Autonomous Curriculum Design via Relative Entropy Based Task Modifications

Muhammed Yusuf Satici[1*], Jianxun Wang[1] and David L. Roberts[1]

[1*]Department of Computer Science, NC State University, Raleigh, North Carolina, USA.

*Corresponding author(s). E-mail(s): msatici@ncsu.edu ;
Contributing authors: jwang75@ncsu.edu; robertsd@csc.ncsu.edu;

Abstract

Curriculum learning is a training method in which an agent is first trained on a curriculum of relatively simple tasks related to a target task in an effort to shorten the time required to train on the target task. Autonomous curriculum design involves the design of such curriculum with no reliance on human knowledge and/or expertise. Finding an efficient and effective way of autonomously designing curricula remains an open problem. We propose a novel approach for automatically designing curricula by leveraging the learner's uncertainty to select curricula tasks. Our approach measures the uncertainty in the learner's policy using relative entropy, and guides the agent to states of high uncertainty to facilitate learning. Our algorithm supports the generation of autonomous curricula in a self-assessed manner by leveraging the learner's past and current policies but it also allows the use of teacher guided design in an instructive setting. We provide theoretical guarantees for the convergence of our algorithm using two time-scale optimization processes. Results show that our algorithm outperforms randomly generated curriculum, and learning directly on the target task as well as the curriculum-learning criteria existing in literature. We also present two additional heuristic distance measures that could be combined with our relative-entropy approach for further performance improvements.

1

1 Introduction

Curriculum Learning tries to simplify the learning process of an agent on a difficult-to-learn target task by leveraging the knowledge the agent obtains from learning on a sequence of source tasks related to the target task. Curriculum design for a reinforcement learning problem can be performed either by manually selecting the tasks or by automated curriculum design algorithms. Automated curriculum design algorithms are more efficient in choosing tasks as they eliminate the time spent on manual task generation. On the other hand, automated curriculum generation requires the crafting of a selection criteria that can leverage what the agent currently knows and what needs to be learned to converge to the optimal policy faster. By employing autonomous curricula design techniques, the agent aims to attain improved learning efficiency and reduced dependence on human expertise in generation of curricula and achieves this purpose by adaptively modifying the curriculum based on its level of knowledge during training. We present Relative Entropy based Autonomous Design of Curricula (*READ-C*), a autonomous-curriculum-design framework using the divergence between policies to identify states with high epistemic uncertainty where additional learning could improve the performance of the agent.

Inspiration for *READ-C* comes from psychology learning theory. Studies have shown that using curriculum-design strategies that reduce students' uncertainty improves learning efficiency and knowledge acquisition [1–4], and it is often advantageous for students to successively learn concepts in accordance with some guidance to keep them in the *zone of proximal development* [5, 6]. Furthermore, entropy-based sampling approaches have already been used, mostly in the context of supervised learning and active learning, to query data points about which the agent has the most uncertainty [7]. We hypothesized that a curriculum-design strategy that reduces the learner's epistemic uncertainty using the relative entropy (KL divergence) between the agent policy and the true policy could provide a significant performance improvement to the agent without resulting in a high overhead of curriculum generation.

READ-C, at a high level, measures the relative entropy of the regions in the agent's state space using probability values derived from the agent's policy representation and probability values derived from the true policy, and selects the states of high uncertainty for the curriculum generation using heuristics based on the relative entropy criteria. It assumes the agent has the ability to change its starting state in the given environment and modifies the Markov decision process (MDP) by moving the start state to a discovered state of high uncertainty for the generation of each curriculum step. In this setting, relative entropy encapsulates, for each part of the agent's state space, how much the learner's current policy diverges from a true policy standing for an optimal or near-optimal solution for the domain at hand. The higher the divergence for a region or state of the MDP, the more likely the learner does not have an optimal policy for that region. *READ-C* then modifies the target task to encourage learning in the selected areas of high uncertainty.

We offer two different realizations of the true policy for *READ-C*, namely *READ-C-TD* for teacher dependent entropy calculation and *READ-C-SA* for self-assessed entropy calculation. *READ-C-TD* assumes the existence of a teacher model having learnt the optimal policy and works in a similar fashion as uncertainty-aware active

learning approaches where the agent picks samples for a teacher to label based on the uncertainty of its learning process w.r.t. various entropy metrics including relative entropy [8]. *READ-C-TD* uses a student-teacher architecture where the teacher provides the optimal policy for the relative entropy calculation and the student obtains the uncertainty estimates based on the relative entropy between the student and teacher policies. *READ-C-SA*, on the other hand, solely relies on the information generated during the agent's training process and estimates the relative entropy using a regression model with information from the current and past policies of the agent to mitigate the reliance of the algorithm on already existing policies. Since true policy is often unknown during the training process, using a regressor to estimate the uncertainty values allows the agent to not require the existence of a teacher model for the generation of curricula. The regression model learns from a simpler environment that resembles the target task using the relative entropy between the agent policy and the true policy of the simpler environment as ground truth values in calculation of the uncertainty.

Furthermore, *READ-C* affords different heuristic criteria for prioritizing the regions of uncertainty, and we present three such criteria: 1) directly using relative entropy, 2) relative entropy filtered by proximity to a goal state, and 3) relative entropy filtered by distance to other low-relative-entropy regions. We provide a proof of convergence for the curriculum learning algorithm and explain how the selection of curricula does not change the convergence of the reinforcement learning agent. We evaluate *READ-C* against randomly selected curricula, learning directly on the target task, and a curriculum-generation criteria from the literature. Results show that *READ-C* outperforms the baseline algorithms and the existing curriculum-selection criteria in many cases, and at worst performs similarly to existing algorithms while reducing curriculum-generation overhead.

The main contributions of our research are

- We present a novel entropy-based curriculum generation algorithm for the purpose of facilitating the learning process of a reinforcement learning agent.
- We provide two realizations of this curriculum learning algorithm, a teacher-student framework and a self-assessed approach, to reduce the dependence of the curriculum learning methods on already learnt policies.
- We provide a novel convergence proof for our curriculum generation algorithm to have a theoretical guarantee for the convergence of our reinforcement learning model.
- We provide extensive evaluations of our algorithm against curriculum learning and reinforcement learning methods from the literature in three different domains with varying degrees of difficulty.

2 Related Work

Curriculum-design techniques are either automated or use a human-in-the-loop paradigm. Human-designed approaches focus on understanding how humans teach using curricula [9], and use human knowledge to exploit aspects of the target task that might improve training efficiency [10, 11]. Automated approaches generate a

sequence of tasks, samples, or states without human intervention. We can further divide autonomous approaches into subgroups based on the type of curricula they generate. Some algorithms provide learners a meaningful sequence of data samples or demonstrations [12, 13], while others generate new initial/goal/terminal states at every curriculum step [14, 15]. Other approaches search a parameterized task space by sampling environment parameters based on the agent's current knowledge using contextual reinforcement learning [16, 17]. Some further approaches attempt to remove the dependence on the parameterized task distributions by using optimal transport to generate a curriculum in a contextual RL setting [18, 19]. Some lifelong learning approaches similar to these curriculum learning techniques decompose the task space into subregions based on diverse model primitives and learn subpolicies for each region of the task space in a bottom-up manner to sequentially solve multiple tasks [20]. Task-sequencing algorithms either heuristically pre-generate a sequence of tasks before training [21, 22] or adaptively generate curricula by leveraging the learner's knowledge at various points during training [23–25]. *READ-C* uses adaptive modifications of the agent's start state, so our focus will be on adaptive curriculum-generation algorithms that perform modifications to the target task.

Adaptive curriculum-generation algorithms train on the target task and use the knowledge they obtain to determine which tasks and/or states benefit the learner. An approach similar to *READ-C* is reversed-curriculum learning that generates a curriculum of start states by sequentially expanding backwards from the goal state [14]. In contrast, *READ-C* uses heuristic selection of start states based on the learner's relative uncertainty. Another approach samples start states proportional to the Euclidean norm of the gradient of a performance measure defined on the value function, guiding the learner towards regions of the environment where nearby states have wildly-different value function estimates [26]. It requires learning a start state selection policy in addition to the RL policy, which increases the training time. Finally, Narvekar et al. [25] present an adaptive task sequencing algorithm that selects curriculum tasks based on the maximal change in the learner's policy, guiding the agent towards tasks that are expected to improve the policy most. This approach requires the learner to train on the target task and every source task for every curriculum step, which results in significant curriculum-generation overhead. See Narvekar et al. [27] for more analysis of curriculum learning approaches.

READ-C-TD also relates to active learning in that it allows a student to query a teacher's policy for the purpose of identifying high uncertainty regions. Settles and Craven [8] use KL Divergence of an ensemble of models to identify the samples that the model is least confident in how to label. Madhawa and Murata [28] test multiple uncertainty measures including information gain between the model predictions and model posterior distribution for graph neural networks. Our algorithm shows similarities to these approaches in the way it uses relative entropy to determine the uncertainty of the agent regarding its learning process. However, we adapt this active learning process to a reinforcement learning problem and measure the agent's uncertainty in its state space rather than data instances. Bougie and Ichise [29] offers an active learning framework using goal-driven demonstrations where the RL agent

identifies in which states the feedback is most needed based on Bayesian or quantile confidence and asks for guidance from an expert only for the states it has low confidence of reaching its goal. In contrast, we offer a curriculum learning framework that generates a curriculum for the RL agent without using demonstrations from an expert through a self-assessed learning approach. Settles [7] explains more about the traditional active learning approaches.

