributional shift* is a well-studied problem in the literature (Fujimoto, Meger, and Precup 2019; Kumar et al. 2019, 2020). Several solutions have been proposed, such as reverting to a safe policy (Richter and Roy 2017), forcefully resetting the agent (Ainsworth, Barnes, and Srinivasa 2019), or requesting human intervention (Laskey et al. 2016; García and Fernández 2019). In our problem definition, we assume that a safe policy is not known outside of the expert trajectories provided, that resetting the agent is not possible and that human supervision in the true environment is very costly and therefore undesirable.

The notion of recovery has recently been applied to safe reinforcement learning, where offline data is used to identify unsafe zones and a learned recovery policy steers the agent back to safe states (Thananjeyan et al. 2020), though here the goal is to satisfy safety constraints during online learning rather than follow policies learned on expert demonstrations. Another online method by Eysenbach et al. (2018) involves an agent simultaneously learning to perform a task and learning to undo the actions it has done. In this way it learns policies that avoid irreversible states and aims to minimize the need for human intervention. There have been several imitation learning methods more related to our problem setting that aim to steer the agent towards the expert behavior (Hester et al. 2019; Siegel et al. 2020). In *soft Q imitation learning*, the learning agent is incentivized to take actions that lead back to states in given expert demonstrations (Reddy, Dragan, and Levine 2020). Similar to our method, they define a sparse reward function that provides reward for taking demonstrated actions from demonstrated states, training the agent to favor behavior close to the expert. They do not, however, isolate the planning problem of returning the agent to known states. Doing this in conjunction with state abstraction to facilitate simulation of the recovery problem from a fixed data set is our novel contribution to existing work.

Background on Markov Decision Processes

This section briefly introduces preliminary material that serves as a foundation for our method. A Markov decision process (MDP) is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ that describes a sequential decision making problem (Puterman 1994). The variables $\mathcal{S}$ and $\mathcal{A}$ denote the state and action space, $\mathcal{T}$ and $\mathcal{R}$ are transition and reward functions, and γ is a discounting factor ($0 \leq \gamma \leq 1$) that determines how far into the future to take into account. At each time step t, an agent observes the state of the environment $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$. Upon executing action a_t, the environment transitions to a new state $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$, according to a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ which maps state-action pairs to a distribution over next states. The agent receives a reward r_t according to the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$. The value of a state $V(s)$ is the sum of the expected discounted reward of the state, given by $V(s) = \mathbb{E} \sum_{t=0}^{\infty} \gamma^t r_t$. The solution to an MDP is a policy π that maps states to actions. The value of a policy $V(\pi)_{s_0=s_i}$ is the sum of expected discounted rewards when following policy π from initial state s_i. An optimal policy π^* is one that maximizes this value for every state. An MDP can be solved using Value Iteration or Policy Iteration; these are standard approaches that produce an optimal policy π^* (Puterman 1994).

Reinforcement learning concerns methods to find solutions for MDPs with unknown transition and reward functions. This typically involves executing actions in the environment and learning to both explore randomly and exploit the knowledge already learned regarding rewarding actions. For the uninitiated reader, we refer to Sutton and Barto (2018).

RECO: Abstraction-Guided Policy Recovery

Our aim is to automate the process of making decisions in an environment given examples of good behavior provided by an expert. These trajectories of expert decisions include states and corresponding expert actions, but do not cover the entire space of possible states and actions. Our behavior cloning agent must use the given data to compute a policy that will be deployed online without any additional data collection or computation.

We first focus on the discrete state-space (tabular) case, where the lack of function approximation renders a behavior cloning agent policy-less in states outside the data set. We assume that a human supervisor is available to provide an action when the agent has no policy to follow. The goal of our method in these problems is to reduce the number of calls to the human supervisor using only the data provided by the expert without a substantial loss in performance.

We later extend our method to continuous state-space problems and describe the modifications required to do so. With the help of function approximation in continuous domains, the agent can use its model to choose actions from a state even if the expert has not visited the same state; however, as demonstrated in our experiments, this can lead to sub-optimal behavior when the model is inaccurate. Here our goal is to recover the agent from states where the imita-

tion policy is uncertain due to lack of coverage by the expert data.

Problem Formalization

The problem considers optimizing a task represented as a Markov decision process, where the state and action space are known but the transition and reward function are unknown. Our data set of trajectories provided by an expert is in the form

$$\mathcal{D} = \{(s_1, a_1, s_2), (s_2, a_2, s_3), ..., (s_T, a_T, s_{T+1})\}$$

containing T (state, action, next state) tuples. The set of states found in $\mathcal{D}$ with actions provided by the expert is called $\mathcal{S}_\mathcal{D}$, where $\mathcal{S}_\mathcal{D} = \{s \in \mathcal{S} \mid (s, a, s') \in \mathcal{D}\}$ and $\mathcal{S}_\mathcal{D} \subseteq \mathcal{S}$. An imitation policy $\pi_\mathcal{D}(s)$ that aims to directly copy the expert behavior is defined for all $s \in \mathcal{S}_\mathcal{D}$.

