Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning

Sanmit Narvekar, Jivko Sinapov, and Peter Stone
Department of Computer Science, University of Texas at Austin
{sanmit, jsinapov, pstone}@cs.utexas.edu

Abstract

Transfer learning is a method where an agent reuses knowledge learned in a source task to improve learning on a target task. Recent work has shown that transfer learning can be extended to the idea of *curriculum learning*, where the agent incrementally accumulates knowledge over a *sequence* of tasks (i.e. a curriculum). In most existing work, such curricula have been constructed manually. Furthermore, they are fixed ahead of time, and do not adapt to the progress or abilities of the agent. In this paper, we formulate the design of a curriculum as a Markov Decision Process, which directly models the accumulation of knowledge as an agent interacts with tasks, and propose a method that approximates an execution of an optimal policy in this MDP to produce an agent-specific curriculum. We use our approach to automatically sequence tasks for 3 agents with varying sensing and action capabilities in an experimental domain, and show that our method produces curricula customized for each agent that improve performance relative to learning from scratch or using a different agent's curriculum.

1 Introduction

As reinforcement learning (RL) agents are challenged to learn increasingly complex tasks, some of these tasks may be infeasible to learn directly. Various transfer learning methods and frameworks have been proposed that allow an agent to better learn a difficult *target* task by levering knowledge gained in one or more *source* tasks [Taylor and Stone, 2009; Lazaric, 2011]. Recently, these ideas have been extended to the problem of *curriculum learning*, where the goal is to design a curriculum consisting of a *sequence* of training tasks that are learned by the agent prior to learning the target task.

There are two main limitations of most methods that employ curricula in reinforcement learning settings. First, the sequencing of source tasks is typically done manually by a domain expert or human users. Second, in cases where the sequencing is done automatically, the curriculum created does not take into account the agent's abilities or experience. As a result, the same curriculum is produced even for very different agents. This curriculum can be suboptimal when different agents have not only different learning abilities, but also different perceptions and actions.

To address these limitations, this paper presents a method for curriculum generation that adapts to the progress and abilities of a particular agent. We formalize the design of a curriculum as a Markov Decision Process (MDP) that explicitly models the accumulation of knowledge by the agent as it learns through a sequence of tasks. As such, it also models the cost of learning a task relative to the current ability of the agent. We show how a policy in this *curriculum* MDP produces a curriculum, and propose a recursive, Monte-Carlo algorithm that approximates an execution of an optimal policy in this MDP. The proposed method was evaluated by producing curricula for 3 heterogeneous agents in a grid-world domain. Our experiments show that the resulting curricula enabled the agents to reduce the overall training time needed to learn the target task compared to learning from scratch. More importantly, the experiments show that the resulting curricula were each tailored to the sensory and action abilities of each agent, and that using a curricula designed for one agent to train another can be detrimental in practice.

2 Background

In this section, we review background material and how it fits into our curriculum design framework.

2.1 Markov Decision Processes

We model the decision-making process of an agent as an episodic Markov Decision Process (MDP). MDPs are used at two different levels in this paper - one for modeling individual tasks, and one for modeling the task sequencing process. We introduce the formulation for individual tasks here.

An episodic MDP M is a 6-tuple $(\mathcal{S}, \mathcal{A}, p, r, S_0, S_f)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $p(s, a, s')$ is a transition function that gives the probability of transitioning to state s' after taking action a in state s, and $r(s, a)$ is a reward function that gives the immediate reward for taking action a in state s. S_0 is the initial state distribution, and S_f is the set of terminal states.

At each time step t, the agent observes its state and chooses an action according to its *policy* $\pi : \mathcal{S} \mapsto \mathcal{A}$. The goal of the agent is to learn an *optimal policy* π^*, which maximizes its expected *return* G_t until the episode ends at timestep T:

$$G_t = \sum_{i=1}^{T-t} R_{t+i} \tag{1}$$

One common way to do this is to first learn the optimal action-value function $Q^*(s,a)$, which gives the expected return for taking action a in state s and following π^* thereafter, using an algorithm such as SARSA or Q-learning [Sutton and Barto, 1998]. The optimal policy then selects action $\arg\max_a Q^*(s,a)$ in each state.

2.2 Transfer Learning

Curriculum learning builds on the assumption that learning one task can speed up learning on another, a concept that's come to be known as *transfer learning* [Taylor and Stone, 2009; Lazaric, 2011]. In transfer learning, instead of learning directly on a *target task* MDP, the agent first trains on one or more *source task* MDPs, and transfers the knowledge gained to the target. In this paper, we transfer information between MDPs using value function transfer [Taylor *et al.*, 2007], which uses the parameters of an action-value function $Q_s(s,a)$ learned in a source task to initialize the action-value function in the target task $Q_t(s,a)$.