READ-C relates to options learning literature due to the interleaved learning process it performs based on the uncertainty of the agent over different regions of the state space. Options in the context of options learning represent higher-level actions or sub-policies that the agent learns to execute. By using these options, the agent manages to skip parts of the training process where it already has a good understanding of optimal policy. The traditional options learning algorithms use tabular Q-learning and planning methods to explore the option space [30, 31]. Adaptive Skills, Adaptive Partitions framework simultaneously learns options for sub-regions of the state space using a modified gradient calculations based on the hyperplanes that partition the state space into subtasks [32]. Another sparse sampling approach uses a generative model to sparsely sample a series of states that would construct a small MDP to be used for planning the optimal path [33]. In more recent deep-RL research, actor-critic architectures have been used to model the agent and the options policies [34–36]. Bacon et al. offers an option-critic architecture where they train the option networks using stochastic gradient descent similar to how an actor would learn and they use a planner to determine which option policy to run at the current step of training [35]. Harb et al. improves upon the option-critic by using a deliberation cost model instead of a planner to select good options [34]. Harutyunyan et al. adds a termination critic to the option-critic architecture to optimize the termination condition of the options learning framework [36]. Although these options learning algorithms show similarities to *READ-C*, their focus lies on skipping the actions that do not offer additional information to the agent whereas the curriculum learning algorithm tries to bring the agent close to the regions of the state space where the learning could improve the policy the most.

3 Problem Formulation

Here we provide background on the RL method and formally define the curriculum-learning problem in this context.

3.1 Reinforcement Learning

We model the learning process as a Markov decision process $M = <S, A, T, R, s_i, S_g, \gamma >$, a tuple consisting of a set of states S, a set of actions A, a transition function T, a reward function R, an initial state s_i, a set of terminal states S_g and a discount factor γ. The transition function $T : S \times A \times S \rightarrow [0, 1]$ corresponds to the probability of transitioning from a state $s \in S$ to another state $s' \in S$ using a valid action $a \in A$ at state s. All of the environments we use are deterministic, so taking action a in state s always transitions into the same resulting state s'—although that is not a requirement for *READ-C*. The reward function $R : S \times A \times S \rightarrow F$ maps from

a state, action, state tuple to a real number corresponding to the reward signal the learner receives. The policy $\pi : S \rightarrow A$ maps states to actions. The cumulative reward G at time t is the discounted sum of all feedback the agent receives from t until it reaches a terminal state,

$$G_t = \sum_{k=0}^{\tau-t} \gamma^k R_{t+k}$$

where γ is the discount factor, τ is the time to reach a terminal state, and R_{t+k} is the feedback received at state s_{t+k} [37]. The agent's objective is to learn the optimal policy π_M^* that maximizes G.

For discrete action spaces, we train using dual deep Q-networks (Dual-DQN) [38]. We use two four-layered, fully-connected, feed-forward networks. The networks take the states as input and output Q-value estimates for each available action. One neural network serves as the learned model, and the other provides target Q-value estimates for batch updates. The loss function is

$$L(\theta) = E_{s,a,r,trm,s' \sim RB} \left[\frac{1}{2} (r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2 \right] \qquad (1)$$

where θ is the weights of the learned model, θ^- denotes the weights of the target model, γ is the discount factor, s is the current state, a is the action taken at state s, s' is the next state and a' is the action taken at state s' [38].

For continuous action spaces, we employ an actor-critic architecture similar to [39]. We use a four-layered, fully-connected, feed-forward network for the actor and critic. We allocate 256 nodes at every layer and employ rectified linear unit (reLU) activation function between each layer. The critic receives the state of the environment as its input and outputs the value function estimate. The actor network outputs two real vectors which we treat as the mean and standard deviation of the multi-dimensional normal distribution that we sample the actions from. We use the advantage loss to train the actor which is given as

$$L_a(\theta, \omega) = E_{s,a,r,trm,s' \sim RB} \left[log(\pi(a; s, \theta)) \left(r + \gamma \max_{a'} V(s'; \omega^-) - V(s; \omega) \right) \right]$$

where ω is the learned critic weights, ω^- is the target critic weights, θ is the actor weights, π is the actor policy that is modeled as a normal distribution and γ is the discount factor [39]. We train the critic network using the mean square error which is given as,

$$L_c(\theta, \omega) = E_{s,a,r,trm,s' \sim RB} \left[\frac{1}{2} (r + \gamma \max_{a'} V(s'; \omega) - V(s; \omega))^2 \right] \qquad (2)$$

where ω is the weights of the critic, ω^- denotes the weights of the target critic model, and γ is the discount factor. We use the *epsilon*-greedy policy same as the DQN mentioned above.

At each training step the agent takes a single action in the environment, records the tuple (s, a, r, trm, s') into a replay buffer (RB) (where trm indicates whether s is a terminal state), randomly samples a batch of tuples from the buffer, and performs a single batch update on the learned model. The target model weights are updated using the weights of the learned model at the end of each episode.

3.2 Defining Curricula

A *READ-C* curriculum of length d consists of a sequence of start states $C = (s_1 \ldots s_d)$ such that $s_i \in S$ for all $i \in [1, d]$. Let π_i be the agent's policy after training on MDPs M_1 to M_i employing the start states s_1 to s_i. Then, a curriculum step i produces the transition $\pi_i \to \pi_{i+1}$ where policy π_i is the starting policy for the curriculum step and π_{i+1} is the policy attained by training on MDP M_{i+1}. Hence, each curriculum step is a modification to the agent's policy based on the current MDP M_{i+1}. Note that all of the MDPs in our formulation use the same state space, action space, dynamics, and reward function. We compare performance using either the total reward the agent achieves during training (asymptotic) or the total time spent on the curriculum and target task to obtain the maximum cumulative reward G^{max} (time to convergence).

The objective of the curriculum learning algorithm is to improve the sample efficiency and the learning speed of the reinforcement learning process. In environments where it is often costly to retrieve samples, it is advantageous to design guidance methods that could direct the agent towards the parts of the state space that the training would create improvements in the learnt policy of the agent. *READ-C* aims to modify the start state of the environment with the purpose of bringing the agent closer to the states it is struggling to learn to facilitate the learning process and reduce the number of samples needed from the environment to reach the optimal policy. Since all of our algorithms perform one batch update after receiving each sample from the environment, reducing the sample complexity, in this case, also reduces the number of batch updates made during the learning process, speeding up the training time of the agent.

3.3 Transfer Learning

The learner retains the same model throughout training, using the model obtained in the prior curriculum step as the starting point for the next step. Generating new models or transferring weights is not required. The learner also retains the contents of the replay buffer across curriculum steps.

4 Curriculum Learning

We first describe our uncertainty measures and provide pseudo-code for the high-level description of *READ-C*. Then, we define two main realizations of *READ-C*, namely *READ-C-SA* and *READ-C-TD*, and describe two distance-metric variants of the relative-entropy heuristic.

4.1 Measuring Uncertainty

We measure the uncertainty between two policies, namely the learnt policy and the true policy, through relative entropy. The relative entropy is defined as

$$D_{KL}(P_{true}||P_{learnt}) = \sum_{a \in A} P_{true}(s,a) log\left(\frac{P_{true}(s,a)}{P_{learnt}(s,a)}\right) \tag{3}$$

where P_{true} denotes the action probabilities obtained from a reference policy and P_{learnt} represents the probabilities of a learnt policy at a certain checkpoint during the training. We take the relative entropy of the reference probabilities with respect to the learnt probabilities to measure the reduction in uncertainty the agent would have in its policy if it were to use the correct policy instead of its own. In this manner, our uncertainty metric reflects how inaccurate the agent is in its action selection for a given state of MDP. Since the true policy already has good performance on the target task, the relative entropy is expected to drop as the agent improves its policy and comes closer to the estimate of the true policy. The probability distribution here could be defined over a discrete or continuous variable depending on how the neural networks treat the output parameters.

Since the policy relies on the Q-value estimates of the agent, we require a transfer function that could convert the Q-value estimates to probability values for the relative-entropy calculation. For this purpose, we use softmax and calculate the probability of action selection as

$$P(s,a) = \frac{e^{\frac{Q(s,a)}{||Q(s)||_2}}}{\sum_{a \in A} e^{\frac{Q(s,a)}{||Q(s)||_2}}}$$

where $Q(s,a)$ is the Q-value for the state, action pair (a,s), $Q(s)$ is the list of Q-values for state s.

4.2 *READ-C*

Given the definition of uncertainty in Eq. 3, we construct a high-level curriculum-learning algorithm that affords the use of different RL algorithms for the training of the agent. Algorithm 1 is the high-level pseudo-code for *READ-C*. It initializes the target environment and the agent model (Line 4). It starts with an empty curriculum (Line 5) and trains the agent model for η iterations using the train function defined in Algorithm 2 to generate an initial agent policy (Line 7)— the starting point for relative-entropy calculation. Each iteration of the loop (Line 8) creates a new curriculum step by selecting a new start state and the agent trains on MDP with the new start state. The function named uncertainty defined in Algorithm 4 (Line 9) uses Eq. 3 to calculate relative entropy values of current policy for each state in the set of visited states (SB), and returns the state with the highest uncertainty to Algorithm 1. Then, Algorithm 1 changes the start state to the state of highest uncertainty (Line 10) and trains on the modified MDP until the convergence criteria—if the entropy has not reduced for the last 10 episodes—is met (Line 12). This convergence method receives

8

the relative entropy of the selected state from the agent before and after each training episode, which is not computationally expensive since the algorithm calculates entropy for only a single state—not all regions. The process repeats until the curriculum reaches a defined length, at which point the student trains on the original target task until an overall convergence criterion—if the agent has not reached the highest cumulative reward for the last 10 episodes—is met (Line 17).

