Automaton-Guided Curriculum Generation for Reinforcement Learning Agents

Yash Shukla[1], Abhishek Kulkarni[2], Robert Wright[3], Alvaro Velasquez[4], Jivko Sinapov[1]

[1]Tufts University
[2]University of Florida
[3]Georgia Tech Research Institute
[4]University of Colorado Boulder

{yash.shukla, jivko.sinapov}@tufts.edu, a.kulkarni2@ufl.edu, robert.wright@gtri.gatech.edu, alvaro.velasquez@colorado.edu

Abstract

Despite advances in Reinforcement Learning, many sequential decision making tasks remain prohibitively expensive and impractical to learn. Recently, approaches that automatically generate reward functions from logical task specifications have been proposed to mitigate this issue; however, they scale poorly on long-horizon tasks (i.e., tasks where the agent needs to perform a series of correct actions to reach the goal state, considering future transitions while choosing an action). Employing a curriculum (a sequence of increasingly complex tasks) further improves the learning speed of the agent by sequencing intermediate tasks suited to the learning capacity of the agent. However, generating curricula from the logical specification still remains an unsolved problem. To this end, we propose *AGCL*, Automaton-guided Curriculum Learning, a novel method for automatically generating curricula for the target task in the form of Directed Acyclic Graphs (DAGs). *AGCL* encodes the specification in the form of a deterministic finite automaton (DFA), and then uses the DFA along with the Object-Oriented MDP (OOMDP) representation to generate a curriculum as a DAG, where the vertices correspond to tasks, and edges correspond to the direction of knowledge transfer. Experiments in grid-world and physics-based simulated robotics domains show that the curricula produced by *AGCL* achieve improved time-to-threshold performance on a complex sequential decision-making problem relative to state-of-the-art curriculum learning (e.g, teacher-student, self-play) and automaton-guided reinforcement learning baselines (e.g, Q-Learning for Reward Machines). Further, we demonstrate that *AGCL* performs well even in the presence of noise in the task's OOMDP description, and also when distractor objects are present that are not modeled in the logical specification of the tasks' objectives.

1 Introduction

Deep reinforcement learning utilizes neural networks to solve complex tasks ranging from Atari games to robot manipulation (Gao and Wu 2021; Karpathy and Van De Panne 2012). Despite these advances, many tasks remain prohibitively expensive to learn from scratch, requiring large number of interactions. This problem worsens in long-horizon tasks where sparse reward settings and poor goal representations make the task difficult to solve. In many

practical scenarios, the objective for a task is known before commencing learning the task, and thus it can be captured using high-level specifications that have an equivalent automaton representation (Velasquez et al. 2021; Jothimurugan et al. 2021). This formulation allows decomposing the task into sub-goals, where each sub-goal can be learned by a Reinforcement Learning (RL) agent. This approach of automaton-guided RL is beneficial in scenarios where the high-level task objective is known beforehand and can be expressed using an automaton, but the low-level transition dynamics are unavailable and hence motion and symbolic planners cannot be reliably used. Recently, many approaches have made use of richer language representations (such as finite-trace Linear Temporal Logic (LTL_f) (De Giacomo and Vardi 2013)) to incorporate history in the task Markovian Decision Processes (MDPs). This technique helps the RL agent to shape reward effectively to achieve the goal in sparse reward settings, eliminating the need to rely on human guidance for shaping the reward function (Camacho et al. 2018; Icarte et al. 2018; Velasquez et al. 2021). However, reward shaping falls short in complex sequential-decision making tasks as it does not generate simpler sub-tasks suited to the current knowledge of the agent, and incentivizes policies to reach local optima. In this work, we extend automaton-guided RL beyond reward shaping, by deriving a curriculum for the target objective. Curriculum Learning (CL) intends to optimize the order in which an agent attempts tasks, increasing the expected return while reducing training time for complex tasks (Narvekar et al. 2020; Foglino, Christakou, and Leonetti 2019). Nevertheless, automaton-guided curriculum generation still remains an unsolved problem in literature.

We explore representing the high-level task objective using finite-trace Linear Temporal Logic (LTL_f) formulas (De Giacomo and Vardi 2013) which can be equivalently represented using Deterministic Finite Automaton (DFA). Using the structure of the DFA, we generate a curriculum for the target objective. While the DFA provides a graphical representation of the sequence in which the sub-goals must be achieved, it does not specify the individual sub-tasks of the curriculum. Thus, generating a curriculum is non-trivial as it requires the agent to reason over multiple potential curriculum environment configurations for the same sub-goal objective. Given a high-level task objective in
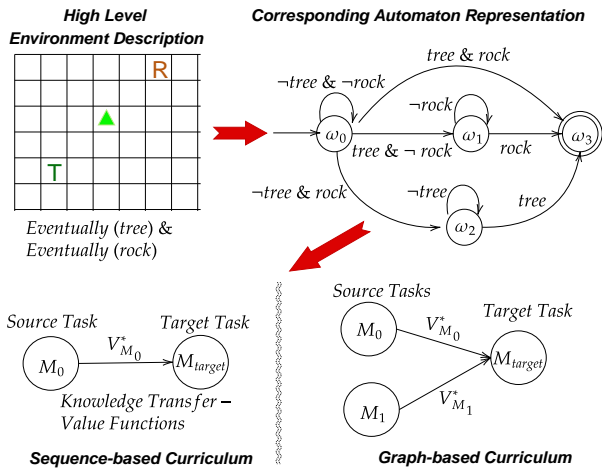
Figure 1: Overview of the automaton-guided curriculum generation procedure. Given the high-level task description, we use the equivalent automaton representation to come up with a sequence-based and a graph-based curriculum.

the form of a DFA, we propose *AGCL*, Automaton-guided Curriculum Learning (Fig. 1), that generates two types of curricula: 1) A sequence of sub-tasks that increase in order of difficulty for the agent, and 2) A directed acyclic graph (DAG), where agents can transfer knowledge concurrently learned in multiple source tasks to learn a common target task[1]. Unlike previous graph-based curriculum approaches (Svetlik et al. 2017; Silva and Costa 2018), ours does not require access to all possible state configurations of the environment. Further, generating the curriculum does not require additional interactions with the environment, and learning through the curriculum yields quicker convergence to a desired performance compared to learning from scratch, automaton-guided reward shaping baselines - GSRS (Camacho et al. 2018), QRM (Icarte et al. 2018), and curriculum learning baselines - Teacher-Student (Matiisen et al. 2020) and self-play (Sukhbaatar et al. 2018).

We further perform an extensive evaluation on a set of challenging robotic navigation and manipulation tasks and demonstrate that *AGCL* reduces the number of interactions with the target environment by orders-of-magnitude when compared to state-of-the-art curriculum learning and automaton-guided reinforcement learning baselines.