One way to quantify the benefit of transfer is by the *time to threshold* metric [Taylor and Stone, 2009], which computes how much faster an agent can learn a policy that achieves return $G_0 \geq \delta$ on the target task if it transfers knowledge, as opposed to learning the target from scratch. In this work, we will measure *time* using the number of actions taken.

2.3 Curriculum Learning

Curriculum learning is an extension of tranfer learning, where the goal is to automatically design and choose a sequence of tasks (i.e. a *curriculum*) $M_1, M_2, \ldots M_t$ for an agent to train on, such that learning speed or performance on a target task M_t is improved. In this paper, we leverage previous work by [Narvekar *et al.*, 2016] to dynamically generate source tasks for use in a curriculum. Narvekar et al. proposed a series of heuristic functions $f : M_t \times X \mapsto M_s$ that take as input a target task M_t and the agent's current experience trajectories X on M_t, in order to produce a source task M_s. Each function does this by varying different aspects of the MDP M_t to create tasks that are relevant to solving M_t, but are easier to learn. For example, they reduce the state and action spaces, alter the transition or reward dynamics, or modify the initial or terminal state distributions. The focus of this paper will be on how to select and sequence tasks from this space.

3 Curriculum Generation as an MDP

MDPs are used at two different levels in this work. One is the standard use, where a *learning agent* interacts with and tries to learn a *task* modeled as an MDP M. Our main contribution is a second, higher level MDP for the *curriculum design agent*, whose goal is to sequence tasks M for the learning agent. We will refer to MDPs that the learning agent interacts with as tasks, and the MDP that the curriculum agent interacts with as the curriculum MDP (CMDP).

3.1 Curriculum MDP (CMDP)

The overall process in a CMDP unfolds as follows: the learning agent starts with some initial policy π_0, which is represented as the initial state S_0 of the CMDP. The curriculum agent then selects an action A_0, where each action corresponds to a different task that can be learned by the learning agent using learning algorithm $\mathcal{L}$. Learning a task transforms the learning agent's policy to a new policy π_1, represented in the CMDP as the next state S_1, by means of transfer learning algorithm $\mathcal{T}$. It also incurs a cost, which is the amount of time needed by the learning agent to learn the task. This process repeats until the curriculum agent transitions to a terminal state, which is a state where the policy of the learning agent can achieve a return $G_0 \geq \delta$ on the target task.

We now define this process formally as an MDP. To distinguish the curriculum MDP from task MDPs, we will use the superscript C to refer to elements of the curriculum MDP.

Definition 1: A *curriculum MDP* M^C is a 6-tuple $(\mathcal{S}^C, \mathcal{A}^C, p^C, r^C, S_0^C, S_f^C)$, where:

State Space The set of states $\mathcal{S}^C$ consist of the set of all policies the learning agent can represent. Each state represents a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$, which determines how the learning agent will act in the target task. For example, the initial state S_0^C could be the uniform random policy. The terminal states S_f^C are defined as states whose policies achieve a return of at least δ on the target task. In this section, we leave open how the policy is represented. Common options are to directly encode a parameterized policy, or to use a policy derived from a value function.

Action Space The set of actions $\mathcal{A}^C$, are the different tasks a learning agent can train on. Each action $a^C \in \mathcal{A}^C$ maps to a task M, and taking an action corresponds to training the learning agent on the corresponding task until convergence. For example, not using a curriculum and training directly on the target task corresponds to taking the target task action from the initial state.

Transition Function Executing an action $a^C \in \mathcal{A}^C$ in state $s^C \in \mathcal{S}^C$ transitions the CMDP to a new state $s'^C \in \mathcal{S}^C$ according to the transition function $p^C(s^C, a^C, s'^C)$. In other words, the transition function describes how the learning agent's policy changes as a result of learning a task. Learning a task can lead to different policies depending on the initial policy before learning.

Reward Function The goal of the curriculum agent is to minimize the amount of time needed by the learning agent to reach a policy that gives a return of at least δ on the target task (i.e. minimize the *time to threshold*, where we count the total time needed, including time spent in all source tasks). We can encode this in the reward function by defining $r^C(s^C, a^C)$ to be the negative of the expected time needed to learn task a^C starting from policy s^C. Thus, $r^C(s^C, a^C)$ is the cost of learning a^C starting from policy s^C.