1: **Inputs:** target MDP: M_{tar}.
2: **Consts:** # of training iterations: η; curriculum length threshold: MAX_LENGTH.
3: **Vars:** learning model: NN; environment: ENV; curriculum: C; state buffer: SB; convergence criteria: $conv$.
4: $NN_{agent}, ENV_{tar} \leftarrow initialize(M_{tar})$
5: $C \leftarrow \{\}$
6: Set $conv$ to η training iterations
7: train(NN_{agent}, ENV_{tar}, $conv$, SB)
8: **while** $|C| < MAX\_LENGTH$ **do**
9:     $s \leftarrow uncertainty(NN_{agent}, ENV_{tar}, SB)$
10:     $ENV_{cur} \leftarrow$ Set s as the start state of ENV_{tar}
11:     Set $conv$ to 10 episodes of no entropy reduction
12:     train(NN_{agent}, ENV_{cur}, $conv$, SB)
13:     $C \leftarrow C \cup s_i$
14: **end while**
15: Set $conv$ to highest cumulative reward
16: Make s_i of original M_{tar} the s_i of ENV_{tar}
17: train(NN_{agent}, ENV_{tar}, $conv$, _)

Algorithm 1: *READ-C*

Algorithm 2 describes the train function used in Algorithm 1. It receives the learning model, the current environment, the convergence criteria and the state buffer as inputs and performs regular RL training on the environment using the learning model until the convergence criteria is met (Line 4). It initializes the environment at the start state (Line 5) and selects the action to take using an ϵ-greedy policy on the learning model (Line 8). Then, it takes the selected action on the environment (Line 9) and stores the (s,a,r,trm,s') tuples in RB for future training (Line 10). It also stores every unique state visited so far in SB for the entropy calculation (Line 12). Then, it performs a single step of optimization on the learning model using a minibatch of samples from RB (Line 15). Algorithm 2 affords the use of different learning models and policies as it (neither *READ-C*) does not rely on any specific policy or network architecture for the entropy calculation or agent training. It can also be easily adapted to non-episodic environments by removing trm variable (Line 6) and replacing the loop that checks for terminal states (Line 7) with the outer loop that checks a suitable $conv$ for non-episodic training (Line 4).

4.3 Relative-Entropy Calculation for *READ-C-TD*

1: **Inputs:** learning model: NN_{agent}; environment: ENV; convergence criteria *conv*, state buffer: SB.
2: **Consts:** set of terminal states of ENV: S_g; initial state of ENV: s_i.
3: **Vars:** replay buffer RB.
4: **while** *conv* is not satisfied **do**
5:     $s \leftarrow s_i$ of ENV
6:     $trm \leftarrow s \in S_g$
7:     **while** trm is False **do**
8:         Select a using ϵ-greedy policy on $NN_{agent}(s)$
9:         Take action a in ENV and observe r, s'
10:         Store (s, a, r, trm, s') in RB
11:         **if** $s \notin SB$ **then**
12:             Store s in SB
13:         **end if**
14:         Sample a minibatch B from RB
15:         Perform an optimization step on NN_{agent} using B
16:         $s \leftarrow s'$
17:         $trm \leftarrow s \in S_g$
18:     **end while**
19: **end while**

Algorithm 2: Train

1: **Inputs:** learning model: NN_{agent}; environment: ENV; state buffer: SB.
2: **Consts:** a teacher model: T.
3: **Vars:** Q-values for state s: Q^s; probabilities for state s: P^s.
4: **Outputs:** the new start state: s.
5: states $\leftarrow$ Sample a subset of states from SB
6: $UNCERTAINTIES[0, ..., N] \leftarrow 0$ where N is the number of states
7: **for** s_j in *states* **do**
8:     $Q^s_{agent} \leftarrow$ get Q-values of s_j from NN_{agent}
9:     $Q^s_{teach} \leftarrow$ get Q-values of s_j from T
10:     $P^s_{agent} \leftarrow softmax(normalize(Q^s_{agent}))$
11:     $P^s_{teach} \leftarrow softmax(normalize(Q^s_{teach}))$
12:     $UNCERTAINTIES[j] \leftarrow entropy(P^s_{teach}, P^s_{agent})$
13: **end for**
14: $k \leftarrow$ argmax of $UNCERTAINTIES$
15: return s_k

Algorithm 3: Relative-Entropy Calculation for READ-C-TD

Algorithm 3 describes the relative-entropy calculation for the teacher dependent variant of *READ-C*. It necessitates the use of a teacher model that already knows the optimal policy of the target environment to compute the relative entropy values for curriculum selection. If there happens to be an already trained model for the target MDP that could serve as the teacher, Algorithm 3 offers a simpler way of calculating the uncertainty metric as it does not require β iterations of training Algorithm 4 does on the learning model to generate two policies of the agent at different checkpoints. On the other hand, if the teacher needs to be trained from scratch, Algorithm 4 offers

a much faster performing framework. If there exist an agent residing in a remote server which the user could query to measure the relative entropy but cannot copy its weights directly to the agent due to privacy and/or security reasons, then that agent could be used in *READ-C-TD* as a teacher model, which present one possible use case of *READ-C-TD* despite its requirement of already having a learnt model. A more concrete example of this situation would be commercial LLMs where we are not given open source access to the neural network itself but we can interact with the model through an API. In a similar sense, when a deep RL model is used in a copyrighted setting, we can interact with the model to generate action probabilities without having the need to access the inner mechanisms of the model. Then, using the output of the model allows me to perform training on *READ-C-TD* and get the benefit of teacher dependent curriculum generation.

Algorithm 3 contains a teacher model that is trained to convergence using Algorithm 2 without employing any curriculum learning. Algorithm 3 receives the agent model, the environment and the state buffer SB as its inputs and uses the teacher along with the information it has on the agent to calculate relative entropy. Algorithm 3 calculates the probability values based on the Q-values estimates of the sampled states (Lines 10 & 11). Then, it calculates the relative entropy between the agent's and teacher's policies and uses it as the uncertainty value for the given state (Line 12). Since the teacher knows the optimal policy, the relative entropy, in this case, shows how much the agent's knowledge is diverging from the optimal policy. Finally, *READ-C-TD* chooses the highest-uncertainty state as the new start state (Line 14).

4.4 Relative-Entropy Calculation for *READ-C-SA*

Algorithm 4 describes the *uncertainty* function given in Algorithm 1 at Line 9 and estimates the relative entropy for the self-assessed version of *READ-C*. *READ-C-SA* uses a regression model to estimate the relative entropy between the agent and teacher and gets rid of the requirement of having a teacher model already trained in the target environment. The regressor serves as a proxy function for estimating the true relative entropy using the information theoretic data generated on a simpler environment. It attempts to learn an underlying relationship between the agent's knowledge of the task and the agent's uncertainty of the action selection. Since *READ-C-SA* uses the true policy of the simpler environment in its training, the true policy in this setting does not necessarily represent how the agent should act in the target environment but due to the simpler environment being a subtask of the target environment having similar characteristics, it is expected that the knowledge acquired on the simpler task would be generalizable to the target task in calculation of the uncertainty. *READ-C-SA* offers a compromise in that it allows us to predict the relative entropy values more efficiently without relying on the teacher policy but it also introduces additional error coming from the training of the regression model, which we try to minimize by selecting a proper regression model for the problem at hand.

Algorithm 4 samples a subset of the states in SB for the uncertainty calculation (Line 5). This step is crucial in ensuring that the algorithm could run in large state spaces especially if the state space is continuous. Then, the algorithm performs training for β iterations to generate a second learning model (Line 9). The two

1: **Inputs:** learning model: NN_{agent}; environment: ENV; state buffer: SB.
2: **Consts:** # of training iterations: β.
3: **Vars:** Q-values for state s: Q^s; probabilities for state s: P^s; convergence criteria: $conv$.
4: **Outputs:** the new start state: s.
5: states $\leftarrow$ Sample a subset of states from SB
6: $UNCS[0,...,N] \leftarrow 0$ where N is the number of states
7: Set $conv$ to β training iterations
8: $NN_{past} \leftarrow NN_{agent}$
9: train(NN_{agent}, ENV, $conv$, SB)
10: **for** s_j in $states$ **do**
11: $\quad Q^s_{past} \leftarrow$ get Q-values of s_j from NN_{past}
12: $\quad Q^s_{agent} \leftarrow$ get Q-values of s_j from NN_{agent}
13: $\quad P^s_{past} \leftarrow softmax(normalize(Q^s_{past}))$
14: $\quad P^s_{agent} \leftarrow softmax(normalize(Q^s_{agent}))$
15: $\quad relEntropy \leftarrow entropy(P^s_{agent}, P^s_{past})$
16: $\quad entropyCur \leftarrow entropy(P^s_{agent})$
17: $\quad entropyPast \leftarrow entropy(P^s_{past})$
18: $\quad UNCS[j] \leftarrow reg(relEntropy, entropyCur, entropyPast, Q_{past}, Q_{agent})$
19: **end for**
20: $k \leftarrow$ argmax of $UNCS$
21: return s_k

Algorithm 4: Relative-Entropy Calculation for READ-C-SA

learning models of the agent, NN_{past} and NN_{agent}, at two different checkpoints in time helps us measure how much the agent is diverging from its own policy as time passes. Although the relative entropy calculated between NN_{past} and NN_{agent} does not reflect the true relative entropy, it represents the speed of learning and change in the Q-value estimates of the model. For every sampled state, *READ-C-SA* calculates the Q-value estimates of both models (Lines 11 & 12) and converts them to probabilities by normalizing and softmaxing (Lines 13 & 14). The algorithm normalizes the Q-values using the L2-norm to make the results independent of the scale of the Q-values and uses softmax to obtain probability estimates. It calculates relative entropy and entropies on the probability values using Eq.3 and 4 (Lines 15, 16 & 17). By doing these operations, *READ-C-SA* generates information theoretic data about the agent that we use for estimating the relative entropy for the given state. We pass the data to a regressor model that was trained on the relative-entropy values of a smaller and easier-to-train environment. Since the input features to the regressor are independent from the state and action dimensions of the environments, we can deploy the trained regression model from a simpler environment to the target environment without any modification. The regressor model serves as a proxy function for the relative entropy between the true policy and the agent policy of the target environment as long as the easier-to-train environment shares the same characteristics as the original task. The regressor predicts the uncertainty of the states based on the relative entropy, entropy and q-value estimates of the agent policy and records it in an array (Line 18). Finally, the algorithm chooses the maximum uncertainty state as the new start state (Line 20) and returns it (Line 21).

12

Our regressor is a gradient-boosting machine (GBM). It trains on the data (relative entropy, entropy, Q-values and number of visits per state) generated by a learning model of a simpler environment. In addition to relative entropy, we also use the entropy for each policy in the training of the regression model. The entropy is defined as

$$H(\pi(\cdot|s)) = -\sum_{a \in A} P(s,a)log(P(s,a)), \tag{4}$$

where $H(\pi(\cdot|s))$ is the entropy of the action policy π for the state s using the probabilities P coming from a given *READ-C* learner's policy. The training of the regressor follows the same pattern as Algorithm 4 to generate the data (Lines 13, 14, 15, 16 & 17) but instead of using the data to predict the uncertainty values, the regressor learns a mapping from the data to the relative entropy between the agent and the optimal policy. As long as the trained environment shows a high level of similarity to the target MDP, the regressor generates accurate uncertainty estimates for every sampled state. Since we employ a simpler MDP for the regressor optimization, the training of GBM takes much less time than the curriculum-learning process allowing READ-C-SA to efficiently calculate uncertainty values without having a need to measure the true policy on the target MDP.