2 Related Work

Curriculum Learning (CL) in RL has been studied for games (Gao and Wu 2021), robotic tasks (Karpathy and Van De Panne 2012), and self-driving cars (Qiao et al. 2018). A survey on CL (Narvekar et al. 2020) summarizes the three main elements of CL as task generation, task sequencing, and knowledge transfer. Task generation builds a set of source tasks given the target task (Kurkcu, Campolo, and Tee 2020). Task sequencing optimizes the task sequence to enhance learning in the target task (Narvekar, Sinapov, and

---

[1] An in-depth analysis of Fig. 1 is provided in Section 3

Stone 2017; Matiisen et al. 2020), and knowledge transfer determines what information must be transferred from the source to the target to promote effective learning (Da Silva and Costa 2019; Taylor and Stone 2009). Existing graph-based CL methods (Svetlik et al. 2017; Silva and Costa 2018) assume knowledge of the set of states in the task, and thus cannot be used for complex tasks that require function approximation. Metaheuristic search methods serve as a tool to evaluate the CL frameworks (Foglino, Christakou, and Leonetti 2019). In most methods, optimizing a curriculum is still computationally expensive and sometimes takes more interactions compared to learning from scratch. Our proposed framework addresses this concern by utilizing high-level logical specification to derive an optimized graphical representation of a curriculum, eliminating the sunk cost of interactions required to optimize the curriculum.

Automaton-guided RL approaches specify tasks using temporal logic-based high-level language specifications (Toro Icarte et al. 2018; Velasquez et al. 2021; Jiang et al. 2021). Most approaches generate a dense reward function to speed up learning. Learning separate policies for sub-goals of a task helps abstract knowledge which can be used in new tasks (Icarte et al. 2018). Another technique is to shape the reward inversely proportional to the distance from the accepting node in the automaton (Camacho et al. 2018) however, it leads to inefficient reward settings. Augmenting the reward function with a dynamic Monte Carlo Tree Search helps mitigate this problem (Velasquez et al. 2021). DIRL interleaves high-level planning with RL to learn a policy for each edge, overcoming challenges introduced by poor representations (Jothimurugan et al. 2021). However, this approach becomes inefficient when there are multiple paths to the target in the graph, and other reward shaping techniques prove inefficient compared to curriculum learning (Pocius et al. 2018). In this work, we use the underlying DFA to develop a graphical representation of the curriculum, and implement a transfer learning mechanism compatible with a function-approximator, outperforming state-of-the-art baselines.

3 Theoretical Framework

3.1 Markov Decision Processes

An episodic Markov Decision Process (MDP) M is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \mathcal{S}_0, \mathcal{S}_f, \gamma, L)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}(s'|s, a)$ is the transition function, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function, $\mathcal{S}_0$ and $\mathcal{S}_f$ are the sets of starting and final states respectively, and $\gamma \in [0, 1]$ is the discount factor. At each timestep t, the agent observes a state s and performs an action a given by its policy function $\pi_\theta(a|s)$, with parameters θ. The agent's goal is to learn an *optimal policy* π^*, maximizing its discounted return $G_0 = \sum_{k=0}^{K} \gamma^k r(s'_k, a_k, s_k)$ until the end of the episode at timestep K. The labeling function $L : \mathcal{S} \rightarrow 2^{AP}$ maps a state in the MDP to a set of atomic propositions that hold true for that state. For example, in Fig. 1 the set of atomic propositions, $AP = \{tree, rock\}$.
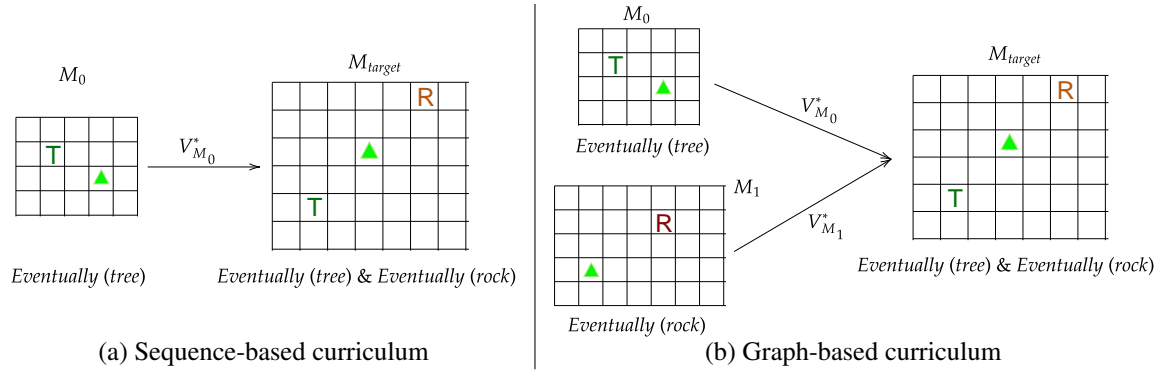
Figure 2: Examples of sequence-based (left) and graph-based (right) curricula.

3.2 Curriculum

Let $\mathcal{M}$ be a set of tasks, where $M_i = (\mathcal{S}_i, \mathcal{A}_i, P_i, R_i, \mathcal{S}_{0,i}, \mathcal{S}_{f,i}, \gamma_i, L_i)$ is a task in $\mathcal{M}$. Let $\mathcal{D}^{M_i}$ be the set of samples associated with task M_i : $\mathcal{D}^{M_i} = \{(s, a, r, s') | s \in \mathcal{S}_i, a \in \mathcal{A}_i, s' \sim \mathcal{P}_i(\cdot|s, a), r \leftarrow R_i(s, a, s')\}$. A curriculum $T = (\mathcal{V}, \mathcal{E}, g, \mathcal{M})$ is a directed acyclic graph, where $\mathcal{V}$ is the set of vertices, $\mathcal{E} \subseteq \{(x, y) | (x, y) \in V \times V \land x \neq y\}$ is the set of directed edges, and $g : \mathcal{V} \rightarrow \{\mathcal{D}^{M_i} | M_i \in \mathcal{M}\}$ is a function that associates vertices to samples of a single task in $\mathcal{M}$. A directed edge (v_j, v_k) indicates that samples associated with $v_j \in \mathcal{V}$ should be trained on before samples associated with $v_k \in \mathcal{V}$. All paths terminate on a single sink node $v_t \in \mathcal{V}$.[2]

A graph-based curriculum is a general case of a curriculum, where the indegree and outdegree of each vertex $v \in \mathcal{V}$ can be greater than one, and there can be multiple source nodes but only one sink node. A sequence-based curriculum is a special case of a curriculum, where the indegree and outdegree of each vertex $v \in \mathcal{V}$ is at most 1, and there is only one source node and one sink node as shown in Fig. 1. Examples of a sequence-based curriculum and a graph-based curriculum are shown in Fig. 2(a) Fig. 2(b) respectively.

