Logical Specifications-guided Dynamic Task Sampling for Reinforcement Learning Agents

Yash Shukla¹, Tanushree Burman¹, Abhishek N. Kulkarni², Robert Wright³, Alvaro Velasquez⁴, Jivko Sinapov¹
¹Tufts University
²University of Texas at Austin
³Georgia Tech Research Institute
⁴University of Colorado Boulder

Abstract

Reinforcement Learning (RL) has made significant strides in enabling artificial agents to learn diverse behaviors. However, learning an effective policy often requires a large number of environment interactions. To mitigate sample complexity issues, recent approaches have used high-level task specifications, such as Linear Temporal Logic (LTL₁) formulas or Reward Machines (RM), to guide the learning progress of the agent. In this work, we propose a novel approach, calledLogical Specifications-guided Dynamic Task Sampling (LSTS), that learns a set of RL policies to guide an agent from an initial state to a goal state based on a high-level task specification, while minimizing the number of environmental interactions. Unlike previous work, LSTS does not assume information about the environment dynamics or the Reward Machine, and dynamically samples promising tasks that lead to successful goal policies. We evaluate LSTS on a grid-world and show that it achieves improved time-to-threshold performance on complex sequential decision-making problems compared to state-of-the-art RM and Automaton-guided RL baselines, such as Q-Learning for Reward Machines and Compositional RL from logical Specifications (DIRL).

Moreover, we demonstrate that our method outperforms RM and Automaton-guided RL baselines in terms of sample-efficiency, both in a partially observable robotic task and in a continuous control robotic manipulation task.

1 Introduction

Agents are now capable of learning optimal control behavior for a broad spectrum of tasks, ranging from Atari games to robotic manipulation tasks, thanks to recent advancements in Reinforcement Learning (RL). Despite the progress made in RL, learning an optimal task policy using model-free RL techniques still suffers from sample complexity issues because of sparse reward settings and unknown transition dynamics (Lattimore, Hutter, and Sunehag 2013). These challenges further intensify in long-horizon settings, where the agent needs to perform a series of correct sequential decisions to achieve the goal. Certain tasks (such as - robot needs to make dinner only if it bought groceries in the afternoon) require the agent to encode and remember its episodic history (whether the groceries were bought) in order to solve the task effectively. To alleviate this issue in complicated tasks, several lines of work have explored representing the goal using an intricately shaped reward function that guides the agent toward the goal (Grzes 2017). However, generating such a reward function requires the human engineer to assign ‘importance’ weights to certain aspects of the task, which is time consuming and assumes knowledge on which aspects of the task are important. Poorly engineered reward functions can lead to local optima, where the agent learns to satisfy only a subset of goals and ignores the rest.

Recent research has investigated representing the goal using high-level specification languages, such as finite-trace Linear Temporal Logic (LTL₁) (De Giacomo and Vardi 2013), Reward Machines (RM) (Icarte et al. 2022), SPECTRL (Jothimurugan, Alur, and Bastani 2019) that allow defining the goal of the task using a graphical representation of sub-tasks. The high-level objective is known before commencing the task, and the graphical representation allows the agent to learn policies that achieve easier sub-goals initially, and build upon them to achieve complex goals. Encoding the task using a graphical structure allows us to tackle the problem in a Markovian manner by tracking the history as a part of the state space (Afzal et al. 2023), thereby allowing the agent to keep track of its episodic history. For instance, if the task for a robot is to reach kitchen and then make dinner, the graphical structure of the task obtained from the high-level specification allows the agent to reason whether it has reached the kitchen before it can commence its policy for making dinner. RM approaches still require human guidance in defining the reward structure, which is dependent on knowing how much reward should be assigned for accomplishing each sub-goal. The process of designing the reward structure assumes that the human engineer is aware of how much should reward should the agent receive when it accomplishes the sub-goals in particular order. This assumption is infeasible in scenarios when the structure of the environment or the exact order in which the sub-goals must be achieved is unknown in advance. In contrast, our method does not require access to the reward structure.

Another method, Compositional RL from Logical Specifications (DiRL) (Jothimurugan et al. 2021) mitigates the reward assignment issue by using Dijkstra’s algorithm to determine which sub-tasks (edges) should be explored in the SPECTRL DAG graph (Jothimurugan, Alur, and Bastani 2019) in order to learn policies to reach nodes in the DAG
that yield the highest success rate. DiRL requires the agent to learn RL policies for satisfying all outgoing edge propositions (each edge encodes a sub-task) from such nodes. However, this approach requires the agent to explore a sub-task for a manually specified number of interactions, which requires knowledge about the task complexity. DiRL can be sample-inefficient as it attempts to learn unproductive policies on unpromising some sub-tasks, as the agent has to spend the defined number of interactions learning a policy for the sub-task. Unlike DiRL, our approach is sample-efficient as it finds unpromising sub-tasks based on the learning progress of the sub-tasks, and discards them; saving costly interactions and converging to a successful policy faster. This problem of minimizing the overall number of interactions while learning a set of successful policies is non-trivial as the problem equates to finding the shortest path in a graph where the true edge weights are unknown a priori (Szepesvári 2004). In our case, the edge weight denotes the total number of environmental interactions required by the agent to learn a successful policy for the sub-task encoded by the edge, in which the agent must induce a visit to a state where certain properties hold true. And, we can sample interactions for a sub-task only if we have a policy to reach the edge’s source node from the start node of the graph, making the learning process more sample-inefficient.