The regression model takes the information theoretic data (relative entropy, entropy, Q-values and number of visits per state) generated on the simplified version of the target task as input and uses the true relative entropy as the ground truth value for its training. The true relative entropy is calculated between the agent policy and the policy of a teacher model trained on the simpler environment. Since it is easier to train on the simplified environment than the target task, generating the teacher model in the case of *READ-C-SA* takes much less time than *READ-C-TD*, allowing the curriculum generation process to be more efficient. The regression model uses the mean squared error between its output and the true relative entropy as the loss function. Once the regression model trains on the data generated for every visited state of the simpler environment, it gets deployed in the target environment to predict the uncertainty values, which is shown in Algorithm 4. We train different types of regressors, namely Linear Regression, Random Forest Regression, Support Vector Machine and GBM, and choose the best performing regressor (GBM) to be deployed in the target environment.

4.5 Clustering of States

Optionally, one could also perform an Agglomerative Hierarchical Clustering [40] over sampled states, merging clusters using the Ward criterion [41] until a distance cutoff is reached, and calculate the uncertainty values over the clusters of states rather than single states. In this case, the fundamental flow of the algorithm remains unchanged as it still calculates a single uncertainty value for every state but the uncertainty values of the states in the same cluster gets aggregated at the end of the algorithm to find the uncertainty of each cluster. Then, the algorithm chooses the highest-uncertainty cluster and selects a random state from that cluster as the new start state. The clustering of the states enables us to reduce the deviation in our measurements

since single states might show high variance in their uncertainty from one run of the algorithm to the other. It is more advisable to perform clustering if the nature of the state space allows such a partitioning but in domains where the state space cannot generate meaningful clusters, the algorithm needs to run in its original form without clustering.

4.6 Heuristic Variants

READ-C as described above chooses the maximum relative entropy, which does not account for the underlying structure of the problem. For example, a region of high uncertainty may be located in an area of the state space extremely far from the start and goal states of the target task, or that may be rarely encountered when following an optimal policy. Relative entropy alone would prioritize training in such regions even if not advantageous. Therefore, we present two additional relative-entropy heuristics:

1) Proximity: proximity to *positive reward terminal states*, which filters out 80% of the regions having the worst proximity (average Euclidean distance in our case) to terminal states that provide a positive reward, and computes relative entry only for the remaining 20%;

2) Max Distance: proximity to *low-entropy regions*, which classifies regions as high-entropy (one standard deviation above the mean relative entropy) and low-entropy (one standard deviation below the mean relative entropy), and sorts the regions of high-entropy based on the (Euclidean, in our case) distance to the closest low-entropy region (selecting the region with the highest distance). This technique promotes selected regions not resulting in lengthy training episodes to improve efficient training.

Figure 1 depicts a 10 by 10 environment containing a single agent (straw man) at the top right corner and a positive reinforcement item (white circle) at the bottom left corner. The rest of the environment is clustered into regions with varying relative entropies. Green denotes low-entropy regions, red (containing dotted shapes inside) denotes high-entropy regions, and yellow and orange denote moderate-entropy regions. Because there are no actions leading into walls, it is faster to learn the regions close to the edges of the environment; hence, those regions are mostly low-to-moderate entropy. Similarly, the regions near the white circle have low entropy because of the short-horizon goal-state reward. The regions near the center of the environment have high entropy. Black lines indicate the closest low-entropy region for the high entropy regions. In this situation, relative entropy by itself would prioritize one of the three red regions, proximity would prioritize the red region with light blue edges (containing a dotted triangle), and maximum distance to the low-entropy regions would prioritize the red region with dark blue edges (containing the dotted diamond). Therefore, the proximity variant of *READ-C* will aim to keep the agent close to the positive reinforcing terminal state whereas the maximum distance to low entropy variant will aim to keep it away from the already learned regions. The relative entropy by itself could still choose a suitable region to train on in this example but due to the randomness of the training process caused by the ϵ-greediness, there will be no guarantee that the region of highest uncertainty the relative entropy generates would be the most useful region for the agent to reduce its uncertainty. However, by using the distance metrics, the
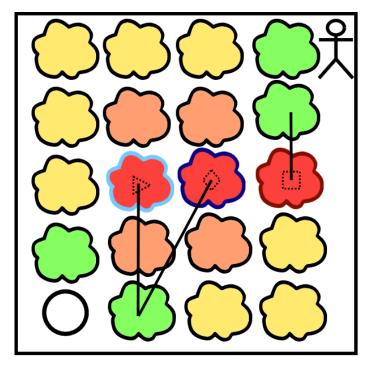
Fig. 1 Visualization of the highest uncertainty region for *READ-C* variants in an environment with a single goal.

agent would at least make sure that the region being selected is not far away from the goal states and/or states with useful information regarding the optimal policy path.

4.7 Proof of Convergence for *READ-C*

Let L_a be the loss function for the actor and L_c be the loss function for the critic.

$$L_a(\theta_t, \omega_t, z_t) = E_{\tau \sim \pi(\theta_t, z_t|s_0)}\left[log(\pi(a_t|s_t, \theta_t))(R_t + \gamma V(s_{t+1}|\omega_t) - V(s_t|\omega_t))\right] \quad (5)$$

where $\pi(a_t|s_t, \theta_t) \sim \mathcal{N}(\mu_t, \sigma_t^2)$, s_t is the environment state, a_t is the action taken at time t, R_t is the reward received from the environment, γ is the discount factor, ω_t is the weights of the critic network, θ_t is the weights of the actor network, V is the value function estimated by the critic, π is the policy of the actor network, z_t is the controlled Markov process modeling non-additive noise, τ is the trajectory of the agent following policy π and (μ, σ) pair is the mean and standard deviation of the Gaussian distribution modeling the actor network policy π.

$$L_c(\theta_t, \omega_t, z_t) = E_{\tau \sim \pi(\theta_t, z_t|s_0)}\left[\frac{1}{2}(R_t + \gamma V(s_{t+1}|\omega_t) - V(s_t|\omega_t))^2\right] \quad (6)$$

The gradient update rule for the loss in advantage actor-critic model is

15

$$\theta_{t+1} = \theta_t - \alpha(t)(\nabla L_a(\theta_t, \omega_t, z_t) + m_t) \tag{7}$$

where $\alpha(t)$ is the learning rate for the actor at time t and m_t is the martingale sequence noise.

$$\omega_{t+1} = \omega_t - \beta(t)(\nabla L_c(\theta_t, \omega_t, z_t) + m_t) \tag{8}$$

where $\beta(t)$ is the learning rate of the critic network and m_t is the martingale sequence noise.

Theorem 1 Under the assumptions of Section 4.7.1, Eq. 7 & 8 converges to

$$(\theta_t, \omega_t) \to (\theta^*, \omega^*) \text{ as } t \to \infty \tag{9}$$

4.7.1 Assumptions

In this section, we describe the assumptions needed to prove the convergence of *READ-C* to an optima. We use the problem formulation from [42] for crafting the assumptions of our proof. We treat the actor-critic learning process as a two time-scale optimization process and make use of the actor-critic convergence proof in proving that the selection of curricula does not change the convergence of the reinforcement learning algorithm.

Assumption 1

z_t is a Markov process taking values in a compact metric space w.r.t. a continuous transition function.

Assumption 2 L_c and L_a are J-Lipschitz and J-Smooth functions such that

$$||\nabla L_a(\theta_t, \omega_t, z_t)|| \leq J \quad \forall \theta \& \omega \in \mathbb{R} \tag{10}$$

$$||\nabla^2 L_a(\theta_t, \omega_t, z_t)|| \leq J \quad \forall \theta \& \omega \in \mathbb{R} \tag{11}$$

where J is the Lipschitz constant.

Assumption 3

m_t is a martingale difference sequence with bounded second moment such that

$$\begin{aligned} E[m_t^2|F_t] &\leq K + \theta_t^2 \\ E[m_t^2|F_t] &\leq K + \omega_t^2 \end{aligned} \tag{12}$$

where F_t is history of martingale variables up until time t and K is the martingale constant.

Assumption 4

$$\sum_t \alpha(t) = \beta(t) = \infty \,, \, \sum_t (\alpha(t)^2 + \beta(t)^2) < \infty \tag{13}$$

for $\alpha(t), \beta(t), t \geq 0$.

Assumption 5

Assume that z_t has an ergodic occupation measure $\Gamma(\theta, \omega)$. Then, let l_c be $\nabla L_c(\theta_t, \omega_t, z_t)$ and l_a be $\nabla L_a(\theta_t, \omega_t, z_t)$.

$$\bar{l}_c = \int l_c(\theta, \omega, z)\Gamma(\theta, \omega) \tag{14}$$

where $\bar{l}_c$ is the ordinary differential equations for the gradient of the loss function.

$\bar{l}_c$ has a global attractor set B_θ and a stable point $\lambda(\theta)$ such that $\lambda(\theta)$ is a J-Lipschitz map.

Furthermore,

$$\bar{l}_a = \int l_a(\theta, \lambda(\theta), z)\Gamma(\theta, \lambda(\theta)) \tag{15}$$

$\bar{l}_a$ has a global attractor set A.

Assumption 6 The weights of the neural networks have a tight upper bound such that

$$sup(||\theta_t|| + ||\omega_t||) < \infty \tag{16}$$

4.7.2 Effect of Curriculum Selection on Convergence

The actor-critic architecture is expected to converge to an optima based on the convergence proofs provided in Wu et al. and Holzleitner et al. [43, 44] using the assumptions from two time scale optimization processes [42]. Under these conditions, we must show that the selection of curriculum states does not violate the convergence of the actor-critic model for *READ-C* to also converge.