3.3 Object Oriented MDP

The Object-Oriented MDP (OOMDP) representation abstracts the task description, as to intuitively generalize the elements in the environment and their properties (Diuk, Cohen, and Littman 2008). In OOMDPs, the task space is abstracted by a set of classes $\mathcal{C}$ where each class $C_i \in \mathcal{C}$ has a parameter set $Par(C_i) = \{C_i.p_1, \ldots, C_i.p_{|p|}\}$. Each parameter $C_i.p_i$ has a range of values, given by $Range(C_i.p_i)$. At any given time, an environment consists of a set of objects $\mathcal{O} = \{o_1, \ldots, o_{|o|}\}$, where each object o_i is an instance of a class C_i and is defined by the parameters for that class $o_i = Par(C_i(o_i))$. For a task, the OOMDP state s_{oo} is given by the union of all object states $s_{oo} = \cup_{o_i \in \mathcal{O}} State(o_i)$, where each object state is the value of the parameters of that object: $State(o_i) = \{o_i.p_1, \ldots, o_i.p_{|p|}\}$. Since the OOMDP representation is a high-level abstraction of the task space, we

---

[2] We find curriculum for single target task. Curriculum design for multiple target tasks is beyond the scope of this paper.

assume a many-to-one mapping between a subset of MDP states ($\mathcal{S}_{sub}$) and the OOMDP state (s_{oo}); $w : \mathcal{S}_{sub} \rightarrow s_{oo}$. For the task in Fig. 1, the OOMDP description consists of classes $\mathcal{C} = \{world\_size, trees, rocks\}$, with the parameters: $\mathcal{C}(world\_size) = \{width, height\}$ (width and height of the grid), $\mathcal{C}(trees) = \{trees_{env}, trees_{inv}\}$ (number of trees in the environment and inventory of the agent), and similarly, $\mathcal{C}(rocks) = \{rocks_{env}, rocks_{inv}\}$. In a task, classes have objects whose parameters are assigned values within a range. Example: $Par(world\_size[height])) = [5, 10]$ denotes that the height of the grid can have any value within the range $[5, 10]$. The MDP transition $s \xrightarrow{a} s'$ and the OOMDP transition $s_{oo} \xrightarrow{a} s'_{oo}$ occur synchronously.

3.4 Linear Temporal Logic and Deterministic Finite Automata (DFA)

We define the high-level specification of our task using finite-trace Linear Temporal Logic (LTL_f) formulas (De Giacomo and Vardi 2013). LTL_f allows us to succinctly express complex temporal tasks such as reachability, safety, recurrence, persistence or a combination of these. Most importantly, the language of any LTL_f formula can equivalently be represented as a Deterministic Finite Automaton (DFA).

Given a set of atomic propositions, AP, a formula in LTL_f is constructed inductively using the operations $p, \neg\phi, \phi_1 \lor \phi_2, \mathbf{X}\phi, \mathbf{G}\phi, \mathbf{F}\phi, \phi_1 \mathbf{U} \phi_2$ where $p \in AP$ and ϕ, ϕ_1 and ϕ_2 are LTL_f formulas. The operator $\neg$ denotes negation, $\lor$ denotes disjunction, and the operators $\mathbf{X}, \mathbf{G}, \mathbf{F}, \mathbf{U}$ denote *Next*, *Always*, *Eventually*, and *Until* respectively. An LTL_f formula can be translated into a DFA $\mathcal{U} = \{\Omega, \omega_0, \Sigma, \delta, F\}$, where Ω is the set of nodes with initial node $\omega_0 \in \Omega$, $\Sigma = 2^{AP}$ is an alphabet defined over a set of atomic propositions AP, $\delta : \Omega \times \Sigma \rightarrow \Omega$ is a deterministic transition function, and $F \subseteq \Omega$ is the set of accepting nodes. Given any two states $\omega, \omega' \in \Omega$ and a symbol $\sigma \in \Sigma$, we denote the transition $\omega' = \delta(\omega, \sigma)$ as $\omega \xrightarrow{\sigma} \omega'$. A path $\zeta = \omega_0 \omega_1 \ldots \omega_n$ is a finite-length sequence of states such that, for any $i = 0 \ldots n - 1$, we have $\omega_{i+1} = \delta(\omega_i, \sigma_i)$. The occurrence operator, $Occ(\zeta)$, defined as $\{\omega \in \Omega | \omega$ appears in $\zeta\}$, returns the set of nodes in the path ζ.

The set of atomic propositions in AP is shared within the

MDP and the DFA. The labeled MDP and the automaton (DFA) operate synchronously. Specifically, a transition $s \xrightarrow{a} s'$ in the labeled MDP triggers a transition $\omega \xrightarrow{L(s')} \omega'$ in the DFA. Therefore, the high-level specification is achieved in the labeled MDP if and only if a final state is reached in the DFA. We define a one-to-many relationship using the function $P : \Omega \rightarrow \mathcal{M}$, that given a node $\omega \in \Omega$ in the DFA, returns the set of tasks that can reach the node ω from the start node ω_0[3]. The equivalent DFA representation of the LTL_f objective $\mathbf{F}(tree) \wedge \mathbf{F}(rock)$ is shown in Fig. 1.

3.5 Running Example

Consider the environment description shown in Fig. 1. We start with the target task MDP M_{target} and its OOMDP description (described in Sec. 3.3) along with the LTL_f formula for the task ($\mathbf{F}(tree) \wedge \mathbf{F}(rock)$) which corresponds to the automaton shown in Fig. 1. When the agent follows a policy π that collects a tree followed by a rock, it satisfies the path $\zeta : \omega_0 \xrightarrow{tree} \omega_1 \xrightarrow{rock} \omega_3$. Since ω_3 is in the set of accepting nodes F, the policy produces a trajectory in the MDP that satisfies the LTL_f objective. Generating curriculum given the DFA involves reasoning over the trace paths as well as the set of MDPs that can generate trajectories to reach a node in the DFA. For example, to have a trajectory (of length > 0) that reaches the node ω_1, the agent must have a *tree* in its inventory by the end of the episode. To achieve this, at least one *tree* must be present in the environment when the episode starts for the agent to collect. Using this information, a potential source task of the curriculum is generated by varying the parameter values for the objects within the range specified in the OOMDP representation. Thus, a query to the function $P(\omega_1)$ will generate the set of MDPs $\mathcal{M}_1$ that have at least 1 *tree* in the initial task configuration, while varying other object parameters. The set of initial and final OOMDP states for $\mathcal{M}_1$ will be given by $w(P(\omega_1)(\mathcal{S}_0))$ and $w(P(\omega_1)(\mathcal{S}_f))$ respectively as the function w maps the MDP state to an OOMDP state. An example of a sequence-based curriculum for the path $\zeta : \omega_0 \xrightarrow{tree} \omega_1 \xrightarrow{rock} \omega_3$ is shown in Fig. 2, where in the first task M_0, the agent learns to collect a *tree* in a smaller environment, while transferring the knowledge to the final task. For the graph-based curriculum in Fig. 2, two agents simultaneously learn to collect a *tree* and a *rock* separately, and transfer their knowledge to learn the target task. The curriculum for the agent that learns to collect a *tree* in M_0 is derived from the set of MDPs produced using the path $\zeta : \omega_0 \xrightarrow{tree} \omega_1 \xrightarrow{rock} \omega_3$, whereas the curriculum for the agent that learns to collect a *rock* in M_1 is derived using the path $\zeta : \omega_0 \xrightarrow{rock} \omega_2 \xrightarrow{tree} \omega_3$. The agent's trajectory in the final target task M_{target} can follow any of the paths in the DFA. The knowledge from the source tasks in the curriculum guides the agent in the final target task to make decisions based on its prior experiences. In the following sections, we discuss our automaton-guided curriculum generation approach.

