Automating Curriculum Learning for Reinforcement Learning using a Skill-Based Bayesian Network

Vincent Hsiao
NRC Postdoctoral Fellow,
Naval Research Laboratory
Washington DC, United States
vincent.hsiao.ctr@us.navy.mil

Mark Roberts
Naval Research Laboratory
Washington DC, United States
mark.c.roberts20.civ@us.navy.mil

Laura M. Hiatt
Naval Research Laboratory
Washington DC, United States
laura.m.hiatt.civ@us.navy.mil

George Konidaris
Brown University
Providence RI, United States
gdk@brown.edu

Dana Nau
University of Maryland
College Park, MD, United States
nau@umd.edu

ABSTRACT

A major challenge for reinforcement learning is automatically generating curricula to reduce training time or improve performance in some target task. We introduce SEBNs (Skill-Environment Bayesian Networks) which model a probabilistic relationship between a set of skills, a set of goals that relate to the reward structure, and a set of environment features to predict policy performance on (possibly unseen) tasks. We develop an algorithm that uses the inferred estimates of agent success from an SEBN to weigh the possible next tasks by expected improvement. We evaluate the benefit of the resulting curriculum on three environments: a discrete gridworld, continuous control, and simulated robotics. The results show that SEBN-based curricula frequently outperform other baselines.

KEYWORDS

Bayesian Networks, Automated Curriculum Generation, Reinforcement Learning, Transfer Learning

ACM Reference Format:
Vincent Hsiao, Mark Roberts, Laura M. Hiatt, George Konidaris, and Dana Nau. 2025. Automating Curriculum Learning for Reinforcement Learning using a Skill-Based Bayesian Network . In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 9 pages.

1 INTRODUCTION

Adapting skills to new or unseen tasks is a major challenge in Reinforcement Learning (RL). Curriculum Learning [5, 25], an approach for training agents using a sequence of increasingly difficult environments, often promotes the effective development of more robust policies. But customizing a curriculum can require substantial human insight, especially in robotics, where the environment or tasks that need to be performed can change frequently. An ideal solution to this problem would be an automated curriculum that enables the robot to discern for itself when it needs to adapt, how long should train, and in what environments.

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

Past work on automated curriculum generation such as [20, 33] has primarily focused on choosing what skills to train while holding the environment itself static. More recent approaches that build a curriculum over different environments such as [27] do not consider agent skill competencies. Furthermore, these environment-based approaches require explicit evaluation on an environment before being able to calculate an estimate of agent success or regret to add those environments to the curriculum.

We address the aforementioned issues by introducing Skill Environment Bayesian Networks (SEBNs) as a potential method for estimating agent competency level and selecting the most appropriate environments for training. SEBNs model a probabilistic relationship between these goals, skill competencies, and environment features using data from *past* rollouts. Competencies can be explicit or latent. The SEBN can estimate agent success rates on new, possibly unseen, environments. We use these estimates to select the next set of training tasks within a curriculum in what we call an SEBN-guided automated curriculum. Importantly, the SEBN does not require explicit evaluation on each possible environment to estimate agent success.

The contributions of the paper include: (1) Introducing and formalizing the SEBN for skills, task features, and reward structure; (2) Providing an algorithm for constructing curricula using SEBNs; (3) Introducing Megagrid, a gridworld environment that simplifies generating partially-specified environments for transfer learning; (4) Assessing SEBN-based curricula on three distinct environments: a discrete gridworld (DoorKey), continuous control (BipedalWalker), and a difficult simulated robotics domain (robosuite); and (5) Demonstrating that SEBN-based curricula produce more robust policies: they reach success more quickly than policies trained with other curricula in the continuous control and robotics environments, and they perform comparably in the gridworld environment.

2 BACKGROUND AND PRELIMINARIES

Bayesian statistics rely on some sort of informed prior, provided or learned, to estimate the future values. In an SEBN, part of this prior is provided in the form of the network and in the strength of relationships, and part of the prior is learned through the collection of samples from the environment. We next provide our motivation for this Bayesian approach (§2.1) with some background on Bayesian

Networks (§2.2). We then formalize the curriculum learning problem (§2.3), how we use task features to construct tasks (§2.4), and an extension to the options framework (§2.5).

2.1 Evidence Centered Design

Our motivation for using a Bayesian Network to estimate learning proficiency comes from the method of Evidence Centered Design (ECD) [23], a technique used in human educational assessments. In Evidence-Centered Design, statistical models, such as Bayesian networks, are used to measure the proficiency levels of a given student. These proficiency measurements are then used to inform task and assessment creation. For example, ECD could be used to help model and analyze the performance of a tennis player. The Bayesian network in this domain can include nodes that represent latent competencies (e.g., mobility, footwork, dynamic vision, etc.) and nodes that represent observable metrics (e.g., number of successful serves, return rate, game score). The performance of a tennis player on the observable metrics is used to infer their latent competencies. New training goals can then be set using these estimated competencies. This technique is effective in human educational contexts, and we hypothesize that a similar approach could be applied to assist in designing a curriculum to improve learning in robotic agents.

2.2 Bayesian Networks (BNs)

Bayesian Networks (BNs) [30] are a type of graphical model that provide an efficient way to represent and reason about probabilistic relationships among a set of random variables. A Bayesian Network (X, D, Φ) is defined by a set of variables X, their corresponding domains D, and a set of parent functions Φ that specify the conditional probability distributions of each variable given its parents. When D is discrete, these parent functions are typically specified in a tabular format known as Conditional Probability Tables (CPTs).

It is common to use BNs to model relationships between latent and observable variables. Once constructed, the network can be used to infer latent values from observed values. New data can be entered in the form of evidence values for observed variables in a BN. The probabilities over other variables in the network are calculated by conditioning on this observed evidence, i.e., as a conditional probability: $P(X_1|X_2, ..., X_N)$. We will employ a standard bucket elimination algorithm (aka variable elimination) [8, 9] to perform inference. In this paper, the observable variables of the BN relate to the environment of an agent and a target performance it is attempting to achieve; both are modeled as a task in an MDP problem and defined in §2.3. The unobservable variables will relate to a set of latent competencies that we define in §3.1.

2.3 Curriculum Learning

We adapt the notation of Narvekar et al. [25] to describe a *task* as the interaction of an agent with its environment to meet some objective. A task, formalized as an episodic Markov Decision Process (MDP), is a tuple $M = (\mathcal{S}, \mathcal{A}, p, r)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $p(s'|s, a)$ gives the probability of being in state s' after taking action a in state s, and $r(s, a, s')$ is the reward function of transitioning to s' after taking action a in s. A solution to M is a policy π that maximizes the cumulative sum of rewards for an episode of length T, i.e., $\sum_{t=1}^{T} R_t$.