The base learning algorithm $\mathcal{L}$ of the learning agent and the transfer algorithm $\mathcal{T}$ are implicitly encoded in the transition and reward functions of the CMDP, since they affect the set of states explored and the efficiency with which a task is solved. As such, they must be specified in any implementation. In addition, note that the cost of learning a task a^C is dependent on the current policy state s^C the agent is in. Starting with a good policy can make it easier to learn a task, leading to a low cost, whereas starting from a worse policy incurs higher cost. Our goal is to find some sequence of tasks $a_0^C, a_1^C, \dots a_t^C$ such that the total cost $\sum_{i=0}^{t} r^C(s_i^C, a_i^C)$ to achieve return δ on the target task is less than the cost incurred by learning directly on the target task $r^C(s_0^C, a_t^C)$.

A policy $\pi^C : \mathcal{S}^C \mapsto \mathcal{A}^C$ on a CMDP specifies which task to train on given a learning agent policy π. Thus, we can *execute* π^C for a particular agent to produce a curriculum. Due to stochasiticity during the learning process, an execution of π^C for a single agent could result in many different curricula. Learning the optimal policy π^{C^*} in a curriculum MDP thus produces execution traces of optimal curricula.

3.2 Discussion

In theory, now that we have posed the curriculum design problem as an MDP, we could apply any of the standard reinforcement learning algorithms to solve for the optimal CMDP policy π^C. For example, in some domains, we may have a model of the transition dynamics of the CMDP, or be able to approximate it using heuristics and domain knowledge. In this case, we could use dynamic programming to directly solve for the optimal curriculum.

If the model is not known, model-free methods such as SARSA or Q-learning could be used. However, model free methods typically suffer from high sample complexity, and this issue is exacerbated in CMDPs since taking an action requires solving an entire task MDP. Instead, we propose to directly find one particular execution of π^C that is representative of what would be expected when drawing a single execution from an optimal policy π^{C^*}.

4 Monte Carlo Approximation

In this section, we describe a recursive Monte-Carlo algorithm that iteratively builds an execution trace of π^{C^*}. The intuition for our approach is as follows: assume the learning agent starts with some initial policy π_0, corresponding to initial state S_0^C in the CMDP. Our goal is to learn a policy $\pi_f : \mathcal{S} \mapsto \mathcal{A}$ that achieves return δ on the target task M_t as quickly as possible. Although we don't know what a terminal state policy π_f looks like, we can identify what parts of the state space $\mathcal{S}$ are relevant to an optimal policy for the target task, because we can *sample* state trajectories from the target task by executing the target task action. We can then use those samples to guide the selection of a source task.

Thus, the main idea behind our algorithm is to incrementally build up the policy using states and experiences the agent is currently facing. The algorithm samples from the target task to figure out what the learning agent needs to learn about. It then creates a set of sources tailored for those experiences using the heuristic functions defined by [Narvekar *et al.*, 2016], recursively breaking down the tasks if they are too *difficult* for the agent to solve. The task which as a result of learning changes the policy the most according to the target samples is then selected. Learning the task updates the learning agent's policy, corresponding to a new state in the CMDP. This in turn leads to a different set of samples from the target task. This process repeats until the agent is able to solve the target directly. Note that the goal is not necessarily to learn a policy π for every state in the state space $\mathcal{S}$ of a target task M_t, as some of these states may not be encountered by the agent, and hence are irrelevant for executing the optimal policy π^*.

Algorithm 1 GENERATECURRICULUM($M_t, \pi, \beta, \delta, \epsilon$)

```
$\mathcal{C} \leftarrow \emptyset$
while true do
    size = $|\mathcal{C}|$
    $(\pi', \mathcal{C}) \leftarrow$ RECURSETASKSELECT($M_t, \pi, \beta, \epsilon, \mathcal{C}$)
    if $\pi' =$ null then
        Increase $\beta$
        POP($\mathcal{C}, |\mathcal{C}| -$ size)
        continue
    end if
    $\pi \leftarrow \pi'$
    if EVALUATE($M_t, \pi$) $\geq \delta$ then
        break
    end if
end while
return $(\pi, \mathcal{C})$
```

The *difficulty* of a task can be quantified by the amount of time needed to solve the task (i.e. the cost of learning a task $r^C(s^C, a^C)$). As stated in Section 3, the cost depends on many different factors, such as the policy the learning agent starts with, the learning algorithm being used, and also aspects of the task itself such as the size of its state and action spaces. The only way to determine the true cost is to solve the task, which is unbounded and unknown ahead of time. Therefore, we introduce the idea of a budget or learning capacity β for the agent, which limits the amount of time an agent will spend trying to learn a task before it decides the task is too difficult. This promotes learning easier tasks first, and tackling harder tasks once the learning agent has accumulated knowledge from these easy tasks. Finally, to prevent unnecessary tasks from being used in the curriculum, we require tasks to affect the policy and be relevant to the target task by at least fraction ϵ (described in detail in the next section).