READ-C determines the start state for the environment based on the KL divergence between the agent's policy and the true policy of the MDP which is given as

$$s_0 = argmax_{s \in S} KL(s|\theta_t, z_t) \tag{17}$$

If we assume that the output of the actor-critic model is normally distributed (which is an assumption that holds for advantage actor-critic), then KL divergence will have a closed form solution which is

$$KL(s|\theta_t, z_t) = log\left(\frac{\sigma_{lrn}(s|\theta_t, z_t)}{\sigma_{true}}\right) + \frac{\sigma_{true}^2}{2\sigma_{lrn}^2(s|\theta_t, z_t)} + \frac{(\mu_{true} - \mu_{lrn}(s|\theta_t, z_t))^2}{2\sigma_{lrn}^2(s|\theta_t, z_t)} \tag{18}$$

where μ and σ refer to the mean and standard deviation of the Gaussian distribution the policy samples the actions from. $(\mu_{true}, \sigma_{true})$ comes from an already existing optimal policy and $(\mu_{lrn}, \sigma_{lrn})$ comes from the current policy of the agent.

The expected loss for following a trajectory initiated at a starting state generated by the curriculum learning algorithm would be

$$\sum_{s_c \in S} P(s_0 = s_c | KL(s_c|\theta_0, z_0)) P(\tau | s_0, \theta_0) E[L_a(\theta_t, \omega_t, z_t, s_t)] \tag{19}$$

where CL refers to the set of start states s_0 calculated by Eq.17 w.r.t. KL divergence given in Eq.18.

Given the convergence of the actor-critic, we wish to show that the distribution of start states above does not change the convergence of *READ-C*.

For the first base case, if $s_c = s_g$, then it is trivial to show that the algorithm will converge since the agent is already at a goal state and does not need to take any action to reach convergence.

For the second base case, if $s_c = neighbor(s_g)$, then the expected loss would be

$$P(s_g|s_0, \theta_0)E[L_a(\theta_t, \omega_t, z_t, s_t)] \tag{20}$$

for a sufficiently large exploration rate of ϵ, $P(s_g|s_0, \theta_0) > 0$ and we know that $E[L_a(\theta_t, \omega_t, z_t, s_t)]$ converges when $t \to \infty$ based on the actor-critic convergence proof [42, 44]. Then, the algorithm is supposed to converge to an optima when the start state is a neighbor of a goal state.

Having established the base cases, we now assume that *READ-C* converges when $s_c = s_n$ where s_n is a state on trajectory τ. We need to show that *READ-C* converges when $s_c = s_{n-1}$ where s_{n-1} is a prior state to s_n on the trajectory. The expected loss is

$$P(s_0 = s_{n-1}|KL(s_{n-1}|\theta_{n-1}, z_0))P(\tau|s_{n-1}, \theta_{n-1})E[L_a(\theta_{t-1}, \omega_{t-1}, z_{t-1}, s_{t-1})] \tag{21}$$

which is equivalent to

$$P(s_0 = s_{n-1}|KL(s_{n-1}|\theta_{n-1}, z_0))P(s_n|s_{n-1}, \theta_{n-1})P(\tau|s_n, \theta_n)E[L_a(\theta_{t-1}, \omega_{t-1}, z_{t-1}, s_{t-1})] \tag{22}$$

$$E[L_a(\theta_{t-1}, \omega_{t-1}, z_{t-1}, s_{t-1})] = E[L_a(\theta_t, \omega_t, z_t, s_t)] - \langle l_a(\Upsilon_{t-1}), \theta_t - \theta_{t-1} \rangle \tag{23}$$

based on Taylor's Theorem [45]. As $t \to \infty$, we know that $|\theta_t - \theta_{t-1}| \to 0$ due to the convergence of the actor, which makes the inner product term 0, leaving us with $E[L_a(\theta_t, \omega_t, z_t, s_t)]$. Similarly,

$$P(s_0 = s_{n-1}|KL(s_{n-1}|\theta_{n-1}, z_0))P(s_n|s_{n-1}, \theta_{n-1}) \propto P(s_0 = s_n|KL(s_n|\theta_n, z_0)) \tag{24}$$

Then, we could rewrite Eq.22 as

$$P(s_0 = s_n|KL(s_n|\theta_n, z_0))P(\tau|s_n, \theta_n)E[L_a(\theta_t, \omega_t, z_t, s_t)] \tag{25}$$

We know that this equation converges due to the inductive assumption. Then, *READ-C* must converge for $s_0 = s_{n-1}$.

We have shown that the curriculum selection does not change the convergence of *READ-C*. Given that the curriculum selection criteria does not change the convergence, *READ-C* must converge under the assumptions given in Section 4.7.1.

5 Evaluation Methods

Here we describe the test environments, the experimental setup, and the state representation used for each domain.

5.1 Test Environments

5.1.1 Key-Lock

We use a 20×20 2D grid environment as well as continuous state-action space domains to test the algorithms. The first domain contains keys, locks, pits, and obstacles similar to Narvekar et al. [25], Konidaris and Barto [46]. The agent's task is to pick up the key and unlock the lock while avoiding the pits and obstacles. Each key picked up gives a reward of 500 and each lock unlocked gives a reward of 1,000. Falling into a pit receives -400. All other actions including moving into an obstacle receive -10. Moving into an obstacle results in no state transition. The learner can only move in cardinal directions and is assumed to have obtained the key or the lock if its location matches the location of the key or the lock. An episode terminates when the agent obtains all the keys and locks, falls into a pit, or reaches 100 time steps. A state in the key-lock environment is represented as a vector, including the Euclidean distance from the learner in all cardinal directions to the nearest key and lock, four binary parameters indicating if there is an obstacle in the neighboring cells, four binary parameters indicating if there is a key or lock in the neighboring cells, and eight binary parameters indicating if there is a pit in the two adjacent cells in all four directions. Lastly, two integers indicate the number of keys and locks captured so far. We use the dual DQN architecture to train on Key-Lock environments since the domain has a discrete state-action space.

5.1.2 Flags

The second domain contains flags. The agent's task is to capture the flags in an order unknown to it prior to training. Each flag picked up gives a reward of 10 and the reward value increases by an additional 10 for every consecutive flag the agent captures. All other actions including moving into an obstacle receive -10. The learner can only move in cardinal directions and is assumed to have obtained the flag if its location matches the location of the flag. An episode terminates when the agent obtains all the flags, or reaches 100 time steps. A state in the capture-the-flag environment is represented as a vector, including the Euclidean distance from the learner in all cardinal directions to all of the flags, four binary parameters indicating if there is an obstacle in the neighboring cells, and four binary parameters indicating if there is a flag in the neighboring cells. Lastly there is one integer for the number of flags captured so far. We use the dual DQN architecture to train on capture-the-flag environments since the domain has a discrete state-action space.

5.1.3 Parking

The last domain is the parking environment from Leurent [47]. The parking lot consists of 30 parking spots, an agent and a goal item randomly positioned in one of

the spots at the beginning of each episode. The agent always starts in the same position but its initial orientation changes randomly. The agent's task is to reach the goal and orient itself in the right direction. The agent takes two continuous actions: velocity and angular velocity both defined in the range of [-1,1]. We use the actor-critic architecture to train on Highway-Parking environment because the domain has a continuous state-action space. Since the output of the model is continuous in this domain, we assume a normal distribution for the output and use the relative entropy for continuous Gaussian variables in calculating the uncertainty, which has a closed form solution for normal distributions given in Pardo [48]. The environment allows the agent to wander outside of the parking lot, which increases the size of the state space. The agent receives a punishment proportional to its distance to the goal at every step of the training. An episode terminates when the agent attains the goal position with the proper heading or when it reaches 100 time steps. A state in the parking environment is a vector, including the agent's position, velocity, angular velocity, and the goal's position and orientation. The input to the neural network is the concatenation of the state and goal information since the environment uses a goal-aware observation space. We use

5.1.4 Environments for Regressor Training

For the training of the regressor, we use simplified versions of the target tasks. In the case of key-lock and capture-the-flag domains, the regressor learns a 10×10 environment having the same goal characteristics (position and direction) as the original environment. In the parking environment, we reduce the number of parking spots to 8 and halve the initial distance between the agent and the parking spaces. Since *READ-C-SA* requires the training of the regressor for a particular domain once, we only need to construct a single source environment for every domain, which ends up being less time-consuming than manual curricula generation.

For comparison, we also train different regression models on different source tasks to see how the regression training affects the curriculum performance. We use three settings: source task simpler than the target, source task similar to the target and source task same as the target. Source task simpler than the target denotes the 10×10 environment we describe in section 5.1, source task similar to target uses a 20×20 environment with slightly different goal positions than the target and source task same as target uses the target environment in training the regressor for the Key-Lock and Capture the Flag domain. Similarly, source task simpler than target uses half the number of parking spots with half the initial distance to the agent, source task similar to target uses similar number of parking spots to the target task and source task same as target uses the target environment in training the regressor for the Parking domain.

5.2 Comparison Algorithm

We compare *READ-C-TD* and *READ-C-SA* to the method from Narvekar et al. [25], which uses the maximum change in the policy as determined by the number of states in which the action selected by the policy before learning the source task differs from the action selected by the policy after learning the source task. The criteria selects the

task with the highest policy change to the curriculum. We use a set of 15 manually generated for this comparison algorithm.

For additional comparison, we also use a randomly generated curriculum and learning directly on the target task. We generate the random curriculum by sampling start states from the neighborhood of the terminal states.

5.3 Experimental Setup and Hyperparameters

We compare performance based on the total reward obtained while learning the target and curriculum tasks. We also compare the convergence times and provide 95% confidence intervals. All algorithms share the same target task.

There are two types of hyperparameters: 1) neural network (including for the convergence criteria) and 2) curriculum selection. Table 1 contains the parameters used by the neural network in the key-lock environment. These parameters are not tuned for any specific algorithm and all algorithms we use share these parameters for all of the experiments and results in the same domain. ϵ is the ϵ-greediness of the algorithm, which decays during training. α and γ are the learning rate and discount factor respectively. Entropy Reduction is the number of consecutive episodes the convergence criteria considers in determining if entropy is reducing or not.