Let $\mathcal{T}$ be a set of all tasks an agent could complete in M, where a task $m_i \in \mathcal{T}$ is a task-specific MDP $m_i = (\mathcal{S}_i, \mathcal{A}_i, p_i, r_i)$. For all tasks in $\mathcal{T}$, let $\mathcal{D}^{\mathcal{T}}$ be the set of all possible transition samples from $\mathcal{T}$ (see Narvekar et al. [25] for a complete definition). In their formalism, a Curriculum $C = (\mathcal{V}, \mathcal{E}, \theta, \mathcal{T})$ is a directed acyclic graph, where $\mathcal{V}$ is the set of vertices, $\mathcal{E} \subseteq \{(x, y)|(x, y) \in \mathcal{V} \times \mathcal{V} \wedge x \neq y\}$ is a set of directed edges, $\theta : \mathcal{V} \rightarrow \mathcal{P}(\mathcal{D}^{\mathcal{T}})$ is a function that associates samples within each vertex, and $\mathcal{P}(\mathcal{D}^{\mathcal{T}})$ is the powerset of $\mathcal{D}^{\mathcal{T}}$.

In this paper, we develop what Narvekar *et al.* [25] call a task-level curriculum, where each vertex $v \in \mathcal{V}$ is associated with samples from a single task in $\mathcal{T}$. That is, the mapping function for task m_i is $\theta : \mathcal{V} \rightarrow \{\mathcal{D}_i^{\mathcal{T}}|m_i \in \mathcal{T}\}$. For convenience, we will refer to a task's available samples at vertex v as m_i^{θ}. In other words, a task descriptor θ_i is used to construct task m_i, and a curriculum is a sequence of tasks m_1, m_2, ..., m_{target} up to some target task.

Before we describe how we construct this function using task features, we point out some deviations from the model just described. The curriculum being a DAG is a very strong assumption and is not true of the SEBN-guided curriculum. While the episodic MDP model of Narvekar et al. provides a more comprehensible model of curriculum learning, the RL algorithms of this paper actually learn with a discount factor γ, and one could argue that the Partially Observable MDP might be more appropriate. Both changes would be extensions to the simplified MDP model presented here.

2.4 Task Descriptors (Env't+Target Features)

We will use *task features* to define θ for a task m_i^{θ}. This is a common approach to quantify potential transfer between two tasks (e.g., [16, 18, 26, 31]). The notion is that two tasks that share similar features will exhibit better transfer. We adopt a task descriptor similar to Rostami et al. [31] and Narvekar et al. [26].

Specifically, we parameterize θ with a vector that consists of set of environment-specific features E and a set of one or more performance targets K. Thus, $\theta((\mathbb{Z}_0)^{|E|}(\mathbb{Z}_0)^{|K|})$ will indicate the specific task m_i^{θ}. We will often omit the task descriptor for clarity and just reference m_i. Note that the task descriptor is underspecified with respect to the environment, so one configuration of θ represents a class of different environments an agent can encounter.

Example Task Descriptors using Bipedal Walker. The Bipedal Walker (BPW) benchmark [38] involves two-legged agent moving through terrain in a 2D environment. Figure 1 shows some example terrains. The top portion Figure 2 shows E and K for the bipedal walker. Here, E consists of five features that control the difficulty of the environment, and there is a single target of moving to the right by at least 30 steps (The dashed latent competencies are defined in §3.1). The task descriptor for this BPW is $\theta_{(\text{pit-gap, stump-height, stair-width, stair-steps, roughness, moved>30})}$. A task m_i^{θ} for BPW involves a particular setting of the parameters for θ. ■

2.5 Extending Targets to Include Options

One could imagine a richer set of targets in K, even ones that are hierarchically organized as a decomposition of subtasks. The options framework [35, 36] is a common model for such situations. Briefly, a subtask j for task m_i is an option $o_j = (\mathcal{S}_j, \pi_j, \text{TERM}_j)$, where $\mathcal{S}_j \subseteq \mathcal{S}_i$ are the starting states of the option, π_j is used to

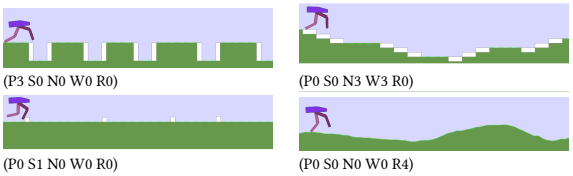

Figure 1: Challenge environments for BipedalWalker with corresponding descriptors (P:pit gap, S:stump height, W:stair width, N:stair steps, and R:ground roughness).

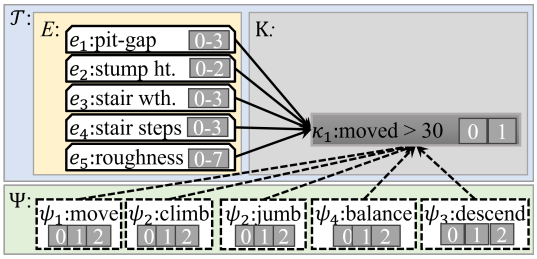

Figure 2: The SEBN for the Bipedal Walker environment.

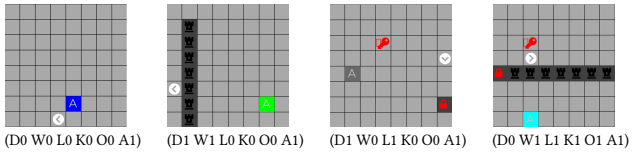

Figure 3: Example environments for DoorKey with corresponding environment features (D:distance, W:wall, L:locked door) and target features (K:key, O:opened, A:at).

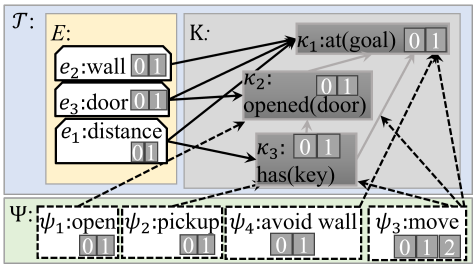

Figure 4: The SEBN for the DoorKey environment.

take action while the option is enabled, and $\text{TERM}_j : \mathcal{S}_i \rightarrow \{0, 1\}$ is a function that indicates the option has terminated.