An abstraction mapping function

$$\mu : \mathcal{S} \mapsto \bar{\mathcal{S}}$$

maps the state space $\mathcal{S}$ to a state space $\bar{\mathcal{S}}$, where $\bar{\mathcal{S}} \subseteq \mathcal{S}$. We assume the state space of our task is factored, thus the state space $\mathcal{S}$ is the space spanned by the domains of k state variables (Boutilier, Dearden, and Goldszmidt 2000):

$$\mathcal{S} = \{\mathcal{S}_0 \times ... \times \mathcal{S}_{k-1}\}.$$

The factored MDP description facilitates hand-design of state abstractions where $\mu(s)$ is a masking function that returns only the variables in s that are relevant to the recovery task: $\mu(s) = \bar{s}$. The inverse function $\mu'(\bar{s})$ returns the set of ground states that map to the abstract state $\bar{s}$ under the mapping function μ.

Our method RECO uses state abstraction to generate data for learning a recovery policy π_O that returns an agent to a state in the expert trajectories from states outside $\mathcal{S}_\mathcal{D}$.

Example In the classical taxi problem, a taxi agent in a grid world is tasked with picking up a passenger from a given location and dropping them off at their destination (Dietterich 2000). The variables in the factored state description indicate the x- and y-coordinate of the taxi and a reference to the locations of the passenger and destination, i.e., $s = (x, y, passenger, destination)$. The actions available are $\mathcal{A} = \{south, north, east, west, pickup, drop\text{-}off\}$. Figures 1(a) and (b) indicate given expert behavior for different locations of the taxi in the grid, when the passenger and destination locations are as shown. Suppose our agent finds itself at location o, pictured in red, with the passenger and destination shown in black. For this particular state of the passenger and destination, the expert has not provided an example, only the path given by the black arrows in Figure 1(a). However, another trajectory, shown as the blue arrows in Figure 1(b), for a different configuration of passenger and destination does contain this taxi location. By ignoring irrelevant information (the passenger and destination), our agent can reason about actions that return it to a known state. A RECO agent leverages the actions in the blue path, taking them from its position at o and getting back to the known policy (black path) which it then executes, depicted in Figure 1(c). It does this by applying

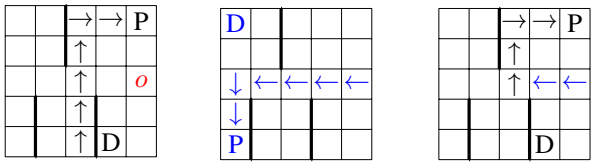

Figure 1: Examples of expert trajectories are shown in (a) and (b), given for two problem instances of the classical taxi domain where the grid location is the location of the taxi, P indicates the passenger location, and D the destination of the passenger. In (c), we see the RECO policy for initial state o learned from the data in (a) and (b).

what it knows in the abstract state space (x- and y-position) to get back to a known state. In this case, $\mu = \{x, y\}$.

We first describe the mechanics of RECO when a single state abstraction is provided. In the section titled Abstraction Selection, we lay out the conditions for suitable abstractions and adapt RECO to the case where several candidate abstractions have been provided and can be used in parallel.

RECO in Tabular Domains

In tabular domains, the imitation policy $\pi_\mathcal{D}(s)$ samples an action a according to a weighted distribution, weighted by the number of times a was executed by the expert from state s. This generalizes to the case where the expert policy may be stochastic. We consider the environment to be deterministic to simplify notation, and describe the extension to stochastic domains later in this section. The given abstraction mapping function μ is applied to each state entry in the data set to get an abstracted data set

$$\bar{\mathcal{D}} = \{(\bar{s}_1, a_1, \bar{s}_2), (\bar{s}_2, a_2, \bar{s}_3), ..., (\bar{s}_T, a_T, \bar{s}_{T+1})\}.$$

The recovery problem of getting back to a state in the data set from a state outside of $\mathcal{S}_\mathcal{D}$ is modelled as an MDP.

Recovery MDP The recovery MDP $\mathcal{M}_O$ is defined over the entire state space $\mathcal{S}$ augmented with an additional sink state z. The action space is copied from the original MDP. Any transition from a state in the expert trajectories transitions to the sink state, as the goal of recovery has been reached. Abstract transitions are projected onto all the ground states that map to the abstract state; this means we copy the abstract transitions for all possible combinations of the un-abstracted variables. Transitions that are not represented in the abstracted data set also transition to the sink state, as we have no data available for these. The reward for taking action a_t from state s_t is 1 if s_t is present in $\mathcal{S}_\mathcal{D}$, otherwise 0.

Definition 1 (Recovery MDP $\mathcal{M}_O$ for tabular domains)**.** Given a deterministic MDP, a data set $\mathcal{D}$ generated by an expert policy and an abstraction function μ, recovery MDP