4.1 Algorithm Details

We now formalize the intuition given into pseudocode. The main call is to Algorithm 1, GENERATECURRICULUM, which takes as input the target task M_t that we want to generate a curriculum for, the learning agent's initial policy π (i.e. S_0^C, typically a uniform random policy), the learning budget β, the return threshold δ desired on the target task, and the minimum policy change and relevance parameter ϵ. It returns a policy π (i.e. a terminal state S_f^C) that can achieve a return of δ on M_t, and the curriculum $\mathcal{C}$.

Each iteration of the loop in Algorithm 1 attempts to add a task to the curriculum by calling RECURSETASKSELECT, and corresponds to a transition in the CMDP. If a task is found and added to the curriculum, the updated policy π is evaluated on the target task. The loop terminates if a return greater than δ is received. If no tasks are found, the budget β is increased, any tasks that were added in this phase are cleared, and the search is repeated.

Algorithm 2, RECURSETASKSELECT, is the core method

Algorithm 2 RECURSETASKSELECT$(M, \pi, \beta, \epsilon, \mathcal{C})$

```
1:  (solved, $X, \pi'$) = LEARN$(M, \pi, \beta)$
2:  if solved then
3:      ENQUEUE$(\mathcal{C}, M)$
4:      return $(\pi', \mathcal{C})$
5:  end if
6:  $\mathcal{M}_s \leftarrow$ CREATESOURCETASKS$(M, X)$
7:  $\mathcal{P} \leftarrow \emptyset$
8:  $\mathcal{U} \leftarrow \emptyset$
9:  for $M_s \in \mathcal{M}_s$ do
10:     (solved$_{M_s}$, $X_{M_s}, \pi_{M_s}$) = LEARN$(M_s, \pi, \beta)$
11:     if solved$_{M_s}$ then
12:         $\mathcal{P} \leftarrow \mathcal{P} \cup \{(\pi_{M_s}, M_s)\}$
13:     else
14:         $\mathcal{U} \leftarrow \mathcal{U} \cup \{(M_s, X_{M_s})\}$
15:     end if
16: end for
17: if $|\mathcal{P}| > 0$ then
18:     ($\pi_{\text{best}}, M_{\text{best}}$, score) = GETBESTPOLICY$(\mathcal{P}, \pi, X)$
19:     if score $> \epsilon$ then
20:         ENQUEUE$(\mathcal{C}, M_{\text{best}})$
21:         return $(\pi_{\text{best}}, \mathcal{C})$
22:     end if
23: end if
24: SORTBYSAMPLERELEVANCE$(\mathcal{U}, X, \epsilon)$
25: for $(M_s, X_{M_s}) \in \mathcal{U}$ do
26:     $(\pi'_s, \mathcal{C}) \leftarrow$ RECURSETASKSELECT$(M_s, \pi, \beta, \epsilon, \mathcal{C})$
27:     if $\pi'_s \neq$ null then
28:         return RECURSETASKSELECT$(M, \pi'_s, \beta, \epsilon, \mathcal{C})$
29:     end if
30: end for
31: return ( null, $\mathcal{C}$)
```

that adds tasks to the curriculum and updates the policy π. It starts by calling LEARN, which attempts to *solve* the given task M starting with initial policy π, using at most β time steps. LEARN returns a boolean *solved* indicating whether the task was solved or not, a set of state-action-reward samples X for each trajectory experienced, and the updated policy π' as a result of learning M.

We propose two methods for determining whether a task has been solved. The first is *policy convergence*, which checks whether the policy has converged (i.e. not changed) for the states the agent has visited over the past few episodes (in our experiments, we checked for stability over 10 episodes). In addition, the episodes must terminate in a goal state. This is in order to prevent an agent that has learned to quickly fail a task from being considered as successfully solving a task. The second is based on the *maximum return* possible in a task, where an agent that receives the maximum return possible on a task can be said to have solved it. The first method assumes the agent can detect a successful completion of a task, while the second assumes the max return (which is task specific) is known.

If the task M can be solved, it is added to the curriculum and the updated policy is returned. Note that we only update the policy if the task can be solved. Otherwise, the learned policy may not be correct. If the task M cannot be solved, RECURSETASKSELECT recursively tries to find and solve a simpler source task.