For a specific task m_i, a subtask $m_i^j = (\mathcal{S}_j, \mathcal{A}_i, p_i, r_j)$ indicates that an option has a specific context: it works over a set of states $\mathcal{S}_j$ that are a subset of the tasks states $\mathcal{S}_i$, it uses a specific reward independent of the task reward, and it has the same actions and transitions as the original task m_i. Each option j is enabled as part of the feature parameters for θ (i.e., $(\mathbb{Z}_0)^{|K|}$)).

Example Task Descriptors using DoorKey. Suppose we want an agent to learn to navigate in a gridworld environment to a goal while opening locked doors. Fig. 3 shows several possible environments for this agent and their corresponding environment feature vector. In the easiest environment (top left), the agent (the white arrow) starts very near the goal ("A") in an empty grid. Adding additional obstacles such as a wall, shown as chess rooks, or a locked door, shown as a lock, with a key to unlock it, adjusts the environmental features accordingly. The first three components of the task descriptor θ indicate whether the distance (D) of agent starts near (within 2 squares) of any point of interest (key or goal), the presence of a wall (W), and the presence of a locked door (L).

DoorKey also enables the use of options. The top right part of Figure 4 shows a network of targets K for this problem, corresponding to ordered subtasks. This problem has three options (at(goal), opened(door), and has(key)), each with its own reward. A distinct policy is learned for each of these options. The last three components of θ indicate which of these three subgoals are enabled for a task: getting a key (K), opening a door (O), or being at (A) a cell. ■

3 BAYESIAN CURRICULUM LEARNING

The key idea in this section is to use a BN to estimate performance on K over the environmental features from E plus a set of estimated proficiency Ψ on latent competencies, which are hidden or unobserved. Before we formally define the SEBN, we describe extend the DoorKey example to discuss this process.

Example of Latent Competencies using DoorKey. Suppose we have collected the agent's past performance on different environments in E. For example, say we ran our agent on the empty-grid (D0 W0 L0), wall-only (D1 W1 L0), and door-only (D1 W0 L1) environments and recorded that the agent was successful on the empty-grid and door-only environments but not the wall-only environment. This failure might be due to the agent not yet knowing how to navigate around walls. We can think of this ability as a latent "avoid wall" competency that an agent needs to have mastered to solve tasks that require it. Furthermore, using the notion that there is this latent competency, we can easily predict that the agent will fail on the wall-and-door (D0 W1 L1) environment without having any data of the agent's performance on that specific type of environment. ■

The bottom row of Fig. 4 shows a set of latent competencies Ψ. In this example, we chose four latent competencies: (move, pick up, avoid wall, open) that we expect the agent to need to master to successfully solve different tasks. These latent variables are provided by an expert, similar to Abel et al. [1].

We can use the SEBN to predict two important quantities. First, when faced with a new, possibly unseen, task $m_i \in \mathcal{T}$, we need to estimate the proficiency of each competency in Ψ. This is important because competencies will advance at different rates and some tasks will require more proficiency than others for specific competencies.

Second, when faced with a new, possibly unseen, task, we need to be able to predict performance on $k_j \in K$ given the current estimates of competency level from the first step. This is important because it can be used to select from a set of candidate tasks for training in the next iteration of a curriculum.

Returning to Fig. 4, suppose a new task is defined over E and K. The proficiency estimate(s) can be calculated using the links to the

bottom row. Once these estimates are provided back to the network, the expected reward can be calculated in the target layer K.

3.1 Competencies (Latent or Explicit)

As with the "avoid wall" latent competency for DoorKey, we propose that there is some shared latent set of competencies of which mastery over can predict an agent's success rate on different metrics in different environments. More concretely, let $\kappa_i \in K$ be a set of observable metrics, which can be any measurable target (e.g., a standard reward function, a shaped or partitioned reward function, or the completion an option). For example, we can define a binary variable that is 1 if an agent received a reward of more than a threshold value, and 0 otherwise.

Drawing inspiration from ECD, we propose that there exists a set of latent competencies Ψ that are not directly observable but can be inferred from the observed metrics. These competencies are such that the probability of success on a given observable metric κ_i of an agent on a specific task m_i is dependent on sufficient mastery of the corresponding latent competencies. This relationship allows us estimate the impact of competencies on unseen tasks. A BN allows us to model this relationship, estimate competency levels from data, and subsequently estimate success rates on different environments.

Explicit competencies can be derived from techniques that decompose a task into subtasks. For example, in hierarchical planning, a method decomposes an abstract task [11, 12]. A set of such methods could be used to construct the competencies. For example, Patra et al. [28, 29] have used hierarchical goal networks to decompose tasks and train RL policies. They call the resulting policies goal skills. For the SEBN defined in Fig. 4, all four of the competencies in Ψ could be defined as an explicit goal network. For this SEBN, the dependencies in the network are exactly the same as a corresponding hierarchical goal skill network. Consequently, we could take any problem with a heirarchical goal skill network, define environment-conditioned dependencies, and convert it to a SEBN.

The flexibility of letting competencies be latent or explicit allows us to model environments where intermediate decompositions may not be well-defined. In the absence of an easy way to check if an agent satisfies a goal for a given goal-skill, then it can be set to be a latent competency in the SEBN.

3.2 Skill Environment Bayesian Network

We can now define a Skill Environment Bayesian Network (SEBN) for estimating the competency level of an agent and modeling the relationship between agent competency levels, env features, and observable goals/metrics. The SEBN is a tuple $\{X, D, \Phi\}$, where the variable set $X = E \cup K \cup \Psi$ is split into three sets of variables:

Environment Variables. E is a set of variables that represent the features of an environment descriptor. Each variable in this set corresponds to a specific feature of the environment; thus the domain of a given variable is the set of possible values the corresponding environment feature can take. In the DoorKey example, E consists of features for wall, door, and distance.

Target Variables. K is set of variables that directly correspond one or more targets. If there is a single policy, then K will have a single node, as in Figure 2. But if there are options available to the agent, then each $\kappa_j \in K$ corresponds to the option for task m_i^j (cf. §2.5). These variables then provide estimates of the value of executing that option in the current environment. For the purposes of this paper, that estimate is thresholded such that each variable returns a boolean value corresponding to whether its estimate meets a performance threshold, which roughly corresponds to an estimate of whether the option will succeed or fail.