Table 2 shows the parameters for *READ-C-TD*. Clustering and Cutoff Distance refer to the clustering algorithm used to generate regions and the cutoff point for stopping the merge of clusters. η denotes the number of training episodes on the target task before the curriculum generation. Lastly, Curriculum Length is the number of tasks in the curriculum. For *READ-C-SA*, β is 15,000 and η is 40,000.

Param	Value	Param	Value
ϵ	1	Optimizer	Adamax
ϵ Decay	0.995	Loss	Mean Squared
Minimum ϵ	0.01	Buffer Size	40,000
α	0.005	Batch Size	16
γ	0.99	Entropy Reduction	10

Table 1 Hyperparameters for the neural network.

Parameter	Value	Parameter	Value
Clustering	Agglomerative	Cutoff Distance	3
η	50,000	Curriculum Length	4

Table 2 Hyperparameters for the curriculum learning.
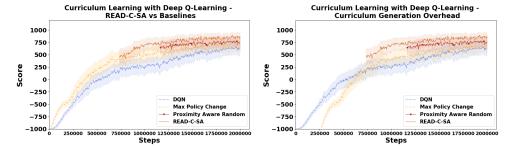
6 Results and Analysis

We perform 25 runs for each algorithm and report the average. All algorithms use the same randomly initialized weights and we alter the DQN weights every run.
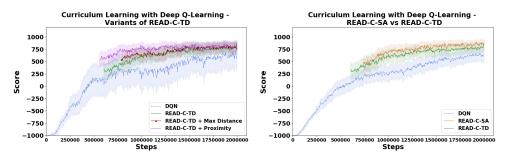
6.1 Results for Key-Lock Domain

Here we present the evaluation results and analysis for the Key-Lock Domain. Without specification, the default training setting for *READ-C-TD* is without any heuristic distance measures. The default training setting for *READ-C-SA* is 200 clusters and the simpler-than-target source task. READ-C-SA with No Clusters directly uses the visited states in curriculum generation. Other than the comparison of performance under different target environment complexity, the default target environment is the 20×20 environment described in section 5.1.

Figure 2(a) contains results for *READ-C-SA* in relation to the Max Policy Change and DQN baselines in the key-lock domain as a function of the number of training steps. Figure 2(b) displays the same results while also counting the curriculum-generation overhead. Only Max Policy Change algorithm has curriculum generation overhead. Following the procedure from the original paper of Max Policy Change [25], we perform 50,000 iterations of training on the target task to generate the prior policy and then perform 5,000 iterations of training on each source task to obtain the posterior policy used to measure the policy change. We repeat this process for two curriculum steps which, in total, results in $2 \times (50,000 + 15 \times 5,000) - 5,000 = 245,000$ iterations of curriculum-generation overhead. *READ-C* and the other baselines, on the other hand, does not require us to perform any training for the curriculum generation. Each graph is offset by the amount of time required to do the curriculum training. Figure 2(c) compares *READ-C-TD*'s different heuristic variants whereas Figure 2(d) shows the performance of *READ-C-SA* with respect to *READ-C-TD* and no curriculum baseline. Figure 2(e) displays the results obtained from training the regressor on data coming from three different source tasks. Figures 3 compares the performance of *READ-C-SA* for different cluster numbers on varying sizes of target environment. Shaded regions are 95% confidence intervals.
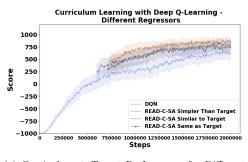
READ-C-SA converges faster than the baseline algorithms even if we do not account for the curriculum-generation overhead but its confidence intervals also show an overlap with max policy change (Figure 2(a)). Accounting for curriculum-generation overhead, the overlap between *READ-C-TD* and Max Policy Change reduces notably indicating performance improvements from *READ-C-SA* (Figure 2(b)). *READ-C-SA* and proximity aware random curricula perform similar to one another but *READ-C-SA* results in much shorter curriculum training (Figure 2(b)). Furthermore, *READ-C-TD* + proximity performs better than *READ-C-TD*, and *READ-C-TD* + max distance, showing that heuristic functions in combination with relative entropy can produce higher performance (Figure 2(c)). *READ-C-TD* performs slightly worse than *READ-C-SA* (Figure 2(d)) mainly due to the fact that a regressor trained on simpler environments generalizes better and captures more relevant parts of the agent's uncertainty compared to a teacher that calculates uncertainty for a larger state space. Training the regressor on a task Similar

(a) Curriculum + Target Performance for *READ-C* and Baselines

(b) Curriculum + Target Performance + Generation Overhead for *READ-C* and Baselines

(c) Curriculum + Target Performance for Variants of *READ-C*

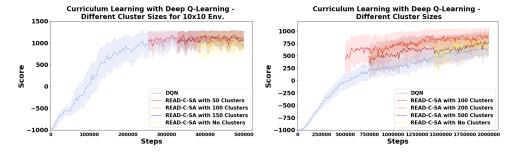(d) Curriculum + Target Performance for *READ-C-TD* and *READ-C-SA*

(e) Curriculum + Target Performance for Different Regressors

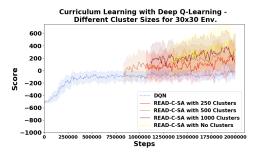**Fig. 2** Performance of the Curriculum-Learning Algorithms as a Function of Training Steps in Key-Lock Domain.

to Target or Same as the Target results in a performance close to *READ-C-TD* slightly below the performance of *READ-C-SA* with Simpler Than Target (Figure 2(e)), which is not unusual since *READ-C-TD* uses the training data on the target task to generate the curriculum as well.

All *READ-C* approaches in a small environment give similar results despite the difference in cluster sizes due to the target environment being simple to learn (Figure 3(a)). Moderate number of clusters such as 100 or 200 performs the best for

(a) Cluster Size Comparison for 10×10 Target Environment

(b) Cluster Size Comparison for 20×20 Target Environment

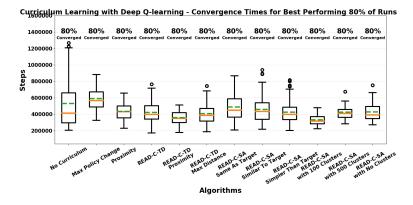(c) Cluster Size Comparison for 30×30 Target Environment

Fig. 3 Effect of Cluster Size on the Performance of the Curriculum-Learning Algorithms in Key-Lock Domain.

medium size environments (Figure 3(b)) while no clustering gives the best results followed closely by 1000 clusters in the large environment although the performance for 500 clusters is very close to 1000 clusters as well (Figure 3(c)). In general, it is safe to say that using a moderate number of clusters always guarantees good performance for *READ-C-SA* on any size of environment even though it might not always produce the best results. Here please note that we use the same set of distance cut-off thresholds ([0.1, 1, 3, 5]) for Agglomerative Clustering in all of these environments but depending on the size of the environment, the number of clusters being generated differs from one figure to the other, which is why the number of clusters is higher in the case of large environment compared to the moderate-size environment (Figures 3(b) & 3(c)).

Convergence results are presented as box plots illustrating the distribution of the number of steps to reach a given cumulative reward across 25 runs (Figure 4). The orange solid line is the median and green dotted line the mean. Each graph represents the total convergence time for curriculum and target training. Runs that do not reach highest reward at any point in the execution are not included in the box plots since there is no convergence time to include for those runs. However, above each box, we also include the percentage of runs that converge to the highest cumulative reward,

(a) The Step Number Where Each Algorithm Reaches A Reward of 900 for 10 Consecutive Episodes

(b) The Step Number Where Best Performing 80% of the Runs Reach A Reward of 900 for 10 Consecutive Episodes

Fig. 4 Box Plots for the Convergence Times of the Algorithms in Key-Lock Domain.

which tells us how likely it is for the runs of an algorithm to reach the highest cumulative reward during execution. If the convergence rate above the graph is low, it means that the given algorithm has a lot of runs that fail to converge.

Figure 4(b) results are selected from the fastest converging 80% of runs so the graph contains the same number of runs for every algorithm.

READ-C-SA with 100 clusters has the lowest variance in the convergence times and highest rate of convergence (Figures 4(a) & 4(b)). *READ-C-TD* and its variants generally result in better mean and median convergence times than the baseline algorithms whereas *READ-C-SA* approaches generate varying performance results depending on how many clusters are used for the curriculum generation. *READ-C-SA* with 100 clusters using the simpler source environment produces the lowest mean convergence time followed closely by *READ-C-TD* + proximity. *READ-C-SA* with 100 clusters also shows only a slightly higher convergence rate than *READ-C-TD* and its variants arguably making it the best converging variant of *READ-C* (Figure 4(a)). On
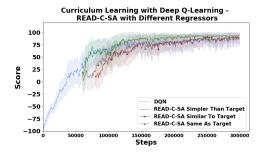
the other hand, *READ-C-SA* with source tasks similar to target and same as target show worse mean and median performance than *READ-C-TD* and proximity aware baseline. The problem in these approaches stem from the regressor training on larger state spaces with higher variance and error rates, which consequently harms the performance on the target environment and shows the importance of using a suitable small environment in modeling the uncertainty of the agent in a particular domain. If we only look at 80% of the runs, the difference between median and mean performance clearly reduces for many of the best performing *READ-C* algorithms, indicating that the outliers are influencing the mean convergence times being higher than medians (Figure 4(b)). *READ-C-SA* with 100 clusters still performs the best when looking at 80% of the runs and the relative ordering of the algorithms mostly remains the same as before.

6.2 Results for Capture-the-Flag Domain

Here we provide the evaluation results and analysis for Capture-the-Flag domain. The default cluster size for *READ-C-SA* is 150. Other default settings are the same as results for Key-Lock domain.
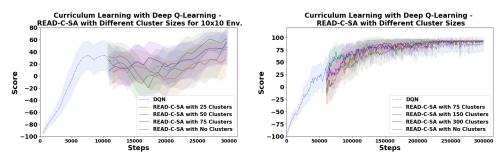

(a) Curriculum + Target Performance for *READ-C* and Baselines
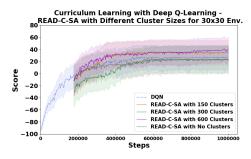
(b) Curriculum + Target Performance for Variants of *READ-C*

(c) Curriculum + Target Performance for Different Regressors

Fig. 5 Performance of the Curriculum-Learning Algorithms as a Function of Training Steps in Capture-the-Flag Domain.