CREATESOURCETASKS(M, X) creates a set of source tasks $\mathcal{M}_s$ tailored to the agent's experiences X on M, using the heuristic functions defined by [Narvekar *et al.*, 2016]. We partition this set into two groups over lines 9 - 16 based on whether the source can be solved or not. $\mathcal{P}$ contains source tasks that could be solved and their corresponding updated learning agent policies. $\mathcal{U}$ contains tasks that could not be solved, and experience trajectories from the learning agent's attempts on those tasks.

If solvable tasks exist in $\mathcal{P}$, the curriculum design agent needs to select a task to add. We use a heuristic (GETBESTPOLICY) that selects the policy-task pair $(\pi_{M_s}, M_s) \in \mathcal{P}$ that results in the greatest change in policy when evaluated on samples X from the target task. Formally, for each state s encountered in the state sequence from samples X, we compare the action selected by $\pi(s)$, the policy before learning M_s, to $\pi_{M_s}(s)$, the policy after learning M_s, and count the number of states for which the action changed. This number is normalized by the number of states in the sequence X to produce a score. Note that states where the learning agent spends more time in M occur more often in X, and hence bias the score towards policies that update these states. The policy-task pair with the highest score is returned from GETBESTPOLICY, and if the score meets a minimum threshold ϵ, the task is added to the curriculum. The threshold ϵ is used to prevent tasks that don't significantly impact the policy from entering the curriculum.

If no solvable source task is selected, the algorithm instead finds the most relevant unsolvable source tasks, and attempts to break them down further. We calculate the relevance of a source task by computing the overlap between samples from a source X_{M_s} and the samples of the target X. Specifically, for each task-sample pair $(M_s, X_{M_s}) \in \mathcal{U}$, we compute the fraction of states s in the target samples X that are also present in the source X_{M_s}. If function approximation is used, a distance metric such as that by [Ferns *et al.*, 2011] can be used to do this. The task-sample pairs in $\mathcal{U}$ are sorted by their relevance, dropping any that have relevance less than ϵ, and are recursively broken down by calling RECURSETASKSELECT, which tries to find a sub-source task for the current source task. If no tasks can be solved, the recursion ends.

Assuming the target task is solvable, Algorithm 1 is guaranteed to terminate once β increases enough to solve the target task directly. In the worst case, no source tasks are useful. If there are m total source tasks, and it takes n iterations of increasing β to learn the target task, then the whole process makes at most $O(mn)$ recursive calls.

5 Experimental Results

We evaluate our curriculum generation algorithm on a grid world domain, inspired by the lights world domain used by [Konidaris and Barto, 2007]. The world consists of a room, which can contain 4 types of objects. *Keys* are items the agent can pick up by moving to them and executing a pickup action. These are used to unlock *locks*. Each lock in a room is dependent on a set of keys. If the agent is holding the right keys, then moving to a lock and executing an unlock action opens the lock. *Pits* are obstacles placed throughout the domain. If the agent moves into a pit, the episode is terminated. Finally, *beacons* are landmarks that are placed on the corners of pits. A sample domain is pictured in Figure 1a.

The goal of the learning agent is to traverse the world and

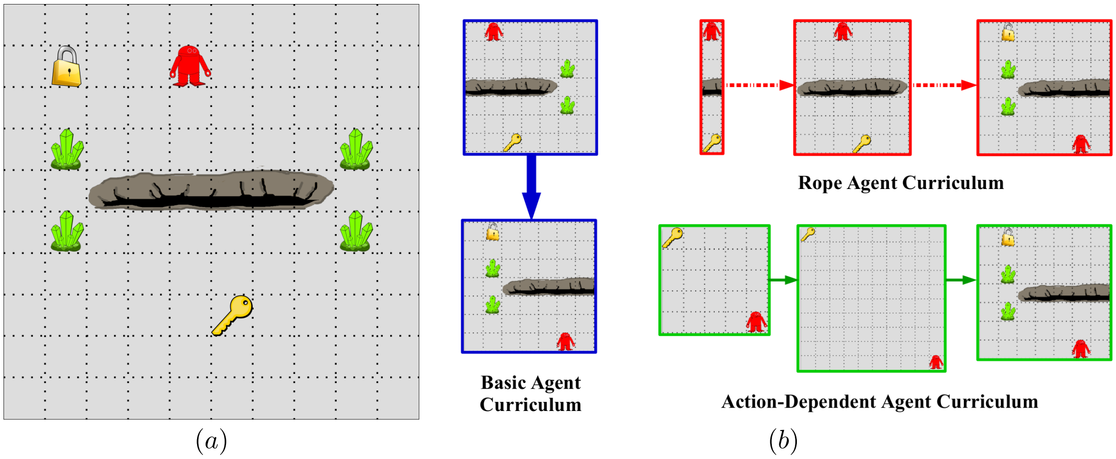


Figure 1: (a) Grid world target task (b) Sample curricula generated for each of the agents. Each one ends in the target task.

unlock all the locks. At each time step, the learning agent can move in one of the four cardinal directions, execute a pickup action, or an unlock action. Moving into a wall causes no motion. Sucessfully picking up a key gives a reward of +500, and sucessfully unlocking a lock gives a reward of +1000. Falling into a pit terminates the episode with a reward of -200. All other actions receive a constant step penalty of -10.