Competency Variables. Ψ contains the set of variables that represent the competency levels of an agent. Each variable in this set denotes the level of proficiency an agent has in a particular competency and takes values in a range from $\{0, 1, ..., N\}$ where N is the highest proficiency level for a given competency. For this paper, we will define competency with two or three levels of proficiency. Competency levels are roughly ordered, as provided in a set of requirement specifications, by a human expert. The rationale for writing these specifications is to convey whether a given environment requires sufficient proficiency in several competencies. Specifications follow roughly ordered values of competency (e.g. a higher "move" competency should enable harder tasks). The competencies can be latent (e.g., capturing whether an agent avoids obstacles while moving) or explicitly learnable (cf. §3.1).

The parent functions Φ provide the distribution of possible values of each variable conditioned on their parent variables. To construct these parent functions, we specify a list of competency requirements that is procedurally used to construct the corresponding CPTs. We provide specific detail about this process in §3.3.

Once constructed, we can use the SEBN to estimate an agent's competency levels and determine what environments or competencies the agent should focus on learning next. To do this, we must estimate two quantities for a task $m_i \in \mathcal{T}$: **Competency Level:** $P(\Psi = \psi | \kappa_i = r, \mathcal{T} = m_i)$ - the probability that an agent has a competency level of ψ, given its prior performance of at least reward r on target κ_i for task m_i. **Expected task reward:** $P(\kappa_i = r | \mathcal{T} = m_i, \Psi = \psi)$ - the probability that an agent can achieve a reward of at least r for target κ_i conditioned on task m_i with a given competency level ψ.

We will estimate the competency levels using past rollouts. We will then apply the estimates of competency levels to estimate the expected task reward over a collection of candidate environments $m_i \in \mathcal{T}$. These estimated probabilities will be used to determine which environments the agent should train on next.

3.3 Defining variable distributions

To complete our network definition, we need to define the distributions of each variable in the network. The distribution of variables in Ψ and K are defined using a hierarchical structure. We start by defining the leaves of this structure which are located in Ψ.

Defining Ψ. Consider the "move" competency $move \in \Psi$ in the gridworld navigation environment. In Fig. 4, we define *move* as having three levels of proficiency $\{0, 1, 2\}$, associated with a corresponding parameter set $\phi_{move} = \{0.8, 0.2, 0.0\}$ such that the probability of the "move" at competency level j is given by: $P(move = j) = \phi_{move}(j)$. In this case, the parameters denote that currently the agent has a *move* competency of 80% probability for no mastery and a 20% probability of having level one mastery.

More generally, Ψ can have a hierarchical structure and there can be latent competencies within Ψ that depend on other latent

competencies. To allow for these hierarchies in Ψ, we define B to be a subset of Ψ which represents a base set of competencies (the leaves). The distribution of these base competencies is determined by a set of associated parameters ϕ_B.

Defining K. The variables in K represent the success of an agent on a given target. In an SEBN, we seek to model the relationship where the success probability of an agent on a variable $\kappa_i \in K$ is dependent on three types of parent variables: (1) the environment features $e_i \in E$ relevant to κ_i (2) the agent's current competency levels $\psi_i \in \Psi$ (3) other targets $\kappa'_i \in K$ This means that variables in K depend on other variables in Ψ and K as well as a set of variables in E. The success of a task depends on the agent's mastery of the competencies required for a given environment configuration and an independent failure rate λ. A task succeeds with a rate of $(1-\lambda)$, if all sub-requirements in K and Ψ are satisfied.

More concretely, for each relevant environment configuration e_i in relation to a goal variable $\kappa_i \in K$, we define a set of competency level requirements R_{κ_i} that an agent must master to successfully complete the task of k_i on the environment e_i. For example, in Fig. 4, *haskey* depends on the *distance* environment feature and the latent *move* and *pickup* competencies. Suppose we define the following competency level requirements for the *haskey* node:

haskey: (distance=0 | move=1, pick up = 1)
haskey: (distance=1 | move=2, pick up = 1)

These state that if the key is close (distance=0), the agent needs a level of proficiency of 1 in the move and pick up competencies to successfully get the key. However, if the key is far (distance=1), the agent needs a higher level of proficiency in the move competency (move=2). The agent should have a high chance of success if it meets all necessary requirements and a high chance of failure otherwise.

We directly translate these requirements into entries in the corresponding CPTs in the following way. For the first competency level (distance=0), we have that:

$$P(haskey = 1 | distance = 0, move = 0, pickup = 0) = \lambda$$
$$P(haskey = 1 | distance = 0, move = 0, pickup = 1) = \lambda$$
$$P(haskey = 1 | distance = 0, move = 1, pickup = 0) = \lambda$$
$$P(haskey = 1 | distance = 0, move >= 1, pickup >= 1) = (1 - \lambda)$$

For the next competency level (distance=1), we have that:

$$P(haskey = 1 | distance = 1, move = 0, pickup = 0) = \lambda$$
$$P(haskey = 1 | distance = 1, move = 0, pickup = 1) = \lambda$$
$$P(haskey = 1 | distance = 1, move = 1, pickup = 0) = \lambda$$
$$P(haskey = 1 | distance = 1, move = 1, pickup = 1) = \lambda$$
$$P(haskey = 1 | distance = 1, move >= 2, pickup >= 1) = (1 - \lambda)$$

These state that the agent has a success rate of $(1-\lambda)$ for a given goal for an environment setting if it satisfies all necessary requirements and a success rate of λ if there is one or more requirement missing.

For the environment features set E, the distribution of variables in this set is fully controlled for the purpose of curriculum generation. Therefore, the data (samples obtained from rollouts) can be used as the distribution for variables in E.

3.4 Curriculum through Inference

The general algorithm to generate an SEBN-guided automated curriculum is as follows. First we define a generation of the algorithm as L rollouts. Let $\mathcal{T}$ be the set of possible tasks and $P_{\mathcal{T}}(m_i)_t$ be the probability that task $m_i = (e_i, \kappa_i)$ is chosen in the current generation t. We first initialize $P_{\mathcal{T}}(m_i)_0$ to some initial task distribution. This initial weighting can be biased towards easier tasks or can be set to a uniform distribution for better initial competency estimation. Let $\Phi_B = \{..., \phi_{B_i}, ...\}$ be the set of parameters associated with the base competencies B in the SEBN. For each generation, we take the following steps:

(1) Sample L rollouts $(m_i = (e_i, \kappa_i) \sim P_{\mathcal{T}}, o_i)$. For each rollout, we record a set of observable metrics κ_i.
(2) Solve the MLE problem:
$$\Phi^*_B = \arg\max_{\Phi_B} \prod_L P(E = e_i, K = \kappa_i) \quad (1)$$
for Φ^*_B, the values of the parameters for the base competencies, which is an estimate of agent proficiency level in those competencies.
(3) Update the task distribution for the next generation: $P_{\mathcal{T}}(m_i) = F(P_{\Phi^*_B}(\kappa_i | m_i))$ where F is some function that maps the estimated success rate of an environment $P_{\Phi^*_B}(\kappa_i | m_i)$ to a probability distribution.