$\mathcal{M}_O = (\mathcal{S}_O, \mathcal{A}_O, \mathcal{T}_O, \mathcal{R}_O, \gamma_O)$ is specified as:

$$\mathcal{S}_O = \mathcal{S} \cup \{z\},$$
$$\mathcal{A}_O = \mathcal{A},$$
$$\mathcal{R}_O(s_t, a_t, s_{t+1}) = \begin{cases} 1, \text{if } s_t \in \mathcal{S}_\mathcal{D}, \\ 0, \text{otherwise.} \end{cases}$$
$$\mathcal{T}_O(s_t, a_t, s_{t+1}) = \begin{cases} 1, \text{if } s_t \in \mathcal{S}_\mathcal{D} \wedge s_{t+1} = z, \\ 1, \text{else if } (\bar{s}_t, a_t, \bar{s}_{t+1}) \in \bar{\mathcal{D}}, \\ 1, \text{else if } (\bar{s}_t, a_t) \notin \bar{\mathcal{D}} \wedge s_{t+1} = z, \\ 1, \text{else if } s_t = s_{t+1} = z, \\ 0, \text{otherwise.} \end{cases}$$
$$\gamma_O = (0, 1).$$

When the environment is stochastic, the transition function of $\mathcal{M}_O$ is defined by its maximum likelihood estimate. This means that instead of assuming abstract transitions happen with probability 1, we replace the 1 on the second line of the transition function in Definition 1 with:

$$\frac{\sum_{s_t^* \in \mu'(\bar{s}_t)} N(s_t^*, a_t, s_{t+1})}{\sum_{s_t^* \in \mu'(\bar{s}_t)} N(s_t^*, a_t)},$$

where recall that $\mu'(\bar{s})$ returns all the ground states that map to the same abstract state as s.

Policy Recovery using the Recovery MDP The fully-defined recovery MDP can be solved using an off-the-shelf method such as Value Iteration, producing the recovery policy π_O which maximizes the expected discounted reward in the recovery problem. The recovery policy takes actions that correspond to finding the shortest path back to a state in the expert trajectories. When acting online, if a RECO agent is presented with a state outside of its expert trajectories, it has the option of executing the recovery policy.

We note that the success of recovery depends on the data we have, and we may not have the ability to control how we collect data or how much data we collect. Fortunately, there is a simple way to check for which states our recovery policy is able to recover the agent. Our definition of a *recoverable* policy is adapted from the *proper* policy definition applied to stochastic shortest path problems (SSPs) (Mausam and Kolobov 2012). Our recovery problem is similar to an SSP except instead of no discounting and a negative reward for every action taken, we have an infinite-horizon discounted-reward problem with a discount factor $0 < \gamma_O < 1$, a reward of 1 for taking any action from a goal state, and a reward of 0 everywhere else.

Definition 2 (Recoverable policy)**.** A policy π applied to $\mathcal{M}_O$ is *recoverable* over some state space $\mathcal{S}_\mathcal{P}$ if, when following π from any state $s \in \mathcal{S}_\mathcal{P}$, there is a positive probability that some goal state $s_d \in \mathcal{S}_\mathcal{D}$ will be reached in a finite number of steps.

With this satisfied, every state in $\mathcal{S}_\mathcal{P}$ has some path under policy π_O that leads to a state in the expert trajectories. Another difference between the recovery problem and SSPs is that we are not guaranteed that our recovery policy can recover from any state, thus we must assess whether a policy π_O is *recoverable*. Without access to the reward function, we can make no conclusions about the value of our policy in the real environment. However, we do know that the only way for a policy to have a non-zero value in the recovery MDP is if it reaches a state in the expert trajectories. We then know that a policy is *recoverable* over $\mathcal{S}_\mathcal{P}$ if it has a positive value for all states in S_P, i.e.,

$$V(\pi)_{s_0=s} > 0, \qquad \forall s \in \mathcal{S}_\mathcal{P}.$$

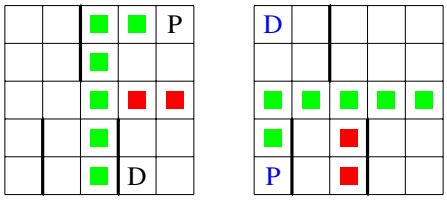

Figure 2: For the same two instances of the passenger and destination locations (P and D respectively) and given the expert trajectories shown in Figures 1(a) and 1(b), green indicates the states in $\mathcal{S}_\mathcal{D}$ over which the imitation policy is defined and red indicates the states in $\mathcal{S}_\mathcal{P}$ for which we have a *recoverable* policy.

In the next section, we discuss the conditions for abstraction under which we are guaranteed that a recoverable policy obtained by solving the recovery problem will recover an agent in the ground MDP. By checking the value of a policy in a particular state in the recovery MDP, we can eliminate non-recoverable policies which we know do not reach the destination from that state. For states without a recoverable policy, our agent executes an emergency action that requests an action from an expert. Combined, the overall RECO policy is defined as follows.

Definition 3 (RECO policy in tabular environments)**.** A tabular RECO agent given an expert data set $\mathcal{D}$ and a single recovery abstraction μ follows the aggregated policy from state s_t:

$$\pi_{RECO}(s_t, \mathcal{D}, \mu) = \begin{cases} \pi_\mathcal{D}(s_t), \text{if } s_t \in \mathcal{S}_\mathcal{D}, \\ \pi_O(s_t), \text{if } s_t \in \mathcal{S}_\mathcal{P}, \\ request, \text{otherwise.} \end{cases}$$

We highlight the state spaces $\mathcal{S}_\mathcal{D}$ and $\mathcal{S}_\mathcal{P}$ in our taxi example in Figure 2.