To address the above challenges, we present Logical Specifications-guided Dynamic Task Sampling (LSTS). We begin with a high-level objective represented using SPECTRL specification formulas which can equivalently be represented using directed acyclic graphs (DAG) (Jothimurugan, Alur, and Bastani 2019). The DAG structure encodes memory, helping the agent understand what events of interest have occurred in the past, and which events must occur to reach the accepting states. Our key insight is to learn RL policies for sub-tasks defined using cost-to-goal formulation of the edges of the DAG. Specifically, the agent transitions from the node $q$ to $p$ in the DAG when the propositional logic formula labeling the edge $(q, p)$ evaluates to true. We use the set of propositional logic formulas labeling the outgoing edges from a given node in DAG to define sub-tasks. The trajectory induced by a successful RL policy for the sub-task enables the agent’s high-level state in the DAG to transition from the source node to the destination node of the edge defining the sub-task. We utilize an adaptive Teacher-Student learning approach, where, (1) the Teacher agent uses its high-level policy to actively sample a sub-task for the Student agent to explore. The high-level policy considers the DAG representation and the Student agent’s expected performance on all the sub-tasks. The goal of the Teacher agent is to satisfy the high-level objective in the fewest number of environmental interactions, and (2) The Student agent explores the environment for few interactions (much fewer than the interactions required to learn a successful policy for the sub-task) while simultaneously updating its low-level RL policy for the sampled sub-task. Next, the Teacher observes the Student’s performance and updates its high-level policy. Steps (1) and (2) continue alternately until the Student agent learns a set of successful policies to satisfy the high-level objective.

Running example: As an example, let us look at the environment shown in Fig. 1a. The goal for the agent is to collect any of the two Keys, followed by opening the Door and then reaching the Goal while avoiding the Lava at all times. The task’s high-level objective $(\phi)$ is represented using the SPECTRL formula and its corresponding DAG representation $G_{\phi}$ in Fig. 1b. The DAG does not contain information about the environment configuration, such as: the optimal number of interactions required to reach Door from $Key_1$ are much higher compared to the interactions required to reach Door from $Key_2$, making the $Key_1$ to Door trajectory sub-optimal. A naive approach to tackle this problem would involve learning individual RL policies for all the transitions of DAG, and then later identifying which path of the DAG leads to goal satisfaction in the fewest interactions. However, this approach will still explore all the paths of the DAG and prove sample-inefficient. It is crucial to prevent any additional interactions the agent spends in learning a policy for the sub-task defined by the edge $q_1 \xrightarrow{\neg L \land d} q_3$ as the path $q_0 \xrightarrow{\neg L \land k_1} q_1 \xrightarrow{\neg L \land d} q_3 \xrightarrow{\neg L \land q} q_4$ will always be sub-

![Figure 1: (a) Task environment. The agent (denoted using red triangle) has to pick-up one of the keys and toggle the door open to reach the goal state; (b) The SPECTRL formula for the task and its DAG. Formulas $l$, $k_1$, $k_2$, $d$ and $q$ correspond to Lava, $Key_1$, $Key_2$, Door and Goal respectively; (c) Learning curves for individual sub-tasks (averaged over 10 trials) generated using LSTS. The path chosen by LSTS is highlighted in red in Fig.1(b)](image)
optimal. In our proposed approach LSTS, the Student agent begins with the aim of learning two distinct RL policies: $\pi_1$ for the task of visiting $K\epsilon y_1$ and $\pi_2$ for the task of visiting $K\epsilon y_2$, both avoiding Lava. The Teacher agent initially samples evenly from these two sub-tasks for the Student but later biases its sampling toward the sub-task on which the Student agent shows higher learning potential. Once the Student agent learns a successful policy for one of the sub-tasks (let’s say the learned policy $\pi_1^*$ corresponding to the sub-task defined by the transition $q_0 \xrightarrow{e\land k_2} q_1$), the Teacher does not sample that task anymore, identifies the next task(s) for the Student using the DAG representation, and appends them to the set of tasks it is currently sampling (in this case, the only next task is: $q_1 \xrightarrow{\neg l\land d} q_3$). Since the Student agent only has access to the state distribution over $q_0$, it follows the trajectory given by $\pi_1^*$ to a state that lies in the set of states where the proposition $\neg Lava\land K\epsilon y_1$ holds true before commencing its learning for the policy (for $q_1 \xrightarrow{\neg l\land d} q_3$). If the Student agent learns the policies $\pi_2^*$ for satisfying the sub-task defined by $q_0 \xrightarrow{e\land k_2} q_2$ and $\pi_4^*$ for $q_2 \xrightarrow{\neg l\land d} q_3$ before learning $\pi_3^*$, it effectively has a set of policies to reach the node $q_3$. Thus, the Teacher will now only sample the next task for the Student in the DAG representation $q_3 \xrightarrow{\neg l\land g} q_4$, as learning RL policies for paths that reach $q_3$ are effectively redundant. This process continues iteratively until the Student agent learns a set of policies that reach the goal node ($q_4$) from the start node ($q_0$). The learning curves in Fig. 1c empirically validate the running example. As evident from the learning curves, the Student agent learns policies for the path $q_0 \xrightarrow{e\land k_2} q_2 \xrightarrow{\neg l\land d} q_3 \xrightarrow{\neg l\land g} q_4$ that produce trajectories to reach the goal node $q_4$ from the initial node $q_0$, without excessively wasting interactions on the unpromising sub-task $q_1 \xrightarrow{\neg l\land d} q_3$. The dashed lines in Fig. 1c signify the interactions at which a task policy converged.