5.1 Learning Agent Descriptions

We created 3 different learning agents that have varied sensing and action capabilities, and show that these different agents can benefit from curricula customized to their individual capabilities, even when facing the same target task.

The first agent, the *basic agent*, has 16 sensors, grouped into 4 on each side. The first sensor in each quadruple measures the Euclidean distance to the closest key from that side, the second measures the distance to the closest lock, the third the distance to the closest beacon, and the fourth detects whether there is a pit adjacent to the agent in that direction. An additional sensor indicates whether all keys in the room have been picked up, which we refer to as the *noKeys* sensor. For example, the perception vector for the agent in Figure 1a is [7.07, 5.10, 6, 6.32, 3.16, 3.16, 4, 2, 4.24, 3.16, 3.61, 2.83, 0, 0, 0, 0, 0], where the first 4 elements are key features for the north, south, east, and west side sensor, followed by the 4 for locks, 4 for beacons, 4 for pits, and the noKey.

The agent used Sarsa(λ) with ϵ-greedy action selection for the learning algorithm $\mathcal{L}$, value function transfer for transfer learning algorithm $\mathcal{T}$, and CMAC tile coding for function approximation. Tile coding is a linear function approximator where the feature space is partitioned into *tiles*. Each tile holds a weight, and typically, several overlapping tilings are used, which control the degree and direction of generalization of the approximator (see [Sutton and Barto, 1998] for a review). For all our agents, the tile widths were 1.

For the basic agent, we created two tilings: one over the 13 percepts from the key, beacon, pit, and noKey sensors, and another over the 13 percepts from the lock, beacon, pit, and noKey sensors. Tiling in this way allows the agent to generalize knowledge about keys and locks learned in source tasks separately. The exploration rate ϵ was set to 0.1, eligibility trace parameter λ to 0.9, and learning rate α to 0.1.

The second, *action-dependent agent*, has the same sensors as the basic agent, but they are tiled differently: one tile is over the lock, pit, and noKey features; a second is over the key, pit, and noKey features; and a third is over the beacon and pit features. In addition, unlike the basic agent, the state representation is action-dependent. That is, when considering the *move right* action, the agent's feature vector uses values only from the right side sensors. For example, the feature vector for the agent in Figure 1a considering the move right action is [6, 4, 3.61, 0, 0], where the values correspond to the key, lock, beacon, pit, and noKey features. The weights in the tilings are shared, so that the same set of weights is used for the state in each of the directions. Sharing weights like this increases the agent's level of generalization.

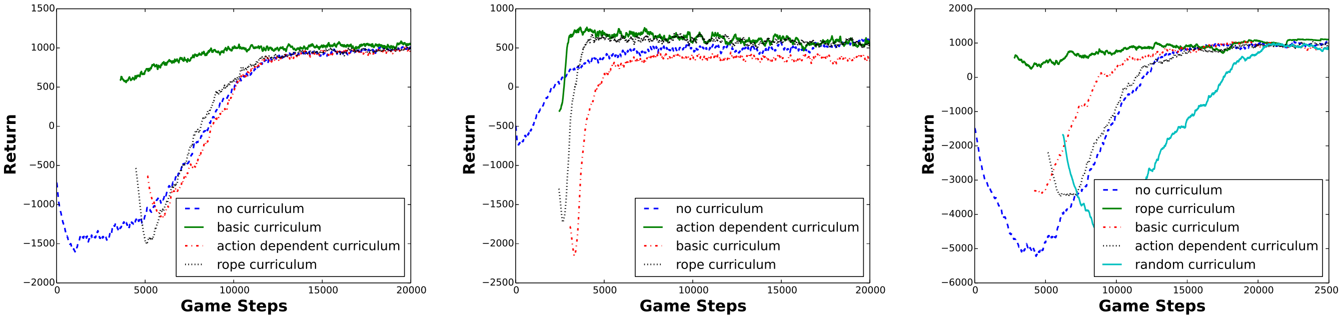
Finally, the *rope agent* is like the basic agent, except that it has 4 additional actions, which are to use a rope in one of the four directions. Doing so opens a path across a pit if one is present, and incurs the step cost of -10. Depending on the task, this action capability can result in a different optimal policy, and thus could benefit from a customized curriculum.