---

[3]Details on function P in Appendix Section A.
https://github.com/tufts-ai-robotics-group/Automaton-guided-CL

3.6 Problem Formulation

CL aims to generate a curriculum, such that the agent's convergence or its time-to-threshold performance on the final target task (M_{target}) improves relative to learning from scratch. Time-to-threshold (Δ) metric computes how much faster an agent can learn a policy that achieves expected return $G \geq \delta$ on the target task if it learns through a curriculum, as opposed to learning from another approach (Foglino, Christakou, and Leonetti 2019). Here δ is desired performance threshold, in success rate or episodic reward. An optimal curriculum would converge to the expected return value in the target task the quickest. Formally:

$$T^* := \operatorname*{argmin}_{T \in \mathcal{T}} [a_\delta^{M_{target}} + \sum_{M_i \in T} a^{M_i}]$$

where $\mathcal{T}$ is the set of all curricula, $a_\delta^{M_{target}}$ is the number of actions the agent took in the target task M_{target} to achieve threshold performance δ, and a^{M_i} is the number of actions the agent took in the source task M_i of the curriculum.

Our objective is to find a curriculum $T = \{\mathcal{V}, \mathcal{E}, g, \mathcal{M}\}$ that improves an agent's time-to-threshold performance on the target task, given the LTL_f representation and the OOMDP and MDP description of the target task.

4 Methodology

The high-level curriculum generation algorithm is given in Algorithm 1. To begin, we assume the following:

1. We have access to a high-level target task (M_{tar}) objective which is expressed using LTL_f formulas, and can be equivalently represented using a DFA $\mathcal{U}$.

2. We have access to the OOMDP description of the target task. The OOMDP description contains the set of classes $\mathcal{C}$ of the environment, and for each class, the set of parameters for that class, along with the range of acceptable values for the parameters. Varying these parameters give us different tasks of the curriculum. We also have the initial and goal OOMDP states of the final task of the environment.

3. We have access to the functions P that given a node in the DFA, return the set of MDPs that can reach the node, and access to function w that performs mapping between MDP states and OOMDP states.

Given the DFA, we first generate a key-value pair dictionary K, where each key is a node of the automaton, and the value is the set of MDPs that can reach the node, with each MDP augmented with the initial and final OOMDP state configuration of the MDP, resulting in the MDP-OOMDP tuple $(M, s_{0,oo}, s_{f,oo})$, lines 1-2. The function Get_Trace_Paths returns the set of all paths (Z) that start from the initial node ω_0 and reach an accepting node $\omega_f \in F$. We consider only acyclic DFAs, ignoring paths that contain a cycle. A curriculum designed with this consideration will prune curriculum candidates that intend to learn the same task repeatedly, ensuring progress in the curriculum. Then, for each path $\zeta \in Z$, the List_Candidates function generates an ordered-list of sequence-based curriculum candidates by choosing a MDP-OOMDP tuple from

each node of the path, and sequencing it according to the sequence of nodes in the path ζ. We repeat this until we attain a set of all possible combinations of MDP-OOMDP tuple sequences (lines 4-7). This yields an exhaustive set Ψ where each element in the set is an ordered list of MDP-OOMDP tuples, and the length of the list is equal to the number of transitions in the trace path. Thus, each list corresponds to a curriculum candidate T where the vertices of the graph are the MDPs in the list, and the edges are defined by the sequence of the MDPs in the list. Once we have a set of sequence-based curricula candidates, we choose an effective curriculum from this set. We introduce the jump score, where we assign a score for each pair of consecutive source tasks in the curriculum (lines 10-16).

Jump score: Inspired by the difficulty scores for supervised learning (Weinshall, Cohen, and Amir 2018), we assign a jump score $J : (M_i, M_j) \rightarrow \mathbb{R} \in [0, 1]$ for each pair of consecutive source tasks M_i and M_j:

$$J_{M_i \rightarrow M_j} = 1/2 * (sim_t(M_j, M_{tar}) - sim_t(M_i, M_{tar})$$
$$+ sim_g(M_j, M_{tar}) - sim_g(M_i, M_{tar})) \quad (1)$$

where, $sim_t(M_i, M_{tar})$ is the task configuration similarity between the task M_i and the final target task M_{tar}[4], while $sim_g(M_i, M_{tar})$ is the goal state similarity. The jump score $J_{M_i \rightarrow M_j}$ calculates how dissimilar two consecutive source tasks in the curriculum are by calculating the difference of the similarity of the task and goal configurations of each of the two consecutive tasks with the final target task.

Since we have the OOMDP initial state configuration $s_{0,oo}$ of each pair of consecutive source tasks of the curriculum (M_i, M_j) and the final target task M_{tar}, the task configuration similarity between a task and the final target task is measured as the parameter value overlap between the initial OOMDP state values of the two tasks.

$$sim_t(M_i, M_{tar}) = \frac{1}{\sum_{o \in \mathcal{O}_{M_{tar}}} |o.p^0|} \sum_{o \in \mathcal{O}_{M_{tar}}} \sum_{o.p \in o} \frac{o.p^0_{M_i}}{o.p^0_{M_{tar}}}$$
$$(2)$$

where $o.p^0_{M_i}$ is the value of the parameter of the object o in the OOMDP state $s_{0,oo}$ of task M_i. Likewise, we also have the OOMDP goal state $s_{f,oo}$ of each pair of consecutive source tasks and the final target task. We calculate the OOMDP goal state similarity of a task with respect to the final target task using:

$$sim_g(M_i, M_{tar}) = \frac{1}{\sum_{o \in \mathcal{O}_{M_{tar}}} |o.p^f|} \sum_{o \in \mathcal{O}_{M_{tar}}} \sum_{o.p \in o} \frac{o.p^f_{M_i}}{o.p^f_{M_{tar}}}$$
$$(3)$$

where $o.p^f_{M_i}$ is the value of the parameter of the object o in the OOMDP state $s_{f,oo}$ of task M_i.