The dynamic task sampling strategy promotes LSTS to achieve sample-efficient learning on complex tasks by identifying unpromising tasks and discarding them, saving costly interactions. Our empirical results show that LSTS reduces environmental interactions by orders of magnitude compared to Specifications-Guided RL Baseline DiRL, Reward Machine-based baselines QRM (Icarte et al. 2018), GSRS (Camacho et al. 2018), and curriculum learning baseline TSCL (Matiisen et al. 2020). We also evaluate LSTS$^4$, a modified algorithm that further improves sample efficiency by continuing exploration on a new sub-task once the sub-task goal state is reached. We perform evaluation on two simulated robotic tasks and demonstrate that LSTS reduces the number of interactions by orders-of-magnitude when compared to automaton-guided RL baselines.

2 Related Work

Automaton-guided RL approaches utilize temporal logic-based language specifications to define tasks (Toro Icarte et al. 2018; Alur et al. 2022; Xu and Topcu 2019; Neary et al. 2021). Separating policies for task sub-goals aids in abstracting knowledge that can be utilized in novel tasks (Icarte et al. 2018), without reliance on a dense reward function. Another technique is to shape the reward in proportion to the distance from the accepting node in the automaton (Camacho et al. 2018); however, this often leads to suboptimal reward settings. Augmenting the reward function with Monte Carlo Tree Search helps mitigate this issue (Velasquez et al. 2021). This approach requires the ability to plan-ahead in the environment, which is not always feasible. Automaton-guided RL has been used to aid navigational exploration for robotic domains (Cai et al. 2023) and for multi-agent settings (Hammond et al. 2021). Generating a curriculum given the high-level objective (Shukla et al. 2023) requires access to the Object-Oriented MDP (Diuk, Cohen, and Littman 2008), which cannot be obtained if environment details are not known in advance. DiRL interleaves high-level planning with RL to learn a policy for each edge, which overcomes challenges arising from poor representations (Jothismurugan et al. 2021). This approach becomes inefficient in terms of number of interactions, as it requires the agent to act for a predetermined number of interactions, even if learning the task does not show any promise. Unlike previous works, in this paper, we propose an logical specifications-guided dynamic task sampling approach that does not require access to the environment dynamics or the Reward Machine, and efficiently samples tasks that show promise toward the high-level objective, saving interactions on unpromising tasks.

Teacher-Student algorithms (Matiisen et al. 2020) have been previously studied in Curriculum Learning literature (Narvekar et al. 2020; Shukla et al. 2022, 2024) and in the Intrinsic Motivation literature (Oudeyer and Kaplan 2009). These approaches utilizes a bi-level structure, where a Teacher agent suggests tasks to the Student based on the Student’s demonstrated potential. This approach facilitates the learning of simpler tasks initially, enabling the transfer of knowledge to more complex tasks and reducing the number of interactions required to attain a successful policy. However, these methods typically focus on optimizing a curriculum for learning a single policy, which may not be suitable for tasks with extended temporal aspects. In contrast, we propose a Logical Specifications-driven Teacher-Student learning approach that targets the learning of policies for promising automaton transitions, thereby enhancing sample efficiency compared to conventional approaches.

3 Theoretical Framework

Episodic MDP. An episodic labeled Markov Decision Process (MDP) $\hat{M}$ is a tuple $(S, A, P, R, S_0, \gamma, K, P, L)$, where $S$ is the set of states, $A$ is the set of actions, $P(s'|s,a)$ denotes the transition probability of reaching state $s' \in S$ from $s \in S$ using action $a \in A$, $R : S \times A \times S \to \mathbb{R}$ is the reward function, $S_0$ is the initial state distribution, $\gamma \in [0, 1]$ is the discount factor, $K$ is the maximum number of interactions in any episode, $P$ is a set of predicates, and $L : S \to 2^P$ is a labeling function that maps a state $s \in S$ to a subset of predicates that are true in that state. During each interaction, the agent observes the current state $s$ and selects
an action \(a\) according to its policy function \(\pi(a|s, \theta)\) with parameters \(\theta\). The agent transitions to a new state \(s'\in S\) with probability \(P(s'|s, a)\). The agent’s goal is to learn an optimal policy \(\pi^*\) that maximizes the discounted return 
\[
G_0 = \sum_{k=0}^{K} \gamma^k R(s'_k, a_k, s_k)
\]
until the end of the episode, which occurs after at most \(K\) interactions.

**High level specification language:** In our framework, we adopt the specification language SPECTRL to articulate reinforcement learning tasks (Jothimurugan, Alur, and Bastani 2019). A specification \(\phi\) in SPECTRL is a logical formula applied to trajectories, determining whether a given trajectory \(\zeta = (s_0, s_1, \ldots)\) successfully accomplishes a desired task. Mathematically, \(\phi\) can be depicted as a function \(\phi: Z \rightarrow \mathbb{B}\), where \(\mathbb{B} = \{\text{TRUE, FALSE}\}\) and \(Z\) is the set of all trajectories.