For our work, we define F using the following fitness function:

$$F(m_i)_t = (P_{\Phi^*_B}(\kappa_i | m_i)_t - P_{\Phi^*_B}(\kappa_i | m_i)_{t-1})^2$$
$$P_{\mathcal{T}}(m_i)_{t+1} = 0.5 \cdot \frac{F(m_i)}{\sum_{m_i} F(m_i)} + 0.5 \cdot P_{\mathcal{T}}(m_i)_t. \quad (2)$$

In [33], it was proposed that curricula should focus on problems where the agent improves the most or has the most expected *improvement in competence*. To simulate this in our approach, we choose a fitness function where the fitness of the environment is a function of the difference between the current estimated success rate and the estimated success rate on the last generation's SEBN. We also add a smoothing factor to improve learning stability.

Note that our curriculum is agnostic to the choice of learning algorithm and policy which can be assumed to be black boxes. The curriculum only requires observations of rollouts and not the internal reward structure of a given policy.

Algorithm 1: SEBN-guided Automated Curriculum
Input: Initial tasks $P_{\mathcal{T}_0}$, Generation size L, SEBN (X, D, Φ)
Initialize: Initialize policy π

while *not converged* **do**
  Sample L environments $m_i = (e_i, \kappa_i) \sim P_{\mathcal{T}_0}$
  for $i = 1 \rightarrow L$ **do**
    Collect rollout data (m_i, κ_i) while training policy π
  Solve the MLE problem (Equation 1) for estimated competency level Φ^*_B on SEBN (X, D, Φ)
  For candidate environments, estimate agent success rate $P_{\Phi^*_B}(\kappa_i | m_i)$ using SEBN with Φ updated with Φ^*_B
  For each candidate environment m_i, update $P_{\mathcal{T}}(m_i)_{t+1}$ using expected improvement weighting of $P_{\Phi^*_B}(\kappa_i | m_i)$ (Equation 2)

3.5 Candidate Selection

On a domain such as Megagrid, we can evaluate $P_{\Phi^*_B}(\kappa_i | m_i)$ for every combination of environmental features. However, calculating $P_{\Phi^*_B}(\kappa_i | m_i)$ for all possible environments in BipedalWalker took too much time at the end of each rollout generation. In general, for domains with a large amount of environmental features, it is impractical to evaluate $P_{\Phi^*_B}(\kappa_i | m_i)$ for every single task $m_i \in \mathcal{T}$.

One way to solve this computational problem is to search in the space of possible environment configurations and only update the distribution of the most promising environments. For our search procedure, we adapt a greedy sample-search procedure based on KL-Search from [14]. This algorithm employs a heuristic search along variables in a Bayesian network to minimize a KL distance heuristic between two networks. By modifying algorithm to instead search for nodes with maximum differences between the current generation's SEBN and the next generation's SEBN updated with Φ^*_B, we can find environments where the estimated success rate changes the most. This modification produces a tree search algorithm that selects nodes in the OR-tree corresponding to a given SEBN where the difference $(P_{\Phi^*_{s_i}}(\kappa_i|m_i)_t - P_{\Phi^*_{s_i}}(\kappa_i|m_i)_{t-1})$ is greatest.

Given a partial configuration X, we use the following heuristic:

$$h(X_1{=}0) = |\log(P_{t-1}(X)) - \log(P_t(X))| \cdot P_t(X) \tag{3}$$

where $P_t(X_1{=}0) = P_{\Phi^*_{s_i}}(\kappa_i|X1)_t$. We use Weighted Mini-bucket Elimination [22] with an ibound of 20 to estimate SEBN probabilities when exact inference is too computationally expensive. Once we select $N = 20$ nodes on the OR-tree using this method, we perform the same calculations from Eq. 2 over the 20 selected environment configurations to define our curriculum for the next generation.

4 EXPERIMENTAL EVALUATION

To demonstrate the effect of our proposed curriculum learning approach, we evaluate the SEBN curriculum on three environments. **DoorKey:** A MiniGrid [7] inspired domain with explicit intermediate goal skills. In this domain, the agent must learn to navigate through a grid-based environment to reach a goal location, while also learning to achieve intermediate goals along the way. **BipedalWalkerHardcore:** a simulated bipedal robot must learn to walk forward as quickly as possible while maintaining balance. The bipedal robot can encounter a variety of obstacles such as rough terrain, stumps, pits and stairs that need robust policies. **Robosuite:** a robotic arm (Kinova Gen 3) must learn to open a door with different weight and latch settings.

In each of these domains, we compare the performance of reinforcement learning agents trained with and without our proposed SEBN-guided curriculum as well as two additional controls: **Uniform curriculum (or Domain Randomization [37]):** all environments have an equal probability of being selected. **Anti-curriculum:** the probability difference in Eq. 2 is replaced with (1 - difference) and we use a min priority queue for candidate selection during the sample-search algorithm.

We evaluate the agents on their ability to learn effective policies that can achieve high rewards in each domain, as well as their ability to generalize to new tasks and environments[1]. Our results show that the SEBN-guided curriculum consistently improves the performance of reinforcement learning agents across all three domains. All runs are performed on an AMD EPYC 7H12 64 core CPU with networks being handled on an A100 GPU.