Abstraction Selection

Our method requires an abstraction in the form of a subset of state variables relevant to recovery, but not necessarily relevant to finding the optimal policy in the true environment. In this section, we detail the conditions on the abstraction that ensure that a *recoverable* policy will recover an agent in the ground MDP. We also discuss the emergent trade-offs in choosing an appropriate abstraction as well as how to manage these trade-offs.

Two important concepts to introduce are recovery-relevant actions and model-consistency in the recovery

MDP. In the recovery MDP, we project abstract transitions back onto ground states, transitioning only the abstracted variables according to the data set. We define a recovery-relevant action as one that, in the recovery MDP, results in a change of state with a non-zero probability (excluding transitions to or from the sink state). This leads to the following definition:

Definition 4 (Recovery-relevant actions)**.** The set of recovery-relevant actions Δ for $\mathcal{M}_\mathcal{O}$ is defined as:

$$\Delta_{\mathcal{M}_\mathcal{O}} = \{a \in \mathcal{A}_\mathcal{O} | \exists_{s^* \in \mathcal{S}_\mathcal{O}} \mathcal{T}_\mathcal{O}(s, a, s^*) > 0 \wedge s \neq s^* \neq z\}.$$

In other words, any action that does not have any effect on the abstract state is irrelevant to recovery. The set of recovery-relevant actions is a function of both an abstraction mapping and the expert trajectories to which it is applied. In the taxi problem, if we select abstraction $\mu = \{x, y\}$, then the pickup and drop-off actions are recovery-irrelevant as they have no effect on the x- or y-coordinate of the agent.

Model-consistency refers to whether the behavior in the abstract space is obeyed for the ground states which map to an abstract state.

Definition 5 (Model-consistent recovery MDP)**.** A recovery MDP $\mathcal{M}_O$ is model-consistent with the ground MDP $\mathcal{M}$ if the transition function agrees for all *relevant actions* in all ground states that map to the same abstract state (excluding transitions to the sink state z):

$$\mathcal{T}_O(s_t, a_t, s_{t+1}) = \mathcal{T}(s_t^*, a_t, \mu(s_{t+1})), \text{ where } s_{t+1} \neq z, \forall a_t \in \Delta_{\mathcal{M}_O} \forall s_t^* \in \mu'(\mu(s_t)).$$

Our method assumes that the transition model in the recovery MDP is model-consistent with the transitions in the ground MDP for relevant actions. With this criteria met, we are guaranteed that a *recoverable* policy will return the agent in the ground MDP to a state in our expert trajectories. In the taxi example, the model-consistency criteria is not met by any abstraction that includes $\{x\}$ without $\{y\}$ because of the walls in the environment. We assume that expert knowledge of the domain can produce abstractions that meet this criteria, though they can also be determined empirically with some confidence depending on the amount and coverage of expert data given. It is also important to note that if the domain of any state variable is increased (exponentially increasing the size of the state space), the abstraction definition will remain fixed and no further burden is put on the designer.

It is possible that several abstractions fit the criteria for model-consistency. A finer abstraction (i.e., smaller abstract state space, more compression) might require less coverage of the state space in the expert trajectories but then have a lower recovery potential. We depict this graphically in Figure 3, where it is clear that a $\{y\}$ abstraction can only recover to states with the same x-position. A coarser abstraction takes into account more information when determining the closest state, thus potentially recovering to more relevant states, however finding recoverable policies requires more coverage of the state space in the expert data. While several

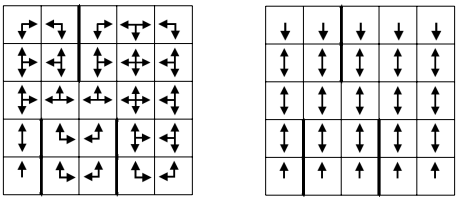

Figure 3: Recovery-relevant actions from the expert trajectories for the $\{x, y\}$ (left) and $\{y\}$ (right) abstractions projected back onto the taxi domain for all possible locations of passenger and destination.

abstractions may fit our criteria for suitability, their success is highly dependent on the actual data available. This brings us to the question: given a fixed set of expert trajectories, how can we determine the best abstraction for recovery?

Dynamic Policy Recovery with Parallel Abstractions As the recovery MDP is defined and solved offline, there is an opportunity to generate and solve the recovery problem for several candidate abstractions. We can use this set of policies, which we call Π, to choose the most suitable for a state we encounter online. Upon deployment, presented with a state s outside of our trajectories, we choose a recovery policy from the set of policies which are *recoverable* in s. This dynamic policy recovery is presented in Algorithm 1. We assume that the recoverable policy produced by the coarsest abstraction is the best choice as it incorporates the most state information in planning its path, and thus lies nearer to the expert behavior. Therefore, the function $RecoveryPolicySelection()$ on line 15 returns the recoverable policy produced by the abstraction with the largest abstract state space. In our empirical evaluations, we show how taking a dynamic abstraction approach can improve performance of a RECO agent by effectively trading off abstraction size with the coverage of the data set.

RECO in Continuous Domains

We expand on the introduction of RECO for tabular problems with an adaptation for continuous-state discrete-action problems. With the use of function approximation and the now continuous state-space, we lose the assumptions and formal definitions presented in tabular RECO. This section is meant to touch on the changes necessary in adapting tabular RECO to continuous problems as well as their current limitations. We present continuous RECO assuming a single abstraction has been given, though in principle dynamic switching between abstractions could be applied.