Formally, a specification is defined over a set of atomic predicates \(\mathcal{P}_0\). Each \(p \in \mathcal{P}_0\) is associated with a function \(f_p : S \rightarrow \mathbb{B}\). The agent’s MDP state \(s\) satisfies \(p\) (denoted by \(s \models p\)) when \(f_p(s) = \text{TRUE}\) (in other words, \(p \subseteq L(s)\)).

The set of predicates \(\mathcal{P}\) comprises conjunctions and disjunctions of atomic predicates \(\mathcal{P}_0\). A predicate \(b \in \mathcal{P}\) follows the grammar \(b ::= p \mid (b_1 \land b_2) \mid (b_1 \lor b_2)\), where \(p \in \mathcal{P}_0\). Each predicate \(b \in \mathcal{P}\) corresponds to a function \(f_b : S \rightarrow \mathbb{B}\) defined naturally over Boolean logic.

The syntax of SPECTRL specifications is given by
\[
\phi ::= \text{achieve } b \mid \phi_1 \text{ ensuring } b \mid \phi_1 ; \phi_2 \mid \phi_1 \lor \phi_2,
\]
where \(b \in \mathcal{P}\). Here, achieve and ensuring correspond to ‘eventually’ and ‘always’ operators in temporal logic.

Each specification \(\phi\) corresponds to a function \(f_\phi : Z \rightarrow \mathbb{B}\), and \(\zeta \in Z\) satisfies \(\phi\) (denoted \(\zeta \models \phi\)) if \(f_\phi(\zeta) := \text{TRUE}\). The SPECTRL semantics for a finite trajectory \(\zeta\) of length \(t\) are:
\[
\begin{align*}
\zeta \models \text{achieve } b & \quad \text{if } \exists i \leq t, s_i \models b \quad \text{(1)} \\
\zeta \models \phi \text{ ensuring } b & \quad \text{if } \exists i \leq t, s_i \models b \quad \text{(2)} \\
\zeta \models \phi_1 ; \phi_2 & \quad \text{if } \exists i \leq t, \zeta_{0:i} \models \phi_1 \text{ and } \zeta_{i+1:t} \models \phi_2 \quad \text{(3)} \\
\zeta \models \phi_1 \lor \phi_2 & \quad \text{if } \zeta \models \phi_1 \text{ or } \zeta \models \phi_2 \quad \text{(4)}
\end{align*}
\]

Intuitively, the condition (1) signifies that the trajectory should eventually reach a state where the predicates \(b\) hold true. The condition (2) signifies that the trajectory should satisfy specification \(\phi\) while always remaining in states where \(b\) holds true. The condition (3) signifies that the trajectory should sequentially satisfy \(\phi_1\) and then \(\phi_2\). The condition (4) signifies that the trajectory should satisfy either \(\phi_1\) or \(\phi_2\). A trajectory \(\zeta\) satisfies \(\phi\) if there is a \(t\) such that the prefix \(\zeta_{0:t}\) satisfies \(\phi\).

Furthermore, each SPECTRL specification \(\phi\) is guaranteed to have an equivalent directed acyclic graph (DAG), called an abstract graph. An abstract graph \(\mathcal{G} = (Q, E, q_0, F, \beta, Z_{\text{safe}}, \kappa)\) is a directed acyclic graph (DAG) with nodes \(Q\), (directed) edges \(E \subseteq Q \times Q\), initial node \(q_0 \in Q\), final nodes \(F \subseteq Q\), subgoal region map \(\beta : Q \rightarrow 2^S\), such that for each \(q \in Q\), \(\beta(q)\) is a subgoal region and safe trajectories \(Z_{\text{safe}} = \bigcup_{e \in E} Z_{\text{safe}}\) where \(Z_{\text{safe}} \subseteq Z_f\) denotes the safe trajectories for edge \(e \in E\). Intuitively, \((Q, E)\) is a standard DAG, and \(q_0\) and \(F\) define a graph reachability problem for \((Q, E)\). Furthermore, \(\beta\) and \(Z_{\text{safe}}\) connect \((Q, E)\) back to the original MDP \(M\); in particular, for an edge \(e = q \rightarrow q'\), \(Z_{\text{safe}}\) is the set of trajectories in the MDP \(M\) that can be used to transition from \(\beta(q)\) to \(\beta(q')\).

The function \(\kappa\) labels each edge \(e = q \rightarrow q'\) with the predicates \(b_e\) (labeled edge denoted as \(e := q \xrightarrow{b_e} q'\)). The agent transitions from \(q\) to \(q'\) when the states \(s_1, s_{i+1} \in \beta(q)\) and \(b_e \subseteq L(s_{i+1})\) and \(j \geq 0\).

Given a SPECTRL specification \(\phi\), we can construct an abstract graph \(\mathcal{G}_\phi\) such that, for every trajectory \(\zeta \in Z\), we have \(\zeta \models \phi\) if and only if \(\zeta \models \mathcal{G}_\phi\). Thus, we can solve the reinforcement learning problem for \(\phi\) by solving the reachability problem for \(\mathcal{G}_\phi\). As described below, we leverage the structure of \(\mathcal{G}_\phi\) in conjunction with reinforcement learning to do so. In summary, SPECTRL specifications provide a powerful and expressive means to define and evaluate reinforcement learning tasks. It allows users to specify complex conditions and requirements for successful task completion, enabling a nuanced approach to learning from specifications.