A lower jump score corresponds to a pair of tasks that have similar initial OOMDP task configurations as well as similar OOMDP goal state configurations. Along any path ζ, the sum of the jump scores among all consecutive task pairs will always be equal to 1. The set of OOMDP states the agent visits in an episode can be potentially large, and

---

[4]More on MDP similarity: (Visús, García, and Fernández 2021)

do not yield much information that might help in choosing a suitable curriculum candidate. Additionally, access to the set of OOMDP states that the agent might visit in an episode before the agent even attempts the episode requires knowledge of the transition dynamics, which is unavailable. Hence, only the initial and final states are used while calculating the jump score.

Next, we calculate the average jump score of each curriculum candidate $\psi \in \Psi$ and store it in a Curriculum - Jump Score dictionary $\mathcal{J}$ (line 16). For sequence-based curricula, the curriculum is determined by the candidate that yields the lowest average jump score (lines 17-18).

$$T = \underset{\psi \in \Psi}{\operatorname{argmin}} \mathcal{J}(\overline{J}_\psi) \quad (4)$$

Intuitively, the curriculum T is an ordered list of MDP-OOMDP tuples that have the lowest cumulative average jump among any two consecutive source tasks, i.e. the curriculum has the lowest task and goal state dissimilarity between any two consecutive tasks. A lower dissimilarity will result in a stronger knowledge transfer.

The graph-based curriculum is determined by the set of elements in Ψ that yield the cumulative average jump score lower than a predetermined threshold value η (lines 19-21).

$$T' = \{\psi \in \Psi \ni \mathcal{J}(\overline{J}_\psi) \leq \eta\} \quad (5)$$

Thus, T' is a set where each element is an ordered list of MDPs (each element is a sequence-based curriculum). A graph is generated by considering equivalent MDPs in T' as common nodes, and the edges are given by the sequence of MDPs in each element in T' (line 22). The weights on the edges denote the proportion of knowledge contribution of a source task and is discussed further below. The worst case time-complexity of the algorithm is $O(|\mathcal{V}|^3|\mathcal{E}||C|^2|p|^2)$ and where $|\mathcal{V}|$ and $|\mathcal{E}|$ are the number of vertices and edges in the DFA's DAG and $|C|$ and $|p|$ are the number of classes and maximum number of class parameters in the OOMDP.

Knowledge Transfer in curriculum learning involves leveraging learned knowledge from a source task and transferring relevant knowledge to the next task in the curriculum. In our curriculum setting, each individual source task is learned using DQN (Mnih et al. 2015), and hence we perform value function transfer, where the weights of the neural network of the learned value function of a source task are initialized as the weights of the value function of the next task, i.e. $V^i_{M_j} \leftarrow V^*_{M_i}$. Thus, instead of commencing the next task M_j with random $V(s)$ values, the learned value function of the source task biases action selection in the next task according to the experience already collected in the source task. In scenarios where the action spaces are continuous, the RL algorithm needs to be adapted to suit the needs of the domain. DDPG (Lillicrap et al. 2016) and PPO (Schulman et al. 2017) allow continuous action spaces and can replace the DQN as the base RL algorithm. In case of a sequence-based curriculum, where we transfer knowledge from only one source task M_i to the next task M_j in the curriculum, the value function of M_j is initialized to be the learned value function of M_i. This process is continued until we reach the end of curriculum, culminating in the final target task

Algorithm 1: AGCG $(\mathcal{U}, M_{tar}, w, P, \eta,$ CL-Type$)$

Output: Curriculum: T
Placeholder Initialization: Set of all acyclic trace paths $Z \leftarrow \emptyset$
Node-MDP Dictionary $K \leftarrow \emptyset$
Curriculum - Jump Score dictionary $\mathcal{J} \leftarrow \emptyset$
Set of curriculum candidates $\Psi \leftarrow \emptyset$
Algorithm:
1: **for** $\omega \in \Omega$ **do**
2:    $K \leftarrow K \cup \{\omega : (P(\omega), w(P(\omega)(\mathcal{S}_0)), w(P(\omega)(\mathcal{S}_f)))$
3: **end for**
4: $Z \leftarrow$ Get_Trace_Paths$(\mathcal{U})$
5: **for** $\zeta \in Z$ **do**
6:    **for** $\omega_i \in Occ(\zeta)$ **do**
7:       $\Psi \leftarrow \Psi \cup \{[$List_Candidates$(K(\omega_i))]\}$
8:    **end for**
9: **end for**
10: **for** $\psi \in \Psi$ **do**
11:    $jump\_sum \leftarrow 0$
12:    **for** $i = 1$ to $|\psi| - 1$ **do**
13:       $jump\_sum \leftarrow jump\_sum + J_{M_i \rightarrow M_{i+1}}$
14:    **end for**
15:    $\overline{J}_\psi \leftarrow jump\_sum / |\psi|$
16:    $\mathcal{J} \leftarrow \mathcal{J} \cup \{\psi : \overline{J}_\psi\}$
17: **end for**
18: **if** CL-Type == Sequence-based **then**
19:    $T = \arg\min_{\psi \in \mathcal{J}} \mathcal{J}(\overline{J}_\psi)$
20: **else if** CL-Type == Graph-Based **then**
21:    $T' = \{\psi \in \Psi \ni \mathcal{J}(\overline{J}_\psi) \leq \eta\}$
22:    $T \leftarrow$ Graph(T')
23: **end if**
24: **return** T

M_{tar}. In case of a graph-based curriculum, where we transfer knowledge from multiple source tasks $(M_i, M_{i+1}, \ldots)$ to the next task in the curriculum (M_j), the value function of the next task (M_j) is initialized to be a weighted sum of the value functions of its source tasks, i.e.

$$V_j^i = \beta_0 \cdot V_i^* + \beta_1 \cdot V_{i+1}^* + \ldots + \beta_l \cdot V_{i+l}^* \quad (6)$$

where V_j^i is the initial value function of the target task M_j, V_i^* is the learned value function of the source task M_i, l is the number of source tasks, and β is a scalar weighting factor. To calculate β, we use the following:

$$\beta_i \propto \frac{1}{J_{M_i \rightarrow M_j}} \quad \text{and} \quad \sum_{k=0}^{k=l} \beta_{i+k} = 1 \quad (7)$$

i.e., the lower the jump score between tasks M_i and M_j, the higher the similarity between their OOMDP task and goal configurations, thus, higher the weightage of the learned value function of the task M_i, and vice versa. A higher similarity will correspond to a stronger positive transfer. In case of a graph-based curriculum, we learn the tasks that correspond to the leaf nodes initially, transferring knowledge using Equations 6 and 7, culminating in the final target task[5].