Model Definition The imitation policy is learned with a supervised approach, predicting expert actions for a given state, trained on the data in $\mathcal{D}$. We again call this policy $\pi_\mathcal{D}$. As in the tabular version, a given abstraction function $\mu(s)$ is applied to the data set resulting in abstracted trajectories $\bar{\mathcal{D}}$.

We gauge how close an encountered state is to our expert trajectories with a distance measure $d(s, \mathcal{D})$. For this measure, we use a simple weighted L1 norm calculation where the weight vector is the size of the number of state vari-

Algorithm 1: Dynamic Tabular RECO

Result: Policy π_{RECO}
Given a set of expert trajectories $\mathcal{D}$;
Given a set of suitable candidate abstractions μ_C;
Given number of test steps N;
Initialize empty set of recovery policies $\Pi = \emptyset$;
foreach $\mu \in \mu_C$ **do**
  $\pi_O = GetRecoveryPolicy(\mu, \mathcal{D})$;
  $\Pi = \Pi \cup \{\pi_O\}$;
end
$t = 0$;
while $t < N$ **do**
  Receive state s_t;
  if $s_t \in S_{\mathcal{D}}$ **then**
    $a_t = ImitationPolicy(\mathcal{D}, s_t)$;
  else if $RecoverablePolicies(\Pi, s_t) \neq \emptyset$ **then**
    $\pi_R = RecoveryPolicySelection(\Pi, s_t)$;
    $a_t = \pi_R(s_t)$;
  else
    $a_t = RequestExpert(s_t)$;
  end
  Execute action a_t;
  $t = t + 1$;
end

ables. This returns the norm between state s and the closest state in $\mathcal{D}$ (i.e., the smallest weighted norm). A second distance measure $d_a((s, a), \mathcal{D})$ returns the distance between a (state, action) pair and the trajectory. For d_a, we calculate the norm considering only states in the trajectory from which action a was taken. Many other distance measures are possible and can replace the one we use (Taylor, Kulis, and Sha 2011; Garcia and Fernandez 2012; Bellemare et al. 2016). In general, the issue of determining when a state is out-of-distribution is an orthogonal problem to the one presented here.

The transition and reward functions (see Definition 6) of our continuous recovery problem can no longer be represented in tabular form. We want to capture the transitions in the abstract space, thus we train a supervised learning model on $\bar{\mathcal{D}}$ to predict abstract next states from abstract states and actions. We call this predictor $\phi_{\mathcal{T}}$, where $\phi_{\mathcal{T}}(\bar{s}_t, a_t) = \bar{s}_{t+1}$. In the tabular setting, we copied the abstract transition for all possible combinations of the un-abstracted variables to effectively multiply our data set, something which is no longer possible in the continuous case. For simplicity, we instead make an assumption that our abstract state will transition according to $\phi_{\mathcal{T}}$ and the other variables remain unchanged. Our transition function ${\mathcal{T}'}_O$ thus takes a full state s_t and action a_t and transitions only the abstract variables, resulting in s_{t+1}.

The reward function takes a (s_t, a_t, s_{t+1}) tuple and gives a reward of 1 for entering a state only if the distance between it and the closest state in the expert trajectories is lower than a threshold $\zeta_{\mathcal{D}}$. The reward is 0 if $(\bar{s}_t, a_t)$ is within a second distance threshold ζ_μ of the closest abstract state in the abstract data set and -1 if neither is true. An episode ends if the agent reaches such a state or gets a reward of -1.

This leads to the following definition.

Definition 6 (Recovery MDP ${\mathcal{M}'}_O$ for continuous domains)**.** Recovery MDP ${\mathcal{M}'}_O = ({\mathcal{S}'}_O, {\mathcal{A}'}_O, {\mathcal{T}'}_O, {\mathcal{R}'}_O, \gamma'_O)$ is defined as:

$$
\begin{aligned}
{\mathcal{S}'}_O &= \mathcal{S}, \\
{\mathcal{A}'}_O &= \mathcal{A}, \\
{\mathcal{T}'}_O(s_t, a_t, \phi_{\mathcal{T}}(\bar{s}_t, a_t)) &= s_{t+1}, \\
{\mathcal{R}'}_O(s_t, a_t, s_{t+1}) &= \begin{cases} 1, \text{if } d(s_{t+1}, \mathcal{D}) < \zeta_{\mathcal{D}}, \\ 0, \text{else if } d_a((\bar{s}_t, a_t), \bar{\mathcal{D}}) < \zeta_\mu, \\ -1, \text{otherwise}, \end{cases} \\
\gamma'_O &= (0, 1).
\end{aligned}
$$

With the transition and reward function defined, we can simulate the recovery environment. We initialize a random starting state by sampling a state from the expert trajectories and replacing the abstracted variables (those in mapping μ) with a uniformly random value within their bounds. Given an action, the environment transitions according to the transition function, producing a next state and reward from the reward function. In practice, we find learning can be sped up if an episode ends when a maximum number of steps is reached with a large reward penalty. The simulator can be used to solve the recovery problem with a standard continuous state-space, discrete action-space reinforcement learning method, obtaining π_O. For the continuous experiments presented in the next section, we used a variant of Proximal Policy Optimization (Schulman et al. 2017).