**Problem Formulation.** Given an MDP \(M\) with unknown transition dynamics and a SPECTRL formula \(\phi\) representing the high-level task objective, let \(\mathcal{G}_\phi\) be the DAG representing the language of \(\phi\). Let Paths\((q, X)\) be the set of all paths in the DAG originating in \(q\) and terminating at a node in \(X \subseteq Q\). The aim of this work is to learn a set of policies \(\pi_i, i = 0, \ldots, n - 1\), with the following three properties: (i) Following \(\pi_0\) results in a trajectory in the MDP that induces a transition from \(q_0\) to some state \(q_1 \in Q\) in the DAG, following \(\pi_1\) results in a trajectory in MDP that induces a transition from \(q_1\) to some state \(q_2 \in Q\) in the DAG, and so on. (ii) The resulting path \(q_0q_1\ldots q_n\) in the DAG terminates at a final node, i.e., \(q_n \in F\), with probability greater than a given threshold, \(\eta \in (0, 1)\). (iii) The total number of environmental interactions spent in exploring and learning sub-task policies are minimized.

**4 Methodology**

**Sub-task definition:** Given the DAG \(\mathcal{G}_\phi\) representing the language of \(\phi\), we define a set of sub-tasks based on the edges of the DAG. Intuitively, given any MDP state \(s \in S\) and a DAG node \(q \in Q\), a sub-task defined by an edge from node \(q\) to \(p\) in \(Q\) defines a reach-avoid objective for the agent represented by the SPECTRL formula,

\[
\text{Task}(q, p) ::= \text{achieve}(b(q, p)) \text{ ensuring } \left( \bigwedge_{r \in \text{SC}(q), r \neq p} \neg b(q, r) \right)
\]

where \(b(q, p)\) is the propositional formula labeling the edge from \(q\) to \(p\) in the DAG and \(\text{SC}(q)\) is the set of successors of node \(q\) in DAG. For example, in Fig. 1b, the propositional formula labeling the edge from \(q_0\) to \(q_1\) is \(b_{(q_0, q_1)} = \neg l \land k_1\). When \(e = (q_0, q_1)\), we use Task\((e)\) instead of Task\((q, p)\) and \(b_e\) instead of \(b_{(q, p)}\) for notational convenience.

\footnote{See DiRL (Jothimurugan et al. 2021) for more details}
Each sub-task $\text{Task}(q, p)$ defines a problem to learn a policy $\pi^{*}_{(q, p)}$ such that, given any MDP state $s_0 \in S$, following $\pi^{*}_{(q, p)}$ results in a trajectory $s_0s_1 \ldots s_n$ in MDP that induces the path $qq_1 \ldots qp$ in the DAG. That is, the agent’s high-level DAG state remains at $q$ until it transitions to $p$. While constructing the set of sub-tasks, we omit transitions that lead to a ‘sink’ state (from which final states are unreachable).

Given the MDP $M$ with unknown transition dynamics and the SPERTL objective, $\phi$, we first translate $\phi$ to its corresponding directed acyclic graphical (DAG) representation $G_\phi = (Q, E, q_0, F, \beta, Z_{saT}, \kappa)$. Next, we define the set of sub-tasks. For this, we consider the edges that lie on some path in the DAG from $q_0$ to some node in $F$. This is because any path that does not visit $F$ leads to a sink state from which the objective cannot be satisfied. Such edges are identified using breadth-first-search (Moore 1959).

**LSTS Initialization:** The algorithm for LSTS is described in Algo. 1. We begin by initializing the following (lines 2-4): (1) Set of: Active Tasks $\mathcal{AT}$, Learned Tasks $\mathcal{LT}$, Discarded Tasks $\mathcal{DT}$; (2) A Dictionary $\Pi$ : $\text{Task}(e) \rightarrow \pi_e$ that maps the sub-task $\text{Task}(e)$ corresponding to edge $e$ of DAG $G_\phi$, to a RL policy $\pi_e$; (3) A Dictionary $Q$ : $\text{Task}(e) \rightarrow \mathbb{R}$ representing the Q-Values for the Teacher agent by mapping $\text{Task}(e)$ to a numerical q-value associated with $\text{Task}(e)$.

Initially, we transform $G_\phi$ into an Adjacency Matrix denoted as $X$ (line 6), then identify the tasks corresponding to all outgoing edges $E_{q_0} \subseteq E$ originating from the initial node $q_0$ (line 7). Satisfying the predicates $b_{(q_0, q_1)} \in \kappa(E_{q_0})$ signifies a reachability sub-task $M'$. In this sub-task, every goal state $s \in S_f$ of $M'$ satisfies the condition $b_{(q_0, q_1)} \subseteq L(s)$. The Student agent is rewarded positively for fulfilling $b_{(q_0, q_1)}$ and negatively at all other time steps. The state, action space, and transition dynamics of $M'$ mirror those of $M$. To accomplish this sub-task, the Student agent must learn a Reinforcement Learning (RL) policy $\pi^{*}_{(q_0, q_1)}$ ensuring a visit from $q_0$ to $q_1$ with a probability exceeding a predetermined threshold ($\eta$). Additionally, the policy must prevent unintended transitions in the Directed Acyclic Graph (DAG); for example, while obtaining $Key_1$, it should not inadvertently obtain $Key_2$.