5.2 Curriculum Generation and Results

We used the algorithm presented in Section 4 to automatically generate curricula for each of the 3 agents. The target task M_t was a 10x10 grid world with 1 lock and 1 key separated by a 6 tile pit, as shown in Figure 1a. This task requires agents to learn at least 3 different behaviors: picking up keys, navigating around pits, and unlocking locks.

Each agent was initialized with a uniform random policy, and given an initial learning budget β of 500, which was increased by 500 in each iteration of the loop in Algorithm 1. In order to add a source task, we specified it had to affect the policy by at least $\epsilon = 0.1$. Curriculum generation was terminated when a return $\delta = 700$ was reached.

Tasks were identified as solved using the policy convergence method described in Section 4. We applied the TaskSimplification and OptionSubGoals heuristics defined in [Narvekar *et al.*, 2016] to create source tasks. These created source tasks that varied elements such as the size of the domain, the number of pits, or changed the goal of the task to be picking up certain keys. A total of 15 unique tasks were considered for use by the curriculum algorithm, and 9 were



(a) (b) (c)

Figure 2: Performance on the target task by the (a) basic agent, (b) action-dependent agent and (c) rope agent, after training using various curricula. Each curve was averaged over 500 runs, and is offset to reflect time spent training in source tasks.

used to compose curricula for the different agents.

We evaluated the performance of each agent on the target task using no curriculum, a curriculum tailored for that specific agent, curricula tailored for each of the other two agents, and a random curriculum consisting of 3 randomly selected tasks. Figures 2(a) - 2(c) show the results for the basic agent, action-dependent agent, and rope agent, respectively. The results clearly show that training via the curriculum customized for an agent provides the best benefit. Using a different agent's curriculum was usually suboptimal, and in some cases even hurt performance. Using a random curriculum generally led to performance quite similar to learning from scratch, only delayed. The random tasks added training time without improving learning speed (results are shown for the rope agent. For the other agents, the shape of the random curve was similar to the "no curriculum" curve, but the randomly selected tasks led to horizontal offsets greater than the scale of the graph axes.) Examples of produced curricula for each agent are shown in Figure 1b.

6 Related Work

A variety of transfer learning methods have been proposed, enabling RL agents to transfer samples [Lazaric *et al.*, 2008], options [Soni and Singh, 2006; Perkins *et al.*, 1999], policies [Fernández *et al.*, 2010], models [Fachantidis *et al.*, 2013], and value functions [Taylor and Stone, 2005]. Typically, these methods only consider one-stage transfer (as opposed to a curriculum), and assume the source tasks are given.

To address cases where source tasks are not specified in advance, [Narvekar *et al.*, 2016] proposed a set of heuristic functions for creating tasks that are tailored to the abilities of a learning agent. The source tasks were then used to manually construct a multi-step curriculum. In contrast, we address how multi-step curricula can be automatically constructed.

In the related problem of *task selection*, the goal is to select the best source task for a given target. Proposed solutions typically compute a similarity measure between the MDPs of the source and target task [Ferns *et al.*, 2012; Ammar *et al.*, 2014b], or learn a model of transferability that can be applied to novel source-target task pairs [Sinapov *et al.*, 2015; Isele *et al.*, 2016]. However, none of these methods have been successfully applied to a select a full sequence of tasks.

Automatically selecting and sequencing tasks into a curriculum is an open problem that has received relatively little attention. In most existing work, the tasks are sequenced manually, by either a domain expert [Narvekar *et al.*, 2016] or naive users [Peng *et al.*, 2016]. Most recently, [Svetlik *et al.*, 2017] proposed an automated method to construct a full curriculum, which relies on task descriptors and a heuristic function to estimate the transferability between tasks. The main limitation of their method is that it does not adjust to the agent's unique abilities and learning progress during training. In contrast, our formulation of the curriculum generation process as an MDP accounts for and explicitly models the progress of an agent towards learning a target task.

Finally, curriculum learning has also been explored in the context of supervised learning [Bengio *et al.*, 2009]. Related paradigms, such as multi-task reinforcement learning [Wilson *et al.*, 2007] and lifelong learning [Ammar *et al.*, 2014a] have also been examined. The distinguishing feature of curriculum learning compared to these works is that in curriculum learning, we have full control over the order in which tasks are selected, and the goal is to optimize performance for a specific target task, rather than all tasks.