In the continuous case, we can generalize our models to states outside our trajectory and avoid the need for a human supervisor, activating the recovery policy when we are far from our expert trajectories. As in the tabular case, we use our recovery policy only if it has a positive value in our simulated MDP, meaning that it found a path from the particular state we are in to a state appropriately close to the trajectories. This also stops us from trusting our recovery policy if we did not run the simulator long enough to solve for a particular state. The space which spans the states over which our recovery policy is *recoverable* is again called $\mathcal{S}_{\mathcal{P}}$. The combined RECO policy is defined below.

Definition 7 (RECO policy in continuous environments)**.** A continuous RECO agent given dataset $\mathcal{D}$, abstraction mapping function μ, and distance thresholds ζ_D and ζ_μ takes an action from s_t according to the following policy:

$$
\pi_{RECO}(s_t, \mu, \mathcal{D}, \zeta_D, \zeta_\mu) = \begin{cases} \pi_{\mathcal{D}}(s_t), \text{if } d(s_t, \mathcal{D}) < \zeta_D, \\ \pi_O(s_t), \text{if } s_t \in \mathcal{S}_{\mathcal{P}}, \\ \pi_{\mathcal{D}}(s_t), \text{otherwise}. \end{cases}
$$

We use the recovery policy to safely guide our agent closer to the given expert trajectories when the agent is in a state far from the given data and we are confident that our recovery policy can do so.

Empirical Evaluation

We conducted experiments in the classic taxi problem (Dietterich 2000) and a continuous problem where the agent is tasked with navigating to a given location in an environment with obstacles. In each trial, the learning for each agent was done offline on the same set of given expert trajectories (a trajectory is one episode). Agents were then deployed and evaluated on their performance online in the test environment, initialized to a random starting state every episode ensuring each agent is given the same starting state.

Tabular Experiments

The taxi problem consists of a 5x5 grid and has 4 possible passenger locations (plus 1 for when the passenger is in the taxi) and 4 destination locations, leading to a total of 500 states. We generated expert trajectories by initializing the agent in a random state and following a trained and converged Q-learning greedy policy until episode termination.

We demonstrate the performance of four RECO agents: three use a single abstraction and the last is the dynamic approach which chooses the best abstraction for every state according to Algorithm 1. All RECO agents use a discount factor of 0.95 in solving the recovery MDP. The RECO agents are then compared to three baselines on their performance in 100 trials of 100 test episodes. The *Imitation* agent follows the imitation policy and calls for help in any state outside the expert trajectories given. The *Nearest Neighbor* agent uses a crude nearest neighbor approach to choose actions when outside the expert trajectories, sampling an action from the nearest Cartesian state in the given data set. In order to maximize the performance of this simple model, we instead consider the nearest abstract Cartesian state according to the abstraction function that gave the best performance in experiments (in this case: $\{x, y\}$). The *Factored Batch RL* approach is a method that has been used as a baseline in other offline RL work (Simão and Spaan 2019). It uses complete knowledge of the transition dynamics in the form of a dynamic Bayesian network and calculates the parameters of the model with the offline data. This baseline acts as the upper bound of offline learning performance in a factored MDP.

Results Figure 4 presents the results from the tabular experiment. The top plot shows the percent of episodes in which at least one request for an expert action was made and the bottom plot displays the average episode reward over 100 test episodes. Note that the *Expert*, *Nearest Neighbor* and *Factored Batch RL* agents never call the expert. All RECO agents show substantially fewer calls to the expert than the *Imitation* agent, with the *Imitation* agent calling for an expert in 60% of test episodes even after 150 trajectories are provided. In comparing the RECO agents, the choice of abstraction has a substantial effect on the performance. As expected, there is a higher reward loss with the smallest abstraction when given few expert trajectories; this is due to the high loss of information, where recovery is planned using actions less related to the expert behavior. The largest abstraction slightly dominates the reward performance when few trajectories are provided. The biggest difference between RECO agents is seen in the number of calls to a supervisor. Dynamic RECO effectively trades off the number of calls to the expert with performance, demonstrating the fewest calls (of the RECO agents) while maintaining high rewards. We also see that Dynamic RECO is on par with our upper baseline *Factored Batch RL*, converging to near-expert performance around 80 trajectories with almost no calls to the expert. If we allow a threshold of around 10% calls to an expert, RECO reaches near-optimal performance with almost half the amount of data required as the upper baseline. The *Nearest Neighbor* approach, despite attempts to tune it to be as good as possible, performs very poorly.