**Teacher-Student learning:** To begin, the Teacher Q-Values for all the sub-tasks corresponding to edges in $\mathcal{AT}$ (i.e., tasks corresponding to $E_{q_0}$) are set to zero (line 8). We formalize the Teacher’s objective of selecting the most promising sub-task within a Partially Observable Markov Decision Process framework (Kaelbling, Littman, and Cassandra 1998). Here, the Teacher lacks access to the complete state of the Student agent but only observes its performance on a sub-task (such as success rate or average returns). The Teacher’s action involves choosing a sub-task $\text{Task}(e) \in \mathcal{AT}$ that the Student agent should prioritize for training. Within this POMDP framework, each action taken by the Teacher is associated with a Q-Value. Generally, higher Q-Values indicate tasks where the Student agent demonstrates greater success, prompting the Teacher to prioritize sampling such tasks more frequently to achieve $\phi$ (reaching a goal node) with the fewest overall interactions.

(A) The Teacher agent samples a sub-task $\text{Task}(e) \in \mathcal{AT}$ using the $\epsilon$-greedy exploration strategy (line 10), and (B) The Student agent trains on task $\text{Task}(e)$ using its corresponding RL policy $\Pi[e]$ for $a$ few interactions (line 11). In one Teacher timestep, the Student trains on $x$ environmental interactions. Since the aim of the Teacher is to keep switching the Student agent’s learning progress to a sub-task on which it shows the highest promise, $x < < $ total number of environmental interactions required by the Student agent to learn a successful RL policy for $\text{Task}(e)$.

(C) The Teacher observes the Student agent’s average return $g_t$ on these $x$ interactions, and updates its Q-Value for $\text{Task}(e)$ (line 12) using: $Q[e] \leftarrow \alpha(g_t) + (1 - \alpha)Q[e]$

where $\alpha$ is the Teacher learning rate, $g_t$ is the average Student agent return on $\text{Task}(e)$ at the Teacher timestep $t$. As learning progresses, $g_t$ increases as $t$ increases, and hence we use a constant parameter $\alpha$ to tackle the non-stationarity of a moving return distribution. Several other algorithms could be used for the Teacher strategy (e.g. Thompson Sampling). Steps (A), (B), (C) continue successively until the policy for any $\text{Task}(e) \in \mathcal{AT}$ task converges.

**Sub-task convergence criteria:** The Student agent’s RL policy for $\text{Task}(q, p)$ is considered to be converged (line 13) if a trajectory $\zeta$ produced by the policy produces the transition in the DAG with probability $Pr_{\zeta \in \mathcal{E}}[\text{Task}(q, p)] \geq \eta$ and $\Delta(g_t, g_{t-1}) < \tau$ where $\eta$ is the expected performance and $\tau$ is a small numerical value. In simple terms, a converged policy achieves an average success rate of at least $\eta$ and shows no further improvement by maintaining consistent average returns. Similar to other RM methods, we assume access to the labeling function $L$ to check if a trajectory $\zeta$ satisfies the formula $b_{(q, p)}$ by verifying if the final state $s$ of the trajectory meets the condition $b_{(q, p)} \subseteq L(s)$. It’s worth noting that the sub-goal regions may overlap, meaning the same state $s$ could satisfy propositions for multiple nodes in the Directed Acyclic Graph (DAG). Once a policy for the $\text{Task}(q, p)$ converges, we add $\text{Task}(q, p)$ to the set of Learned Tasks $\mathcal{LT}$ and remove it from the set of Active Tasks $\mathcal{AT}$ (line 14).

**Discarding unpromising sub-tasks:** On learning a successful policy for the $\text{Task}(q, p)$ (transition $q \xrightarrow{b_{(q, p)}} p$), we find the sub-tasks to be discarded (line 15). We find the sub-tasks corresponding to edges that: (1) lie before $p$ in a path from $q_0$ to any $q \in F$, and, (2) do not lie in a path to $q \in F$ that does not contain $p$. Intuitively, if we already have a set of policies that can generate a successful trajectory to reach the node $p$, we do not need to learn policies for sub-tasks that ultimately lead only to $p$ (e.g., in Fig. 1 if we have successful policies for $\text{Task}(q_0, q_2)$ and $\text{Task}(q_2, q_3)$, we can discard $\text{Task}(q_0, q_1)$ and $\text{Task}(q_1, q_3)$). We add all such sub-tasks to the set of Discarded Tasks $\mathcal{DT}$ (line 16). As an extension, in the limit, an optimal policy can be found by not completely discarding such sub-tasks, but rather biasing away from them so that they would still be explored.
Algorithm 1: LSTS (Gϕ, M, η, x)

Output: Set of learned policies: Π∗, Edge-Policy Dictionary: P
1: Placeholder Initialization:
2: Sets of: Active Tasks (AT) ← ∅;
3: Discarded Tasks (DT) ← ∅;
4: Teacher Q-Value Dictionary: Q : Task(e) → −∞
5: Algorithm:
6: X ← Adjacency Matrix (Gϕ)
7: AT ← AT ∪ {Tasks(X[q0])}
8: ∀ Task(e) ∈ AT : Q[e] = 0
9: while True do
10: e ← Sample(Q)
11: Π[e], g ← Learn(M, Gϕ, e, x, P)
12: Update Teacher(Q, e, g)
13: if Convergence(Q, e, g, η) then
14: Π∗ ← Π∗ ∪ Π[e];
15: LT ← LT ∪ {Tasks(X[e])};
16: AT ← AT \ {Tasks(X[e])}
17: Tasks(EAT) ← Discarded Tasks(X, e)
18: DT ← DT ∪ Tasks(EAT)
19: if |Tasks(EAT)| = 0 then
20: break
21: ∀ Task(π) ∈ Tasks(EAT) : Q[π] = 0
22: AT ← AT ∪ Tasks(EAT)
23: end if
24: end while
25: return Π∗, Π

Traversing in the DAG until φ satisfied: Next, we identify the subsequent set of tasks Tasks(EAT) within the DAG to include in the AT set (line 17). This is determined by pinpointing sub-tasks corresponding to all outgoing edges from p. Given that the edge e(q,p) represents the transition q b(e,p) p, we have learned policies that can produce a trajectory leading to a state where b(q,p) holds true. Tasks(EAT) consists of X[p] DT where \ denotes set-minus, which indicates sub-tasks corresponding to outgoing edges from p not included in the DT set.