26

Figure 5(a) shows the comparison of *READ-C-SA*, *READ-C-TD* and no curriculum baseline. Figure 5(b) shows the variants of *READ-C-TD* and Figure 5(c) displays different regressors trained on a 10×10 environment, a 20×20 environment similar to target task and the target environment respectively, on the capture-the-flag domain. Figure 6(a) , Figure 6(b) & Figure 6(c) display the effect of different cluster numbers on the performance of *READ-C-SA* using different target environment sizes. Finally, we present the box plots for convergence times of *READ-C* approaches on the capture-the-flag domain illustrating the distribution of the number of steps required to reach a given cumulative reward for a given number of consecutive episodes similar to the key-lock domain (Figure 7(a) & Figure 7(b)). Shaded regions show 95% confidence intervals.
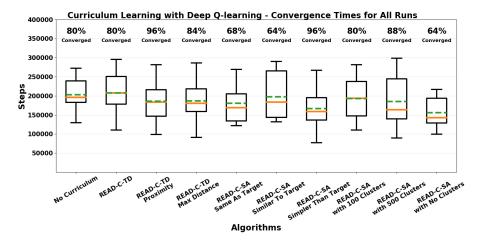

(a) Cluster Size Comparison for 10×10 Target Environment

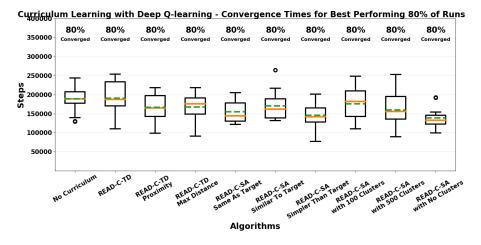(b) Cluster Size Comparison for 20×20 Target Environment

(c) Cluster Size Comparison for 30×30 Target Environment

Fig. 6 Effect of Cluster Size on the Performance of the Curriculum-Learning Algorithms in Capture-the-Flag Domain.

READ-C-SA converges faster than the no curriculum baseline and *READ-C-TD* (Figure 5(a)). *READ-C-TD* + proximity performs better than other *READ-C-TD* variants (Figure 5(b)). Using a simpler source environment for the regressor results in better performance compared to other types of regressors (Figure 5(c)). The results are mainly consistent with what we have seen on the key-lock domain and further supports the evidence seen in the prior graphs. When it comes to the different cluster numbers, there does not emerge a meaningful difference in the performance of the

(a) The Step Number Where Each Algorithm Reaches A Reward of 84 for 500 Consecutive Episodes

(b) The Step Number Where Best Performing 80% of the Runs Reach A Reward of 75 for 500 Consecutive Episodes

Fig. 7 Box Plots for the Convergence Times of the Algorithms in Capture-the-Flag Domain.

algorithms in the small environment due to the environment taking little time to train despite the moderate cluster sizes reaching a slightly higher convergence point than the other options (Figure 6(a)). For the moderate-size environment, using a moderate number of clusters such as 150 seems to result in the best performance although most of the cluster sizes show similar results in this domain (Figure 6(b)). For the large-size environment, a moderate number of clusters between 159 and 600 again produces good performance similar to the key-lock domain (Figure 6(c)). The choice of cluster numbers does not appear to have a large influence on the performance for capture-the-flag domain regardless of the target environment size and using any number of

28

clusters in this experiment results in more or less slightly better outcome than the baseline approaches (Figures 6(b)& 6(c)).
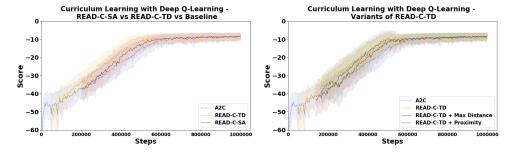
READ-C-SA with no clusters shows the lowest variance in the convergence times and produces the best mean and median performance while having a very low convergence rate meaning that many of its runs are classified as outliers (Figure 7). *READ-C-SA* with source task simpler than the target task gives the second best mean and median performance while also having the highest convergence rate (Figure 7(a)), arguably making it the best performing algorithm in this domain. *READ-C-SA* with source task similar to the target task and same as the target task result in poorest convergence rates (Figure 7(a)) and along with the results from the key-lock domain, prove that to obtain good performance on *READ-C-SA*, it is necessary to train the regressor on a simpler environment showing similarities to the target task. *READ-C-SA* with 500 clusters shows good median and mean performance similar to *READ-C-SA* with 200 clusters but it displays a higher variance among its runs indicating that using a too high number of clusters could make the algorithm more sensitive to the random weight initialization (Figure 7(a)). If we only look at 80% of the runs, the variance among multiple executions clearly reduces for many of the best performing *READ-C* algorithms (Figure 7(b)). Overall, the results for capture-the-flag domain show less variance (when accounted for the scale of the y-axis) and less performance differences among different *READ-C* approaches compared to the key-lock domain, partially being caused by less-complex structure of the environment characteristics (Figure 7(a) & Figure 7(b)).
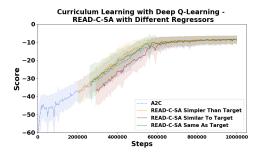
6.3 Results for Parking Domain

Here we provide the evaluation results and analysis for the Parking domain. The cluster size remains the same for all *READ-C-SA* configurations. Other default settings are the same as the results from the grid environments.

Figure 8(a) contains the comparison of *READ-C-SA*, *READ-C-TD* and advantage actor-critic algorithm [49] for the parking domain specified in Leurent [47], Figure 8(b) shows the variants of *READ-C-TD* and Figure 8(c) displays different regressors used for *READ-C-SA*. Figure 8 performs 25 runs of each algorithm similar to before. We do not present the results for different numbers of clusters as it does not add any additional significant findings to our results. Finally, we present the box plots showing the convergence times of each algorithm for a given cumulative reward value for 25 runs (Figure 9).

READ-C-TD performs better than *READ-C-SA* and A2C in the parking environment while *READ-C-SA* still shows slight performance improvements over the baseline (Figure 8(a)). The main reason for this difference between the parking and the grid world environments is that since the parking domain contains continuous state and action spaces, and *READ-C* uses relative entropy for continuous variables, the error rate for the GBM regressor ends up being higher than before causing *READ-C-SA* to fall behind *READ-C-TD* in certain cases. *READ-C-TD* also performs better than its variants in this domain although *READ-C-TD* + proximity almost gives the same performance results as *READ-C-TD* (Figure 8(b)). Most of the algorithms display overlapping confidence intervals but the confidence intervals for the baseline algorithm

29

(a) Curriculum + Target Performance for *READ-C* and Baselines

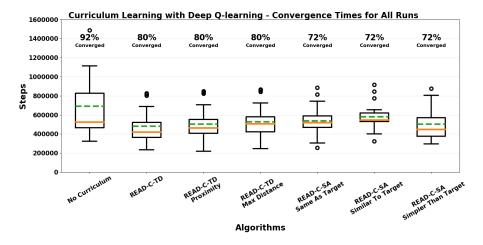(b) Curriculum + Target Performance for Variants of *READ-C*

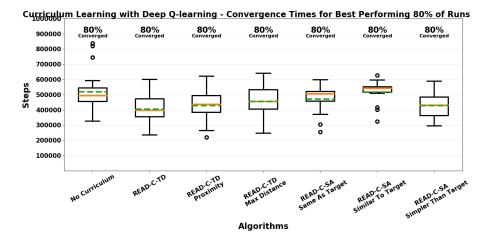(c) Curriculum + Target Performance for Different Regressors

Fig. 8 Performance of the Curriculum-Learning Algorithms as a Function of Training Steps in the Parking Domain.

still remains significantly below the variants of *READ-C* showing that *READ-C* is able to provide improvement in learning efficiency for both continuous and discrete domains (Figure 8(b)). *READ-C-SA* with source task simpler than the target task shows better performance than the other two competing regressors showing consistent results with the grid world domains (Figure 8(c)). Although the gap between *READ-C* and the baseline algorithm is narrower in these results, there still exists a benefit of using *READ-C* in the continuous domain environments since most of *READ-C* variants consistently manage to remain above the benchmark criteria aside from *READ-C-SA* with source task similar to the target task.

For the convergence times, all *READ-C* approaches show better mean and median performance than the A2C algorithm but they also produce worse convergence rates (Figure 9(a)). The lower convergence rates in this case does not mean that the agent is not learning the task since we have seen in the line graphs that the asymptotic performance of the agent was better than the baseline algorithm but rather, the lower convergence rates indicate that the agent in some runs is not able to reach the reward value of -15 for 10 consecutive episodes despite improving its policy performance. However, if we look at the results obtained using the best performing 80% runs, we see that all *READ-C* algorithms, except READ-SA with source task similar to the target task, still produce better mean and median values while having the same convergence

(a) The Step Number Where Each Algorithm Reaches A Reward of -15 for 10 Consecutive Episodes

(b) The Step Number Where Best Performing 80% of the Runs Reach A Reward of -20 for 10 Consecutive Episodes

Fig. 9 Box Plots for the Convergence Times of the Algorithms in Parking Domain.

rate (Figure 9(b)), meaning that the reason *READ-C* shows low convergence rate in the first case is because of the 20% poor performing runs. Overall, *READ-C-SA* manages to outperform its competition albeit it cannot guarantee convergence for all cases. *READ-C-TD* shows better convergence rates than its variants and *READ-C-SA*, which is consistent with the results of the line graphs (Figure 9(a)). Also, *READ-C-SA* with simpler source task than the target task obtains a better mean and median convergence rate than *READ-C-SA* with source task same as the target task and *READ-C-SA* with source task similar to the target task, which is consistent with the results we obtained in the Key-Lock domain (Figure 9(b)).