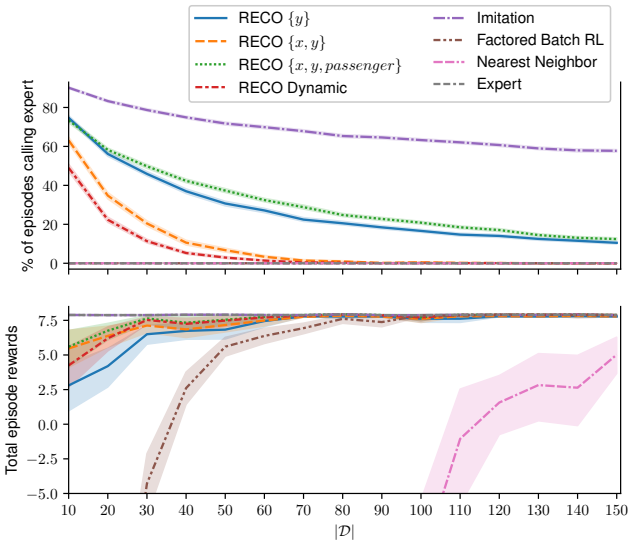

Figure 4: Performance on 100 test episodes of the 5x5 taxi problem averaged over 200 trials of random initial states. 98% confidence interval indicated by shaded regions.

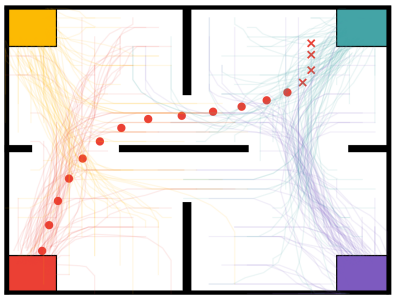

Figure 5: Actions taken by the RECO agent in one test episode of the continuous problem. Translucent lines depict the 100 expert trajectories provided to the agent, where color denotes the goal location. Solid points are actions taken online; X marks the recovery policy and circles indicate the imitation policy.

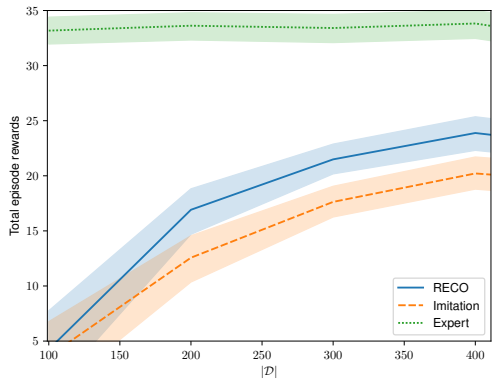

Figure 6: Performance on 100 test episodes of the continuous problem averaged over 200 trials of randomly initialized episodes. 98% confidence interval indicated by shaded regions.

Continuous Experiment

In the continuous problem, an agent is tasked with navigating to one of 4 possible goal locations in an environment that includes walls. There is a -1 reward every time step to encourage the agent to solve the task in as few time steps as possible and a large penalty for hitting a wall or for reaching a maximum number of time steps, all of which end the episode. The agent receives a +100 reward when it reaches the goal (also ending the episode). The state is described by variables $\{x, y, x\text{-}displacement, y\text{-}displacement, goal\ x, goal\ y\}$ and the agent must choose an action from $\{north, south, west, east\}$. Each action results in a discrete magnitude of acceleration propelling the agent in the corresponding direction, with acceleration decaying each time step. We purposefully force the initial position to be always on the same vertical half (in terms of x-position) as the goal to ensure that parts of the state space are left un-demonstrated by the expert.

The "expert" is a reinforcement learning agent trained with Proximal Policy Optimization (PPO) on 3e6 time steps of this task. In order to preserve reproducibility, all executions of PPO use the standard PPO2 implementation of the Stable Baselines library (Hill et al. 2018). We use a weighting for the distance measures of $(1, 1, 0.1, 0.1, 10, 10)$, meaning that the goal location is weighed heavily and the displacement much less so when considering distance between states. The distance thresholds are set to $\zeta_D = 10$ and $\zeta_\mu = 6$ and the discount factor is 0.99. The abstraction we use is $\{x, y, x\text{-}displacement, y\text{-}displacement\}$. Both the imitation agent and the transition functions are trained neural networks with a single hidden layer of 100 neurons. We ran PPO on the simulated recovery domain for 3e4 time steps to obtain the recovery policy. The performance of RECO is compared against the *Imitation* agent, which follows the imitation policy for any state encountered.

Results Figure 5 is a visualization of a single test episode in the continuous environment, showing how RECO guides an agent back to known states using actions that are similar to those taken before in an abstract state. The imitation policy is then executed and the task solved. The results of our experiments are plotted in Figure 6. RECO is superior to the *Imitation* agent once it has enough data to leverage, indicating the success of the recovery policy in retrieving the agent from states where its imitation model would be incorrect. Both agents converge to sub-optimal performance due to the purposeful limiting of the state space covered by the expert demonstrations. When we allow the expert to initialize trajectories from any random state, the imitation policy converges to near-expert performance after around 1000 given trajectories and the RECO policy does not have a noticeable advantage. This emphasizes that RECO is especially suited to improve behavior cloning in problems where the expert demonstrations do not have good coverage of the state space.

Conclusion and Future Work

Safety-critical or otherwise costly real-world applications are unlikely to allow automated decision-making algorithms to freely take random actions online. Encoding existing knowledge can make automated methods more applicable in such scenarios. RECO does this by applying state abstraction to expert trajectories to learn a recovery policy that steers an agent from states outside of the given data back to states where it can confidently execute its imitation policy. We show in experiments that RECO can drastically improve the performance of a behavior cloning agent in tabular domains, requiring far fewer calls to a human supervisor than a naïve behavior clone and achieving near-optimal performance on par with the baseline. We also demonstrate how a RECO agent can be implemented in continuous domains to recover from states where a supervised-learned-based imitation policy would otherwise fail. Our method is especially equipped for scenarios where there are un-modelled effects in the environment that force the agent into a state where its imitation policy is poorly trained or defined. In future work, we will investigate more efficient distance measures in continuous problems, including measures that take into account the uncertainty in the imitation model. We also hope that our novel approach will inspire tangential research into more generalized problem descriptions.