DoorKey. We start with a gridworld environment named Megagrid inspired by MiniGrid [7]; we reimplemented this standard gridworld environment to enable easier generation of environments

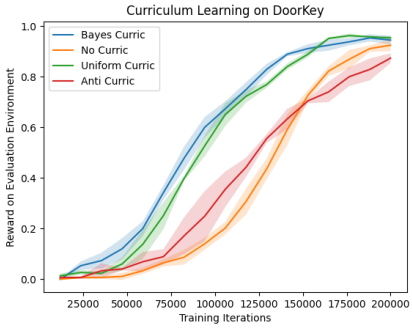

Figure 5: Result of employing a SEBN-guided automated curriculum on the DoorKey environment.

using the task descriptor[2]. We evaluate on the simple DoorKey environment to evaluate the effectiveness of our curriculum learning approach when combined with explicit goal-skills. We use the SEBN shown in Fig. 4 which has the following variables:

- E: includes a wall feature $\{0, 1\}$, door feature $\{0, 1\}$, and a distance feature $\{0, 1\}$. A selection of the different environments that can be experience by an agent can seen in Fig. 3.
- Ψ: includes "move to" $\{0, 1, 2\}$, "pick up" $\{0, 1\}$, "avoid wall" $\{0, 1\}$, "drop" $\{0, 1\}$, and "open door" $\{0, 1\}$.
- K: includes three options: "at(goal)" $\{0, 1\}$, "opened(door)" $\{0, 1\}$, and "has(key)" $\{0, 1\}$ each trained with a PPO policy.

Observations of the environment are provided as partially observed cardinal direction data, following the sensor convention for the Lightworld domain in [18]. We assume that the agent has four cardinal sensors for each item. For example, in the rightmost environment of Fig. 3, the agent would receive the observation that there is a key one step above it, a wall one step below it, and the goal 4 steps below it (0.875 key - up, 0.875 wall - down, 0.5 goal - down). An agent gets a reward of 1 if it reaches the goal square or completes any intermediate goals required (e.g. picking up a key or opening a door). There is no step penalty but the episode is automatically terminated if no progress has been made in 50 steps.

We train the policies using an Actor-Critic architecture [34] trained using Proximal Policy Optimization (PPO) [32]. The policy and value networks each have four hidden layers which are used to calculate their corresponding outputs.

The evaluation curve of our policies can be seen in Fig. 5. This evaluation is performed on the hardest environment (rightmost environment in Fig. 3). Because this environment is relatively simple, it is easy for policies learned without a curriculum to achieve a high success rate. However, there is a noticeable jumpstart where the SEBN-guided curriculum provides a gain in learning efficiency.

We performed a generalization experiment where the learned policies are transferred to an evaluation on an *unseen* and much larger 32 x 32 grid environment. The policies learned using the SEBN-guided curriculum are more robust to this change in grid size succeeded at an average rate of 93 percent compared to only 82 percent without a curriculum.

[1]Supplemental figures can be found at https://arxiv.org/abs/2502.15662

[2]Please send an email to Mark Roberts to request the Megagrid code

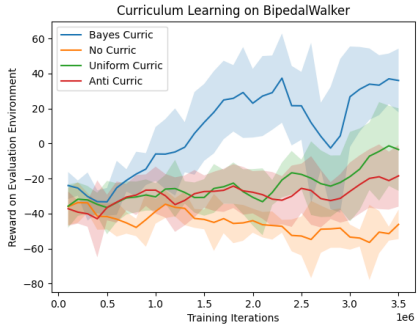

Figure 6: SEBN-guided automated curriculum on BipedalWalker. Evaluation environments are randomly generated within a given environment feature set.

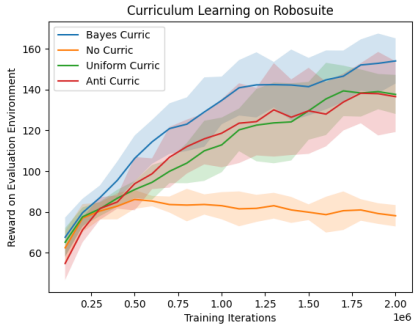

Figure 7: SEBN-guided automated curriculum on the Robosuite Door environment. 15 policies are learned for each line and evaluated on the hardest (mass=6, latch=1) environment.

BipedalWalker Hardcore. For the next domain, we evaluate our SEBN-guided curriculum on BipedalWalker (BPW), a continuous control environment. We employ a modified version of the BipedalWalkerHardcore environment by Parker-Holder et al. [27] to suit a limited computational budget. We define the following SEBN:

- E: includes five design parameters: ground roughness $\{0-7\}$, pit gap $\{0-3\}$, stump height $\{0-2\}$, stair width $\{0-3\}$, and stair steps $\{0-3\}$. Since there are too many different environments to perform exact inference, it is necessary to use a sample-search procedure to select candidate environments.
- Ψ: includes "move", "climb", "jump", "balance", "descend". All with proficiency levels $\{0, 1, 2\}$.
- K: includes one observable metric $\{0, 1\}$: whether the agent has traveled a distance of 30 units ($\approx$1/3rd of the level).

The observation of the agent consists of internal sensor measurements such as (hull angle speed, angular velocity, horizontal velocity, etc..) an a set of 10 lidar rangefinder measurements. On this environment, the robotic walker gets a dense positive reward for traveling forward on the terrain, a small negative reward for using its motors, and a negative reward of -100 if it falls down. On environments that are too challenging for the agent at its current capabilities, this reward structure can promote a locally optimal behavior of simply staying still and preventing itself from falling.

To learn our policies, we use the TD3-Fork algorithm from Honghao et al. [39], which was shown to train much faster than standard PPO on BPW. We train agents for 3.5 million timesteps. During training we evaluate against four specific challenges (Stairs, PitGap, Stump, and Roughness in Fig. 1) as well as a combined environment (Evaluation) that contains all challenges together. We compare against a policy trained on only the combined environment.

The results of the evaluation on the combined environment can be seen in Fig. 6. Since we do not train for a large number of environment timesteps, we can observe that without a curriculum, TD3-Fork does not manage to learn to the point of a positive reward on any of the test environments.

There is a significant reward divergence in the results starting at 1 million timesteps. The uniform and anti curriculum perform better than having no curriculum with the uniform curriculum perform marginally better than the anti-curriculum. It can be seen on each graph that the policies trained using the SEBN-guided curriculum manages to get to a point where the agent starts receiving a positive reward for each environment, getting past the initial hurdle of the locally optimal staying still behavior.

It is interesting to observe that the learning curve for the SEBN-guided curriculum has much higher variance than the learning curves for other methods. Due to the size of the environment design space, it is necessary to use approximate inference techniques to search for candidate environments. Because we use a sample-search procedure, the search process is not guaranteed to find environments with the largest heuristic difference (see Eq. 3). This means that sub-optimal environments can be introduced into the curriculum. This introduces a large factor into learning curve variance beyond standard noise from reinforcement learning algorithms.