31

7 Conclusion and Future Work

We presented *READ-C*, a novel approach to using relative entropy for automatic curriculum generation. We detailed two versions of the approach, one using a teacher to measure relative entropy and another self-calculating the relative entropy without the teacher. We adapted a curriculum generation criteria based on max policy change from the literature to compare against *READ-C* and evaluated performance on the key-lock domain, capture-the-flag domain and the parking domain using discrete and continuous action and state spaces. Our results showed that *READ-C* was able to outperform a proximity-aware randomly generated curriculum, the comparison criteria, and learning the target task directly in all our experiments. Further, *READ-C-SA* produced better results than the baselines and *READ-C-TD* in most cases even without having access to a teacher model.

Declarations

Funding

The authors did not receive support from any organization for the submitted work.

Conflict of interest/Competing interests

The authors have no relevant financial or non-financial interests to disclose.

Ethics approval and consent to participate

There are no ethical considerations which we feel must be specifically highlighted here.

Consent for publication

All authors consent to the publication of the materials presented in this paper.

Materials Availability

Not applicable

Data and Code availability

All relevant resources will be made available on Github upon publication.

Author contribution

Methodology: Muhammed Yusuf Satici; Formal analysis and investigation: Muhammed Yusuf Satici; Writing - original draft preparation: Muhammed Yusuf Satici; Writing - review and editing: Muhammed Yusuf Satici, Jianxun Wang, David Roberts; Supervision: David Roberts.

References

[1] Forbes-Riley, K., Litman, D.: Adapting to student uncertainty improves tutoring dialogues, vol. 200, pp. 33–40 (2009). https://doi.org/10.3233/978-1-60750-028-5-33

[2] Pon-Barry, H., Schultz, K., Bratt, E., Clark, B., Peters, S.: Responding to student uncertainty in spoken tutorial dialogue systems. I. J. Artificial Intelligence in Education **16**, 171–194 (2006)

[3] Forbes-Riley, K., Litman, D.: Designing and evaluating a wizarded uncertainty-adaptive spoken dialogue tutoring system. Comput. Speech Lang. **25**(1), 105–126 (2011) https://doi.org/10.1016/j.csl.2009.12.002

[4] Jog, V., Loh, P.-L.: Teaching and learning in uncertainty. IEEE Transactions on Information Theory **67**(1), 598–615 (2021) https://doi.org/10.1109/TIT.2020.3030842

[5] Vygotsky, L.S.: Mind and Society: The Development of Higher Mental Processes. Harvard University Press, Cambridge, MA (1978). http://www.learning-theories.com/vygotskys-social-learning-theory.html

[6] Stuyf, R.R.V.D.: Scaffolding as a teaching strategy. In: Adolescent Learning and Development (2002)

[7] Settles, B.: Active learning literature survey. (2009)

[8] Settles, B., Craven, M.W.: An analysis of active learning strategies for sequence labeling tasks. In: EMNLP (2008)

[9] Khan, F., Mutlu, B., Zhu, J.: How do humans teach: On curriculum learning and teaching dimension. In: Shawe-Taylor, J., Zemel, R.S., Bartlett, P.L., Pereira, F., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 24, pp. 1449–1457. Curran Associates, Inc., ??? (2011). http://papers.nips.cc/paper/4466-how-do-humans-teach-on-curriculum-learning-and-teaching-dimension.pdf

[10] Peng, B., MacGlashan, J., Loftin, R., Littman, M.L., Roberts, D.L., Taylor, M.E.: Curriculum design for machine learners in sequential decision tasks. IEEE Transactions on Emerging Topics in Computational Intelligence **2**(4), 268–277 (2018) https://doi.org/10.1109/TETCI.2018.2829980

[11] MacAlpine, P., Stone, P.: Overlapping layered learning. Artificial Intelligence **254**, 21–43 (2018) https://doi.org/10.1016/j.artint.2017.09.001

[12] Ren, Z., Dong, D., Li, H., Chen, C.: Self-paced prioritized curriculum learning

with coverage penalty in deep reinforcement learning. IEEE Transactions on Neural Networks and Learning Systems **PP**, 1–11 (2018) https://doi.org/10.1109/TNNLS.2018.2790981

[13] Seita, D., Chan, D., Rao, R., Tang, C., Zhao, M., Canny, J.: ZPD Teaching Strategies for Deep Reinforcement Learning from Demonstrations (2019)

[14] Florensa, C., Held, D., Wulfmeier, M., Abbeel, P.: Reverse curriculum generation for reinforcement learning. CoRR **abs/1707.05300** (2017) arXiv:1707.05300

[15] Zhang, Y., Abbeel, P., Pinto, L.: Automatic Curriculum Learning through Value Disagreement (2020)

[16] Klink, P., D'Eramo, C., Peters, J., Pajarinen, J.: Self-Paced Deep Reinforcement Learning (2020)

[17] Portelas, R., Colas, C., Hofmann, K., Oudeyer, P.-Y.: Teacher algorithms for curriculum learning of Deep RL in continuously parameterized environments (2019)

[18] Huang, P., Xu, M., Zhu, J., Shi, L., Fang, F., ZHAO, D.: Curriculum reinforcement learning using optimal transport via gradual domain adaptation. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems, vol. 35, pp. 10656–10670. Curran Associates, Inc., ??? (2022). https://proceedings.neurips.cc/paper_files/paper/2022/file/4556f5398bd2c61bd7500e306b4e560a-Paper-Conference.pdf

[19] Klink, P., Yang, H., D'Eramo, C., Peters, J., Pajarinen, J.: Curriculum reinforcement learning via constrained optimal transport. In: Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., Sabato, S. (eds.) Proceedings of the 39th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 162, pp. 11341–11358. PMLR, ??? (2022). https://proceedings.mlr.press/v162/klink22a.html

[20] Wu, B., Gupta, J.K., Kochenderfer, M.: Model primitives for hierarchical lifelong reinforcement learning. Auton Agent Multi-Agent Syst **34**(28) (2020) https://doi.org/10.1007/s10458-020-09451-0

[21] Silva, F.L., Costa, A.H.R.: Object-oriented curriculum generation for reinforcement learning. In: André, E., Koenig, S., Dastani, M., Sukthankar, G. (eds.) Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018, pp. 1026–1034. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, ??? (2018). http://dl.acm.org/citation.cfm?id=3237850

[22] Svetlik, M., Leonetti, M., Sinapov, J., Shah, R., Walker, N., Stone, P.: Automatic

curriculum graph generation for reinforcement learning agents. In: AAAI (2017)

[23] Qiao, Z., Mülling, K., Dolan, J., Palanisamy, P., Mudalige, P.: Automatically generated curriculum based reinforcement learning for autonomous vehicles in urban environment. (2018). https://doi.org/10.1109/IVS.2018.8500603

[24] Matiisen, T., Oliver, A., Cohen, T., Schulman, J.: Teacher-Student Curriculum Learning (2017)

[25] Narvekar, S., Sinapov, J., Stone, P.: Autonomous task sequencing for customized curriculum design in reinforcement learning. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence. IJCAI'17, pp. 2536–2542. AAAI Press, ??? (2017). http://dl.acm.org/citation.cfm?id=3172077.3172241

[26] Wöhlke, J., Schmitt, F., Hoof, H.: A performance-based start state curriculum framework for reinforcement learning. AAMAS '20, pp. 1503–1511. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2020)

[27] Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M.E., Stone, P.: Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey (2020)

[28] Madhawa, K., Murata, T.: Active learning for node classification: An evaluation. Entropy **22**(10) (2020) https://doi.org/10.3390/e22101164

[29] Bougie, N., Ichise, R.: Goal-driven active learning. Auton Agent Multi-Agent Syst **35**(44) (2021) https://doi.org/10.1007/s10458-021-09527-5

[30] Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence **112**(1), 181–211 (1999) https://doi.org/10.1016/S0004-3702(99)00052-1

[31] Stolle, M., Precup, D.: Learning options in reinforcement learning, vol. 2371, pp. 212–223 (2002). https://doi.org/10.1007/3-540-45622-8_16

[32] Mankowitz, D.J., Mann, T.A., Mannor, S.: Adaptive Skills, Adaptive Partitions (ASAP) (2016)

[33] Kearns, M., Mansour, Y., Ng, A.Y.: A sparse sampling algorithm for near-optimal planning in large markov decision processes. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2. IJCAI'99, pp. 1324–1331. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999)

[34] Harb, J., Bacon, P.-L., Klissarov, M., Precup, D.: When Waiting is not an Option : Learning Options with a Deliberation Cost (2017)

[35] Bacon, P.-L., Harb, J., Precup, D.: The Option-Critic Architecture (2016)

[36] Harutyunyan, A., Dabney, W., Borsa, D., Heess, N., Munos, R., Precup, D.: The Termination Critic (2019)

[37] Sutton, R.S., Barto, A.G.: Reinforcement Learning - an Introduction, 2nd edn. Adaptive computation and machine learning. MIT Press, Cambridge, MA (1998). http://www.worldcat.org/oclc/37293240

[38] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)

[39] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning (2016)

[40] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

[41] Ward, J.H.: Hierarchical grouping to optimize an objective function. Journal of the American Statistical Association **58**(301), 236–244 (1963)

[42] Karmakar, P., Bhatnagar, S.: Two Timescale Stochastic Approximation with Controlled Markov noise and Off-policy temporal difference learning (2017)

[43] Wu, Y., Zhang, W., Xu, P., Gu, Q.: A Finite Time Analysis of Two Time-Scale Actor Critic Methods (2022)

[44] Holzleitner, M., Gruber, L., Arjona-Medina, J., Brandstetter, J., Hochreiter, S.: Convergence Proof for Actor-Critic Methods Applied to PPO and RUDDER (2020)

[45] Nesterov, Y.: Lectures on Convex Optimization, 2nd edn. Springer, Switzerland AG (2018)

[46] Konidaris, G., Barto, A.: Building portable options: Skill transfer in reinforcement learning. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 895–900 (2007)

[47] Leurent, E.: An Environment for Autonomous Driving Decision-Making. GitHub (2018)

[48] Pardo, L.: Statistical inference based on divergence measures. (2005). https://api.semanticscholar.org/CorpusID:116968026

[49] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. CoRR **abs/1602.01783** (2016) 1602.01783