---

[5]Link to source code and appendix:
https://github.com/tufts-ai-robotics-group/Automaton-guided-CL

5 Experimental Results

We aim to answer the following questions: (1) Does AGCL yield sample efficient learning? (2) How does it perform in environments that have distractor objects that are not modeled in the LTL$_f$ specification? (3) How does it perform when the exact OOMDP description is unknown? (4) Does it yield sample efficient learning when the OOMDP parameter space is continuous? (5) How does it perform when generating all potential curricula candidates is intractable?

5.1 AGCL - Gridworld Results

To answer our first question, we evaluated *AGCL* on a gridworld domain with the LTL$_f$ objective:

$$\mathbf{G}((t \rightarrow \neg r \wedge \neg p) \wedge (r \rightarrow \neg t \wedge \neg p) \wedge (p \rightarrow \neg r \wedge \neg t))$$
$$\wedge (\neg p \mathbf{U}(t \wedge \mathbf{X}(\neg p \mathbf{U} t))) \wedge (\neg p \mathbf{U} r) \wedge \mathbf{F} p \quad (8)$$

where t, r, p correspond to the atomic propositions *tree*, *rock* and *pogo-stick* respectively. Essentially, the agent needs to collect 2 *trees* and 1 *rock* (by navigating to the objects and *breaking* them) before approaching the crafting table to *craft* a *pogo-stick*. This is a complex sequential decision making task and requires $\sim 10^8$ interactions to reach convergence (Shukla et al. 2022). The OOMDP description is given by: $\mathcal{C} = \{world\_size, trees, rocks, crafting\_table\}$, where the class parameters are: $C(world\_size) = \{width, height\}$; $C(x) = \{x_{env}, x_{inv} | x \in \{trees, rocks\}\}$ and $C(crafting\_table) = \{crafting\_table_{env}\}$. The parameters $width$ and $height$ can assume values in the range $[6, 12]$ and the number of $trees, rocks$ and $crafting\_table$ in the environment can assume a value in the ranges of $[0, 4]$, $[0, 2]$ and $[0, 1]$ respectively. In this environment, the agent can move 1 cell forward if the cell ahead is clear or rotate $\pi/2$ left or right. In the target task, the agent receives a reward of 10^3 upon crafting a *pogo-stick*, and -1 reward for all other steps. The agent's sensor emits a beam at incremental angles of $\pi/4$ to determine the closest object in the angle of the beam (i.e., the agent receives a local-view of its environment). Two additional sensors provide information on the amount of *trees* and *rocks* in the agent's inventory (More details in Appendix B).

The learning curves in Fig. 3 depict the performance of our proposed sequence and graph-based *AGCL* method against five baseline approaches namely learning from scratch, automaton-guided reward shaping baslines: GSRS (Camacho et al. 2018), QRM (Icarte et al. 2018), and curriculum learning baselines: Teacher-Student (Matiisen et al. 2020) and self-play (Sukhbaatar et al. 2018), all implemented using the same RL algorithm DQN (Mnih et al. 2015). The automaton-guided reward shaping baselines do not employ a curriculum and build upon naive reward shaping by modifying the reward inversely proportional to the distance from the DFA goal state and by learning policies for each DFA state transition. The curriculum learning baseline approaches do not utilize a reward machine, and rely on optimizing the task sequence through agent's experience[6],.

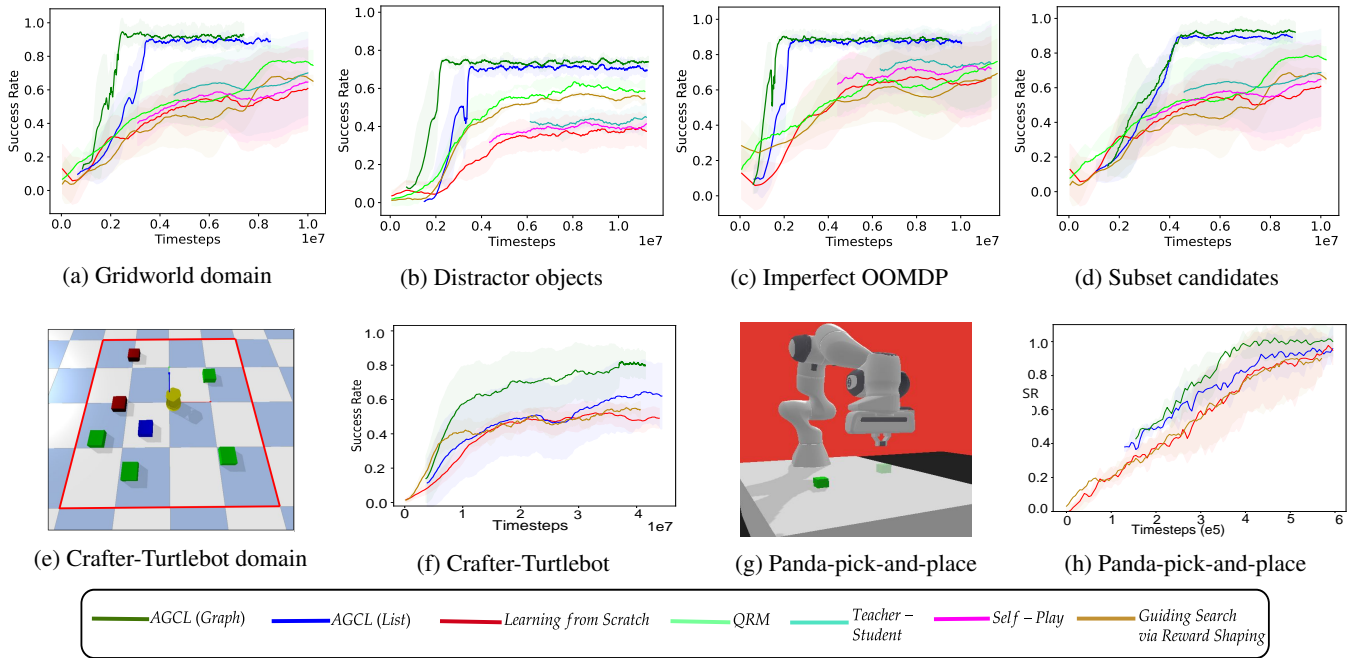
[6]Baseline implementation details in Appendix Section C

Figure 3: Learning curves (Averaged over 10 trials): (a) Gridworld domain; (b) Gridworld - with distractor objects in the environment; (c) Gridworld - with imperfect OOMDP description of the environment; (d) Gridworld - with subset of curriculum candidates. Figs.(e) and (g) show the two robotic environments and their learning curves in (f) and (h)