Acknowledgments

We would like to thank Qisong Yang and Thiago Dias Simão for their valuable feedback. This project is part of the research program Physical Sciences TOP-2 with project number 612.001.602, financed by the Dutch Research Council (NWO). This work has also received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 758824 —INFLUENCE).

References

Ainsworth, S.; Barnes, M.; and Srinivasa, S. 2019. Mo'States Mo'Problems: Emergency Stop Mechanisms from Observation. In *Advances in Neural Information Processing Systems*, volume 32, 15182–15192. Curran Associates, Inc.

Bain, M.; and Sammut, C. 1996. A Framework for Behavioural Cloning. *Machine Intelligence 15* 103–129.

Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying Count-Based Exploration and Intrinsic Motivation. In *Advances in Neural Information Processing Systems*, volume 29, 1471–1479. Curran Associates, Inc.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic Dynamic Programming with Factored Representations. *Artificial Intelligence* 121(1): 49 – 107.

Dietterich, T. G. 2000. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research* 13(1): 227–303.

Eysenbach, B.; Gu, S.; Ibarz, J.; and Levine, S. 2018. Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning. In *International Conference on Learning Representations*.

Fujimoto, S.; Meger, D.; and Precup, D. 2019. Off-Policy Deep Reinforcement Learning without Exploration. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 2052–2062. Long Beach, California, USA: PMLR.

Garcia, J.; and Fernandez, F. 2012. Safe Exploration of State and Action Spaces in Reinforcement Learning. *Journal of Artificial Intelligence Research* 45: 515–564.

García, J.; and Fernández, F. 2019. Probabilistic Policy Reuse for Safe Reinforcement Learning. *ACM Transactions on Autonomous and Adaptive Systems* 13(3).

Hester, T.; Vecerík, M.; Pietquin, O.; Lanctot, M.; Schaul, T.; Piot, B.; Sendonaris, A.; Dulac-Arnold, G.; Osband, I.; Agapiou, J. P.; Leibo, J. Z.; and Gruslys, A. 2019. Deep Q-learning from Demonstrations. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 4967–4974. Honolulu, USA: AAAI Press.

Hill, A.; Raffin, A.; Ernestus, M.; Gleave, A.; Kanervisto, A.; Traore, R.; Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; and Wu, Y. 2018. Stable Baselines 2.10.0. https://github.com/hill-a/stable-baselines.

Kumar, A.; Fu, J.; Soh, M.; Tucker, G.; and Levine, S. 2019. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. In *Advances in Neural Information Processing Systems*, volume 32, 11784–11794. Curran Associates, Inc.

Kumar, A.; Zhou, A.; Tucker, G.; and Levine, S. 2020. Conservative Q-Learning for Offline Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

Laskey, M.; Staszak, S.; Hsieh, W. Y.; Mahler, J.; Pokorny, F. T.; Dragan, A. D.; and Goldberg, K. 2016. SHIV: Reducing Supervisor Burden in DAgger using Support Vectors for Efficient Learning from Demonstrations in High Dimensional State Spaces. In *2016 IEEE International Conference on Robotics and Automation*.

Mausam; and Kolobov, A. 2012. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Hoboken, NJ, USA: John Wiley & Sons, Inc.

Reddy, S.; Dragan, A. D.; and Levine, S. 2020. {SQIL}: Imitation Learning via Reinforcement Learning with Sparse Rewards. In *International Conference on Learning Representations*.

Richter, C.; and Roy, N. 2017. Safe Visual Navigation via Deep Learning and Novelty Detection. *Robotics: Science and Systems XIII* .

Ross, S.; and Bagnell, D. 2010. Efficient Reductions for Imitation Learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, 661–668. JMLR Workshop and Conference Proceedings.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms *arVix:1707.06347*.

Siegel, N.; Springenberg, J. T.; Berkenkamp, F.; Abdolmaleki, A.; Neunert, M.; Lampe, T.; Hafner, R.; Heess, N.; and Riedmiller, M. 2020. Keep Doing What Worked: Behavior Modelling Priors for Offline Reinforcement Learning. In *International Conference on Learning Representations*.

Simão, T. D.; and Spaan, M. T. J. 2019. Safe Policy Improvement with Baseline Bootstrapping in Factored Environments. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 4967–4974. Honolulu, USA: AAAI Press.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: The MIT Press, 2nd edition.

Taylor, M. E.; Kulis, B.; and Sha, F. 2011. Metric Learning for Reinforcement Learning Agents. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*.

Thananjeyan, B.; Balakrishna, A.; Nair, S.; Luo, M.; Srinivasan, K.; Hwang, M.; Gonzalez, J. E.; Ibarz, J.; Finn, C.; and Goldberg, K. 2020. Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones. *arXiv:2010.15920*.