7 Conclusion and Future Work

In this paper, we presented a novel formulation of curriculum generation as a Markov Decision Process, which explicitly models the progress of an agent as it learns through a sequence of tasks. We described how a policy over a curriculum MDP could be used to produce a curriculum, and proposed an algorithm that approximates one trace of an optimal policy in this MDP. The algorithm was evaluated in a grid world domain to produce curricula tailored to the sensing and action capabilities of 3 different agents. Our results showed that the curricula produced improved learning compared to learning without a curriculum, but also that having *customized* curricula for each agent makes a significant difference.

A limitation of most exisiting automated sequencing approaches, including ours, is that producing the curriculum requires collecting extensive experience in source tasks. Thus, generating a curriculum for a single agent is not practical. We showed that different agents could benefit from different curricula, but produced a curriculum for each agent independently. Thus, an interesting question for future work is: how can we adapt a curriculum generated for one agent to a new agent?

Acknowledgements

The authors would like to thank Matteo Leonetti for helpful discussions on this work. This work took place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CNS-1330072, CNS-1305287, IIS-1637736, IIS-1651089), ONR (21C184-01), AFOSR (FA9550-14-1-0087), Raytheon, Toyota, AT&T, and Lockheed Martin. Peter Stone serves on the Board of Directors of Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

[Ammar *et al.*, 2014a] Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. Online multi-task learning for policy gradient methods. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1206–1214, 2014.

[Ammar *et al.*, 2014b] Haitham Bou Ammar, Eric Eaton, Matthew E Taylor, Decebal Constantin Mocanu, Kurt Driessens, Gerhard Weiss, and Karl Tuyls. An automated measure of mdp similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[Bengio *et al.*, 2009] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48. ACM, 2009.

[Fachantidis *et al.*, 2013] Anestis Fachantidis, Ioannis Partalas, Grigorios Tsoumakas, and Ioannis Vlahavas. Transferring task models in reinforcement learning agents. *Neurocomputing*, 107:23–32, 2013.

[Fernández *et al.*, 2010] Fernando Fernández, Javier García, and Manuela Veloso. Probabilistic policy reuse for intertask transfer learning. *Robotics and Autonomous Systems*, 58(7):866 – 871, 2010.

[Ferns *et al.*, 2011] Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation metrics for continuous markov decision processes. *SIAM Journal on Computing*, 40(6):1662–1714, 2011.

[Ferns *et al.*, 2012] Norm Ferns, Pablo Samuel Castro, Doina Precup, and Prakash Panangaden. Methods for computing state similarity in markov decision processes. *arXiv preprint arXiv:1206.6836*, 2012.

[Isele *et al.*, 2016] David Isele, Mohammad Rostami, and Eric Eaton. Using task features for zero-shot knowledge transfer in lifelong learning. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1620–1626, 2016.

[Konidaris and Barto, 2007] George Konidaris and Andrew G Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pages 895–900, 2007.

[Lazaric *et al.*, 2008] A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the Twenty-Fifth Annual International Conference on Machine Learning (ICML-2008)*, pages 544–551, Helsinki, Finalnd, July 2008.

[Lazaric, 2011] A. Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning: State of the Art*. Springer, 2011.

[Narvekar *et al.*, 2016] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source task creation for curriculum learning. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, May 2016.

[Peng *et al.*, 2016] Bei Peng, James MacGlashan, Robert Loftin, Michael L. Littman, David L. Roberts, and Matthew E. Taylor. An empirical study of non-expert curriculum design for machine learners. In *In Proceedings of the IJCAI Interactive Machine Learning Workshop*, 2016.

[Perkins *et al.*, 1999] Theodore J Perkins, Doina Precup, et al. Using options for knowledge transfer in reinforcement learning. *University of Massachusetts, Amherst, MA, USA, Tech. Rep*, 1999.

[Sinapov *et al.*, 2015] Jivko Sinapov, Sanmit Narvekar, Matteo Leonetti, and Peter Stone. Learning inter-task transferability in the absence of target task samples. In *Proceedings of the 2015 ACM Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. ACM, 2015.

[Soni and Singh, 2006] Vishal Soni and Satinder Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *Proceedings of American Association for Artificial Intelligence (AAAI)*, 2006.

[Sutton and Barto, 1998] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[Svetlik *et al.*, 2017] Maxwell Svetlik, Matteo Leonetti, Jivko Sinapov, Rishi Shah, Nick Walker, and Peter Stone. Automatic curriculum graph generation for reinforcement learning agents. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, February 2017.

[Taylor and Stone, 2005] Matthew E. Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 53–59, New York, NY, July 2005. ACM Press.

[Taylor and Stone, 2009] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

[Taylor *et al.*, 2007] Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

[Wilson *et al.*, 2007] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 1015–1022. ACM, 2007.