The learning curve for the curriculum approaches have an offset on the x-axis to account for the interactions used to go through the curriculum before moving on to the target task, signifying *strong transfer* (Taylor and Stone 2009). The results in Fig. 3a show that *AGCL* reaches a successful policy quicker, and the graph-based curriculum proposed by *AGCL* outperforms the sequence-based approach. Learning source tasks in a graph-based curriculum is also parallelized, reducing the overall wall time. Furthermore, our method outperforms the five baseline approaches in terms of learning speed. The other four baseline approaches perform better than learning from scratch, but do not outperform *AGCL*. All approaches may converge to a higher asymptote, but will need more interactions with the environment. For this task, the curriculum generated by *AGCL* has $|\mathcal{V}| = 4$ and $|\mathcal{V}| = 6$ for the sequence-based and graph-based curricula respectively. To demonstrate that the average convergence rate of *AGCL* is consistently higher than the baseline approaches, we perform an unpaired t-test (Kim 2015) to compare *AGCL* against the best performing baselines at the end of 10^7 training interactions and we observed statistically significant results (95% confidence). Thus, *AGCL* not only achieves a better success rate, but also converges faster.[7]

5.2 AGCL on Tasks with Distractor Objects

Next, we test our *AGCL* approach (sequence and graph) on the *pogo-stick* task described in Sec. 5.1 but now the OOMDP description of the environment contains a new class $C(distractor) = \{distractor_{env}, distractor_{inv}\}$,

---
[7]Statistical significance result details in Appendix Section D.

whose object $distractor_{env}$ or $distractor_{inv}$ can assume values in the range $[0, 2]$, signifying that the initial task state or the inventory may contain 0 to 2 instances of the $distractor$ object. The agent interacts with this object the same way it interacts with $trees$ and $rocks$, i.e. by navigating to the object and collecting it in inventory. However, presence of $distractor$ objects in the inventory of the agent is not necessary to achieve the target objectives and is not modeled in the LTL_f specification.

Fig. 3b demonstrates how our proposed *AGCL* performs against the baseline approaches. Since our curriculum generation approach models not only the goal configuration but also the initial task space configuration, it is successful in estimating the difficulty posed by the introduction of the distractor object as they are present in the initial state of the target task. The curriculum generated by *AGCL* guides the agent to a successful policy in fewer interactions compared to the other baselines. Approaches that design a curriculum purely from the LTL_f specification will not model the distractor objects, and will thus fail to determine the complexity of a source task that contains these distractor objects.

5.3 Tasks with Imperfect OOMDP Descriptions

In certain partially observable settings, it might not be possible to obtain an accurate OOMDP description of the environment. To demonstrate the efficacy of our approach in imperfect OOMDP descriptions, we incorporate a gaussian noise over the class parameter value ranges, i.e. if the parameter range is $Par_{given}(world\_size[height]) = [a, b]$, we assume that this range is imperfect, and the true range is given

by $Par_{noisy}(world\_size[height]) = [a - \mathcal{N}(0,\sigma), b + \mathcal{N}(0,\sigma)]$, where $\sigma = (b-a)/6$; covering the entire range of the parameter values in six standard deviations. We incorporate a gaussian noise over all the class parameters ranges.

We test *AGCL* (sequence and graph) on the same task of Sec. 5.1 but now the environment's OOMDP description is noisy over the parameter values, i.e., the exact OOMDP description is unknown. We observe that even with imperfect OOMDP descriptions, *AGCL* converges faster than the baselines (Fig 3c), and the graph-based *AGCL* converges to a successful policy the quickest, in 2.5×10^6 interactions.

5.4 Tasks with Continuous OOMDP Ranges

In the gridworld, the OOMDP parameter values were integers. In this experiment, we test how our approach performs in tasks that have continuous parameter ranges. We modify the function $P : \Omega \to \mathcal{M}$ that given a node in the DFA, samples a subset of MDPs $\mathcal{M}_{sub} \subseteq \mathcal{M}$ such that $|\mathcal{M}_{sub}| = b$. We test this on two challenging simulated robotic environments where the interaction cost is high.

Fig. 3e shows a robotic task with the same objective described in Sec. 5.1. Here the values for the parameters $width, height$ for class $world\_size$ are in the continuous range $[2m, 4m]$. The move forward (backward) action causes the robot to move forward (backward) by $0.1m$ and the robot rotates by $\pi/8$ radians with each rotate action. Additionally, the objects can be placed at continuous locations in this robotic domain as compared to discrete grid locations for the gridworld task. These changes increase the number of MDP and OOMDP states the agent can attain.

The second simulated robotics environment (Fig. 3g) consists of a robotic arm performing a pick-and-place task (Gallouédec et al. 2021) with the LTL_f objective: $\mathbf{F}(g \wedge \mathbf{F}(p \wedge \mathbf{F}q))$ where g, p, q are the atomic propositions for *'reach-object'*, *'pick-up-object'* and *'place-object'*. The OOMDP is modeled as $\mathcal{C} = \{world\_size, objects\}$ with $Par(world\_size) = \{length, width, height\}$ and $Par(objects) = \{no\_objects\}$. The continuous parameters $\{length, width, height\}$ assume values between $[10, 50]cm$. The robot has continuous action parameters for moving the arm and a binary gripper action (close/open). The robotic domains were modeled using PyBullet (Coumans and Bai 2016–2021). For each node, we sample $b = 25$ different MDPs and evaluate *AGCL*. The results in Figs.(3f,3h), show that *AGCL* outperforms all other baselines in both environments. Additionally, the graph-based *AGCL* converges to a successful policy the quickest.

5.5 AGCL on Subset of Curriculum Candidates

In cases where the range of parameter values or the DFA representation is too large, the number of curriculum candidates can grow exponentially in terms of the number of nodes in the DFA, making it infeasible to calculate the jump-score for each curriculum candidate. For this, we evaluate *AGCL* on the task described in Sec. 5.1 by sampling a subset Ψ_{sub} of the total curriculum candidates Ψ, such that $|\Psi_{sub}| = |\Psi|/4$, thus reducing the total number of curriculum candidates and the amount of computation required to determine a suitable curriculum. From the results in Fig. 3d, we observe that in

this experiment, *AGCL* takes more interactions to converge to a desired policy as compared to Fig. 3a, as sampling a subset prunes some desirable curriculum candidates, however it outperforms the other curriculum learning and automaton-guided RL baselines in terms of learning speed.

6 Conclusion and Future Work

We proposed *AGCL*, a framework for curriculum generation using the high-level LTL_f specification and the OOMDP description of the environment. *AGCL* decomposes the target objective into sub-goals, and generates a sequence-based or a graph-based curriculum for the task. Through experiments, we demonstrated that *AGCL* accelerates learning, converging to a desired success rate quicker as compared to other curriculum learning and automaton-guided RL baselines. Moreover, our proposed approach improves learning performance even in the presence of distractor objects in the environment that are not modeled in the LTL_f specification. Finally, we evaluate our approach on long-horizon complex robotic tasks where the state space is large. *AGCL* reduces training time without relying on human-guided dense reward function nor does it require a perfect OOMDP description of the environment. The graph-based curricula produced by *AGCL* affords learning separate behaviors in parallel, reducing wall clock time for complex sequential decision making tasks. Thus, our proposed *AGCL* approach offers accelerated learning when the high-level task objective is available.