Once Tasks(EAT) is identified, we assign Teacher Q-values of 0 to all Task(π) ∈ Tasks(EAT) to ensure the Teacher samples these tasks (line 21). In our episodic scenario, each episode initiates from a state s  S0 where the propositions for q0 are true. If the current sampled sub-task is Task(p, r), the agent follows a trajectory utilizing learned policies from Π∗ to reach a state where the propositions for reaching DAG node p are true (i.e., s ∈ β(p)). Subsequently, the agent endeavors to learn a separate policy for Task(p, r).

These aforementioned steps (lines 9-24) continue iteratively until |Tasks(EAT)| = 0. This signifies that there are no further tasks to incorporate into our sampling strategy, and we have arrived at a node q ∈ F. Hence, we exit the while loop (line 19) and return the set of learned policies Π∗ along with the edge-policy dictionary Π (line 25). From Π and Π∗, we obtain an ordered list of policies Π∗ list = π(q1,q2), . . . , π(qn−1,qn) such that sequentially following π ∈ Π∗ list generates trajectories satisfying the SPECTRL objective φ

Guarantee: Given the ordered list of policies Π∗ list, we can generate a trajectory ζ in the task M with Prζ∈ζ[ζ satisfies φ] ≥ η (Details in Appendix B).

5 Experimental Results

We aim to answer the following questions: (Q1) Does LSTS yield sample efficient learning compared to state-of-the-art baselines? (Q2) After reaching a sub-task goal state, can we sample a new sub-task to continue training and improve sample efficiency? (Q3) Does LSTS yield sample efficient learning for complex robotic tasks with partially observable or continuous control settings? (Q4) How does LSTS scale to complex search-and-rescue scenarios?

5.1 LSTS - Gridworld Domain

To answer (Q1), we evaluated LSTS on a Mini-grid (Chevalier-Boisvert, Willems, and Pal 2018) inspired domain with the SPECTRL objective:

φ grid := (achieve(k1) or achieve(k2);
achieve(d) ; achieve(g)) ensuring(¬ l) (5)

where k1, k2, d, g, l represent the atomic propositions Key1, Key2, Door, Goal, Lava respectively. The environment and its φ are given in Fig. 1. The agent needs to pick-up any of the Keys before heading to the Door. After toggling the Door open, the agent needs to reach the Goal. At all times, the agent needs to avoid the Lava. For our experiments, we assume an episodic setting where an episode ends if the agent touches the Lava object, reaches the Goal or exhausts the number of allocated interactions.

The problem is complex as the agent needs to perform a series of high-level correct actions to satisfy φ grid. The action space of the agent consists of three navigational actions:
move forward, rotate left and rotate right. The agent can also perform: pick-up action that adds a Key to the agent’s inventory if the agent is facing the Key, drop places the Key in the next grid if grid is empty and Key is present in the inventory, and, a toggle action for the Door (closed → open) only if the agent is holding the Key. In this experiment, we assume a fully-observable setting where the environmental state is an image-like low-level encoding of the state. For each cell in the grid, the encoding returns an integer that describes the item occupying the grid, along with any additional information (e.g., the Door can be open or closed).

We employ PPO (Schulman et al. 2017) for the Student RL agent, which works for discrete and continuous action spaces. We consider a standard actor-critic architecture with 2 convolutional layers followed by 2 fully connected layers. For LSTS, the reward function is sparse. The agent gets a reward of \((1 - 0.9^{\frac{\text{interactions taken}}{\text{interactions allocated}}})\) (where \(\text{interactions allocated} = 100\)) if it achieves the goal in the sub-task, and a reward of 0 otherwise.

**Baselines:** We compare our LSTS method with six baseline approaches: learning from scratch (LFS), Reward Machine-based (RM) baselines: GSRS (Camacho et al. 2018), QRM (Icarte et al. 2018); and Compositional RL from Logical Specifications (DiRL) (Jothimurugan et al. 2021). All the baselines are implemented using the RL algorithm (PPO), described above. GSRS assigns reward inversely proportional to the distance from the RM goal node. QRM employs a separate Q-function for each node in the RM, and DiRL uses Dijkstra’s algorithm (edge cost is the average RL policy success rate) to guide the agent in choosing a path from the specification graph. For the fifth baseline, we modify DiRL such that instead of manually specifying a limit on the number of interactions, which needs to be fine-tuned to suit the task, we stop learning a sub-task once it has reached the convergence criteria defined in Section 4. We call this modified baseline as DiRLc. The sixth baseline (TSCL (Matiisen et al. 2020)) follows a curriculum learning strategy where the Teacher samples most promising task without the use of any automaton to guide the learning progress of the agent. (More details in Appendix C)