Robosuite - Open Door. In our final domain, we evaluate the SEBN-guided curriculum in a simulated robotics domain using the robosuite simulation environment. In particular we choose the Door task in which an agent needs to manipulate a robot to open a door. For our environment design space, we include two main parameters: the weight of the door (with 6 different settings) and whether the door has a latch or not. The specific robot that we choose to simulate is a Kinova3 arm. We define the following SEBN:

- E: includes design parameters mass $\{0-6\}$ and latch $\{0, 1\}$.
- Ψ: includes "move arm" $\{0, 1\}$, "unlock" $\{0, 1\}$, "door" $\{0, 1, 2\}$.
- K: includes one observable metric $\{0, 1\}$: whether the agent has successfully opened the door.

To learn our policies, we use PPO on a neural network with two hidden layers. We use the inbuilt observation setup and reward shaping in the robosuite environment to accelerate the learning process and we train our agents for 2 million timesteps. We evaluate our learned policies every 100,000 timesteps and the results of the experiment can be seen in Fig. 7.

It is interesting to observe that the default training method stagnates after reaching a reward plateau. When viewing the actual behavior of the learned policy, this reward plateau is indicative of learning a policy of moving the arm towards the door handle but not actually moving the door. It is possible that either the weight of the door or the presence of a latch in harder environments prevents the agent from attempting the difficult action of applying force to

the door to get an increased reward. Since the SEBN-guided curriculum will have started with at least some of its distribution in the easy case of a light door with no latch, the agent will have learned that opening the door can give a positive reward and transfer this behavior to more difficult environments.

We also performed a generalization study, where we test our learned policies on an *unseen* heavy door env not included during our learning process. We observed that with the policy learned by the SEBN-guided curriculum is easily transferred to more heavy doors (obtaining a reward of 240 on the heavy door env compared to 150 without a curriculum), in contrast to the other methods .

5 RELATED WORK

Automated Curriculum Generation. Several topics relate to ordering tasks to improve learning performance. A few approaches have considered the problem of estimating agent skill competencies. In the context of education, in addition to ECD [23], another approach close to ours is that of Green et al. [13], who used a BN to determine the next task for the human student. This approach is similar to [20], which considered an active learning problem in a robotics domain of choosing which skills to practice to maximize future task success, which involves estimating the competence of each skill and situating it in the task distribution through competence-aware planning. In contrast to our approach, they employ a simplified Bayesian time series model that does not relate environmental features with goal and skill competencies. This limits the applicability of their approach towards only choosing what skill to train and not the agent's environment. Similar to our selection process, Ballera et al. [4] used a roulette wheel selection of tasks.

A related area is the literature on Unsupervised Environment Design (UED) [10] and other developed mechanisms for curating environments based on a regret heuristic [17]. In prior UED approaches such as PAIRED [10], the agent's curriculum is generated using a regret-based heuristic. The heuristic is typically an estimate of the true regret, since the optimal policy is unknown. In PAIRED, this heuristic is calculated by learning an antagonistic policy and evaluating the difference between it and the protagonist policy. In contrast, like ACCEL [27], our method does not need to learn a second antagonistic policy and instead uses rollouts from a single agent to compute the next part of the curriculum. In contrast to ACCEL, our curriculum does not rely on local changes and can incorporate larger jumps in environment selection. Furthermore, while it is necessary to use rollouts on each environment for ACCEL to obtain a regret estimate, we can estimate success rates on unseen environments by leveraging the relationships encoded between competencies and environmental features in our SEBN.

Task Descriptors. As mentioned in §2.4, grouping tasks using features, i.e., task descriptors, are a well understood technique for task creation ([16, 18, 26, 31]). The key idea in these works is to facilitate learning transfer by creating similar tasks that share common features. These features can leave certain variables free during task construction that enable a family of similar tasks. Our work supplements prior work by adding the target of the task to the task descriptor, allowing the curriculum to emphasize subtasks.

To our knowledge, there has been limited previous work in integrating curriculum learning on both skills and environment features. However, it can be said that our research expands on the concept of using task descriptors in the creation of automated curricula. In [29], a task-graph curricula is used to generate a curriculum over tasks and environmental features. However, they employ a simple greedy best-first search on the task-graph to choose an order for their curriculum. This is different from our approach that updates a distribution over the task-graph and dynamically adjusts this distribution based on data from rollouts.

Hierarchical Goal Networks. The structure of our Skill Environment Bayesian network shares similarity to both goal skill networks and fault diagnosis networks. In fault diagnosis networks [6], BNs are used to model the relationship between a set of sensors and a set of faults. In our case, the sensors are analogous to an SEBN's observable goal metrics, and the faults are analogous to an SEBN's skills. An SEBN can then be seen as a fault diagnosis network where different roll-outs are independent tests that determine what latent competencies may have not been mastered.

Expert guidance for RL training. One of the limitations of the SEBN is its reliance on the expert-provided competencies. As noted, these could be derived from hierarchical approaches. But providing domain knowledge is common in many hierarchical RL settings. Similar to our work, Patra et al. [28], provided a hierarchical learning structure. This kind of expert knowledge is common in imitation learning (e.g., [40] [15] [21]), where an expert human guides a learning agent. Providing expert guidance is also common in Hierarchical RL approaches (e.g., [2]) and in standard RL approaches (e.g., [3]). Finally, expert guidance was shown to be helpful for a sparse-reward task in an Object Oriented MDP setting [1].

6 CONCLUSION AND FUTURE WORK

We presented a method for automated curriculum generation over goal, skill, and environmental features using a Skill-Environment Bayesian Network (SEBN). The SEBN was used to estimate agent competency level using *past rollouts* and was used to infer estimated agent success rates on unseen environments. We demonstrated the effectiveness of SEBN-based curricula for three domains.

For this work, we relied on a pre-defined set of skills and environmental features. Future work should apply the SEBN to more open ended environments where new environmental features or agent skills are dynamically added to the SEBN. Also, it might be interesting to see if we can apply techniques such as GO-MTL [19] for learning a latent space over tasks and approaches for detecting critical regions [24] to learn new skills.

There has also been recent interest in the use of Large Language Models (LLMs) in planning domains for the purpose of automatically generating planning models. As SEBNs can be built from a heirarchical goal network, it might might be possible to use LLMs to automatically generate SEBNs from domain documents. Given the support of latent competency nodes, it may be easier to generate these SEBNs using LLMs than the equivalent goal networks.

ACKNOWLEDGMENTS

We thank the Basic Research Office of OUSD and NRL for funding this research.