We assume that the high-level task objective is available, and can be characterized using a LTL_f formula. However, in certain cases, the target tasks can be completely novel, with no prior information on the task objective. Additionally, it might not be feasible to represent the domain using OOMDP representation, as the objects in the environment can be unknown, and the only way to get more information about the environment is through interaction with the environment. In scenarios where the LTL_f specification becomes complex, the DFA gets larger, and it might become intractable to come up with exact solutions of curriculum optimization. Future work would involve devising approximate solutions that tightly bound exact solutions. Also, our heuristic, the jump score, while it was shown to be useful here, might not perform universally well for all task objectives and environments. In future work, we plan to modify this heuristic to be more theoretically-grounded. Furthermore, we plan to extend it to settings where obtaining an accurate LTL_f specification might be difficult. Finally, we would like to extend our work to multi-agent settings, and investigate curriculum generation using even more expressive high-level languages, such as CTL* and μ-calculus.

Ethical Impact

AGCL aims to boost the learning speed of reinforcement learning agents for sequential decision making tasks. This will enable quicker learning for robotic applications. It will lead to an overall reduction in the training times, saving computation time and energy. On the other hand, this work can also be used for negative applications - e.g., malicious

use of RL. However, the above mentioned concerns are central to all works that deal with any aspects of AI.

Acknowledgements

References

Camacho, A.; Chen, O.; Sanner, S.; and McIlraith, S. A. 2018. Non-Markovian rewards expressed in LTL: Guiding search via reward shaping (extended version). In *GoalsRL, a workshop collocated with ICML/IJCAI/AAMAS*.

Coumans, E.; and Bai, Y. 2016–2021. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

Da Silva, F. L.; and Costa, A. H. R. 2019. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64: 645–703.

De Giacomo, G.; and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI'13 Proc. of the Twenty-Third Intl. joint Conf. on Artificial Intelligence*, 854–860. Association for Computing Machinery.

Diuk, C.; Cohen, A.; and Littman, M. L. 2008. An object-oriented representation for efficient reinforcement learning. In *25th Intl. Conf. on Machine learning*, 240–247.

Foglino, F.; Christakou, C. C.; and Leonetti, M. 2019. An optimization framework for task sequencing in curriculum learning. In *Intl Conf. on Development and Learning*.

Gallouédec, Q.; Cazin, N.; Dellandréa, E.; and Chen, L. 2021. panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning. *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*.

Gao, Y.; and Wu, L. 2021. Efficiently mastering the game of nogo with deep reinforcement learning supported by domain knowledge. *Electronics*, 10(13): 1533.

Icarte, R. T.; Klassen, T.; Valenzano, R.; and McIlraith, S. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Intl. Conf. on Machine Learning*, 2107–2116.

Jiang, Y.; Bharadwaj, S.; Wu, B.; Shah, R.; Topcu, U.; and Stone, P. 2021. Temporal-logic-based reward shaping for continuing RL tasks. In *AAAI Conf. on AI*.

Jothimurugan, K.; Bansal, S.; Bastani, O.; and Alur, R. 2021. Compositional reinforcement learning from logical specifications. *Neural Information Processing Systems*.

Karpathy, A.; and Van De Panne, M. 2012. Curriculum learning for motor skills. In *Canadian Conf. on AI*.

Kim, T. K. 2015. T test as a parametric statistic. *Korean journal of anesthesiology*, 68(6): 540–546.

Kurkcu, A.; Campolo, D.; and Tee, K. P. 2020. Autonomous Curriculum Generation for Self-Learning Agents. In *16th Intl Conf. on Control, Automation, Robotics and Vision*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In *ICLR (Poster)*.

Matiisen, T.; Oliver, A.; Cohen, T.; and Schulman, J. 2020. Teacher-Student Curriculum Learning. *IEEE Trans. Neural Networks Learn. Syst.*, 31(9): 3732–3740.

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.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Narvekar, S.; Peng, B.; Leonetti, M.; Sinapov, J.; Taylor, M. E.; and Stone, P. 2020. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *JMLR*, 21: 1–50.

Narvekar, S.; Sinapov, J.; and Stone, P. 2017. Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning. In *IJCAI*, 2536–2542.

Pocius, R.; Isele, D.; Roberts, M.; and Aha, D. 2018. Comparing reward shaping, visual hints, and curriculum learning. In *AAAI Conf. on Artificial Intelligence*, volume 32.

Qiao, Z.; Muelling, K.; Dolan, J. M.; Palanisamy, P.; and Mudalige, P. 2018. Automatically generated curriculum based reinforcement learning for autonomous vehicles in urban environment. In *IEEE Intelligent Vehicles Symposium*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Opti Algo. *CoRR*.

Shukla, Y.; Thierauf, C.; Hosseini, R.; Tatiya, G.; and Sinapov, J. 2022. ACuTE: Automatic Curriculum Transfer from Simple to Complex Environments. In *21st Intl. Conf. on Autonomous Agents and Multiagent Systems*, 1192–1200.

Silva, F. L. D.; and Costa, A. H. R. 2018. Object-oriented curriculum generation for reinforcement learning. In *17th Intl Conf. on Autonomous Agents and MultiAgent Systems*.

Sukhbaatar, S.; Lin, Z.; Kostrikov, I.; Synnaeve, G.; Szlam, A.; and Fergus, R. 2018. Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. In *ICLR*.

Svetlik, M.; Leonetti, M.; Sinapov, J.; Shah, R.; Walker, N.; and Stone, P. 2017. Automatic curriculum graph generation for reinforcement learning agents. In *AAAI Conf. on AI*.

Taylor, M. E.; and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *JMLR*, 10(7).

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2018. Teaching multiple tasks to an RL agent using LTL. In *Autonomous Agents and MultiAgent Systems*.

Velasquez, A.; Bissey, B.; Barak, L.; Beckus, A.; Alkhouri, I.; Melcer, D.; and Atia, G. 2021. Dynamic automaton-guided reward shaping for monte carlo tree search. In *Proc. of the AAAI Conf. on Artificial Intelligence*, 13.

Visús, Á.; García, J.; and Fernández, F. 2021. A taxonomy of similarity metrics for markov decision processes. *arXiv preprint arXiv:2103.04706*.

Weinshall, D.; Cohen, G.; and Amir, D. 2018. Curriculum learning by transfer learning: Theory and experiments with deep networks. In *Intl. Conf. on Machine Learning*.