The results in Table 1 (averaged over 10 trials) show that LSTS reaches a successful policy quicker compared to all baselines. LSTSct is a modified version of LSTS and is described below. Our custom baseline, DiRLc is more sample-efficient than DiRL, and both outperform other baselines, which do not learn a meaningful policy. We performed an unpaired t-test (Kim 2015) to compare LSTS against the best performing baselines at the end of 107 training steps and we observed statistically significant results (95% confidence). Thus, LSTS not only achieves a better success rate, but also converges faster (statistical significance result details in Appendix D). Time-to-threshold metric is defined as the difference in number of interactions between two approaches to reach a desired performance (Narvekar et al. 2020). The time-to-threshold between LSTS and the best-performing baseline DiRLc is 1.34 \(\times\) 106 interactions for 95% success rate.

**LSTSct (LSTS + Cont. Training) - Gridworld Domain**

In LSTS, while learning a policy for Task\((q,p)\), we reinitialize the environment to a random initial environmental state \(s \sim S_0\) once the agent reaches a state where the propositions \(b_{(q,p)}\) hold true. To answer the question Q2, instead of resetting the environment after reaching such a state where \(b_{(q,p)}\) hold true, we let the Teacher agent sample a task (let’s say Task\((p,r)\)) from the set \(X[p] \setminus DT\), where \(X\) is the adjacency matrix for the graph, and \(DT\) is the set of Discarded Tasks, as defined in Algo. 1. This helps the agent continue its training by attempting to learn a policy \(\pi_{(p,r)}\) for Task\((p,r)\) while simultaneously learning a separate policy \(\pi_{(q,p)}\) for Task\((q,p)\). If the agent fails to satisfy Task\((p,r)\), we reset the environment to state \(s \sim S_0\). Otherwise, the agent continues its training until its trajectory satisfies the high-level objective \(\phi\). We call this approach LSTSct (Detailed algo in Appendix A). Results in Table 1 demonstrate that this approach improves sample efficiency by reducing the number of interactions required to learn a successful policy for the gridworld task, with a time-to-threshold metric of 2.7 \(\times\) 105 interactions as compared to LSTS.

**5.2 LSTS and LSTSct - Robotic Domains**

To answer (Q3), we test LSTS and LSTSct on two simulated robotic environments with high interaction cost. The task in Fig. 2a has the following SPECTRL objective:

\[
\phi^\text{navigation} := (\text{achieve(Key}_1) \text{or achieve(Key}_2); \\
\text{achieve(Goal)} \text{ ensuring}(\neg \text{Lava})
\]

In this task, the agent (a simulated TurtleBot) needs to collect any of the keys (yellow blocks) present in a \([3m, 3m]\)\) continuous environment before reaching the goal position (gray block). At all times, the agent needs to avoid the lava object (red wall) present in the center. 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. The pick-up and drop actions have effects similar to the gridworld domain. The robotic domain is more complex as objects can be placed at continuous locations. The agent receives an ego-centric image view of the environment (top-right corner of Fig. 2a), which makes the task partially observable in nature and more complex to get a successful policy. The RL agent is described in Sec. 5.1.

The second environment (Fig. 2c) consists of a simulated robotic arm pushing two objects to their target locations (Gallouédec et al. 2021) with the SPECTRL formula:

\[
\phi^\text{manipulation} := \text{achieve(p}_1) ; \text{achieve(p}_2)
\]

where \(p_1\) and \(p_2\) are the atomic propositions for ‘push-object-1’, ‘push-object-2’. The robot has continuous action parameters for moving the arm and a binary gripper action (close/open). An episode begins with the two objects randomly initialized on the table, and the robotic arm has to push these two objects to its final location. The agent receives its current end-effector pose, positions and velocities of the two objects, and the desired goal position for the two objects. For this task, we use the Deep Deterministic Policy Gradient with Hindsight Experience Replay
We proposed \textit{LSTS}, a framework for dynamic task sampling for RL agents using the high-level SPECTRL objective coupled with the Teacher-Student learning strategy. Our approach formulates each transition of the DAG as a sub-task and identifies promising transitions in the DAG that have the potential to satisfy the high-level objective sample efficiently. Through experiments, we demonstrated that \textit{LSTS} accelerates learning when the high-level objective is correct and available, converging to a desired success rate quicker as compared to other curriculum learning and automaton-guided RL baselines. Furthermore, \textit{LSTS} does not rely on a human-guided dense reward function.

\textbf{Limitations & Future Work:} Our future plans involve incorporating scenarios where obtaining a precise SPECTRL specification is challenging. If the provided specification does not contain any feasible paths, then the algorithm will not be able to invent new feasible paths. A possible solution to this limitation would involve modifying the reward function for each sub-task so that the agent learns policies that enable the agent to reach new high-level states that are not represented in the high-level specification. As an extension, we would like to explore biasing away from sub-tasks rather than completely discarding them once the target node is reached, so in the limit, optimal policies can be obtained.

\textbf{Acknowledgements}

A portion of this work was conducted in the Multimodal Learning, Interaction, and Perception (MuLIP) Lab at Tufts University, University of Texas at Austin, Georgia Tech Research Institute, and the University of Colorado Boulder, with support from the Air Force Research Lab under contract FA875022C0501.
References


Grzes, M. 2017. Reward shaping in episodic RL.