REFERENCES

[1] David Abel, David Hershkowitz, Gabriel Barth-Maron, Stephen Brawner, Kevin O'Farrell, James MacGlashan, and Stefanie Tellex. 2015. Goal-based action priors. In *Proc. of ICAPS*. 306–314.

[2] Mazda Ahmadi, Matthew E. Taylor, and Peter Stone. 2007. IFSA: incremental feature-set augmentation for reinforcement learning tasks. In *Proc. of AAMAS*. 1–8.

[3] Jacob Andreas, Dan Klein, and Sergey Levine. 2017. Modular multitask reinforcement learning with policy sketches. In *Proc. of ICML*. 166–175.

[4] Melvin Ballera, Ismail Ateya Lukandu, and Abdalla Radwan. 2014. Personalizing E-learning curriculum using: reversed roulette wheel selection algorithm. In *Proc. of ICETC*. 91–97.

[5] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proc. of ICML*. 41–48.

[6] Baoping Cai, Lei Huang, and Min Xie. 2017. Bayesian networks in fault diagnosis. *IEEE Transactions on industrial informatics* 13, 5 (2017), 2227–2240.

[7] Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and J Terry. 2023. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. In *Proc. of NeurIPS*. 73383–73394.

[8] Adnan Darwiche. 2009. *Modeling and reasoning with Bayesian networks*. Cambridge university press.

[9] Rina Dechter. 2013. Reasoning with probabilistic and deterministic graphical models: Exact algorithms. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 7, 3 (2013), 1–191.

[10] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. 2020. Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design. In *Proc. of NeurIPS*. 13049–13061.

[11] Malik Ghallab, Dana Nau, and Paolo Traverso. 2016. *Automated Planning and Acting*. Cambridge University Press.

[12] Malik Ghallab, Dana Nau, and Paolo Traverso. to appear. *Acting, Planning and Learning*. Cambridge University Press.

[13] Derek Green, Thomas Walsh, Paul Cohen, and Yu-Han Chang. 2011. Learning a Skill-Teaching Curriculum with Dynamic Bayes Nets. *Proc. of AAAI*, 1648–1654.

[14] Vincent Hsiao, Dana S Nau, Bobak Pezeshki, and Rina Dechter. 2024. Surrogate Bayesian Networks for Approximating Evolutionary Games. In *Proc. of AISTATS*. 2566–2574.

[15] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation learning: A survey of learning methods. *Comput. Surveys* 50, 2 (2017), 1–35.

[16] David Isele, Mohammad Rostami, and Eric Eaton. 2016. Using task features for zero-shot knowledge transfer in lifelong learning. *Proc. of IJCAI* (2016).

[17] Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. 2021. Replay-Guided Adversarial Environment Design. In *Proc. of NeurIPS*. 1884–1897.

[18] George Konidaris, Ilya Scheidwasser, and Andrew G. Barto. 2012. Transfer in reinforcement learning via shared features. *JMLR* 13 (May 2012), 1333–1371.

[19] Abhishek Kumar and Hal Daumé. 2012. Learning task grouping and overlap in multi-task learning. In *Proc. of ICML*. 1723–1730.

[20] Nishanth Kumar, Tom Silver, Willie McClinton, Linfeng Zhao, Stephen Proulx, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Jennifer Barry. 2024. Practice Makes Perfect: Planning to Learn Skill Parameter Policies. In *Proc. of RSS*.

[21] Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. 2018. Hierarchical imitation and reinforcement learning. In *Proc. of ICML*. 2917–2926.

[22] Qiang Liu and Alexander Ihler. 2011. Bounding the partition function using hölder's inequality. In *Proc. of ICML*. 849–856.

[23] Robert J Mislevy, Russell G Almond, and Janice F Lukas. 2003. A brief introduction to evidence-centered design. *ETS Research Report Series* 2003, 1 (2003), i–29.

[24] Daniel Molina, Kislay Kumar, and Siddharth Srivastava. 2020. Learn and link: Learning critical regions for efficient planning. In *Proc. of ICRA*. 10605–10611.

[25] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. 2020. Curriculum learning for reinforcement learning domains: A framework and survey. *JMLR* 21, 181 (2020), 1–50.

[26] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. 2016. Source task creation for curriculum learning. In *Proc of AAMAS*. 566–574.

[27] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. 2022. Evolving Curricula with Regret-Based Environment Design. In *Proc. of ICML*. 17473–17498.

[28] Sunandita Patra, Mark Cavolowsky, Onur Kulaksizoglu, Ruoxi Li, Laura Hiatt, Mark Roberts, and Dana Nau. 2022. A hierarchical goal-biased curriculum for training reinforcement learning. In *Proc. of FLAIRS*, Vol. 35.

[29] Sunandita Patra, Paul Rademacher, Kristen Jacobson, Kyle Hassold, Onur Kulaksizoglu, Laura Hiatt, Mark Roberts, and Dana Nau. 2023. Relating Goal and Environmental Complexity for Improved Task Transfer: Initial Results. In *Working Notes of the Workshop on Generalization in Planning at NeurIPS*.

[30] J. Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

[31] Mohammad Rostami, David Isele, and Eric Eaton. 2020. Using Task Descriptions in Lifelong Machine Learning for Improved Performance and Zero-Shot Transfer. *JAIR* 67 (2020), 673–703.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[33] Andrew Stout and Andrew G Barto. 2010. Competence progress intrinsic motivation. In *Proc. of Int'l Conf. on Development and Learning*. 257–262.

[34] Richard S Sutton and Andrew G. Barto. 2018. *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

[35] Richard S. Sutton, Marlos C. Machado, G. Zacharias Holland, David Szepesvari, Finbarr Timbers, Brian Tanner, and Adam White. 2023. Reward-respecting subtasks for model-based reinforcement learning. *AIJ* 324 (2023), 104001.

[36] Richard S. Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *AIJ* 112, 1 (1999), 181–211.

[37] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. 2017. Domain randomization for transferring deep neural networks from simulation to the real world. In *Proc. of IROS*. 23–30.

[38] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. arXiv:2407.17032

[39] Honghao Wei and Lei Ying. 2021. FORK: A Forward-Looking Actor For Model-Free Reinforcement Learning. In *Proc. of Conference on Decision and Control*. 1554–1559.

[40] Ruohan Zhang, Faraz Torabi, Lin Guan, Dana H. Ballard, and Peter Stone. 2019. Leveraging Human Guidance for Deep Reinforcement Learning Tasks. In *Proc. of IJCAI*. 6339–6346.