Relating Goal and Environmental Complexity for Improved Task Transfer: Initial Results

Sunandita Patra[1*] **Paul Rademacher**[3] **Kristen Jacobson**[3] **Kyle Hassold**[1,3]

Onur Kulaksizoglu[1] **Laura Hiatt**[3] **Mark Roberts**[3] **Dana Nau**[1,2]

[1]**Dept. of Computer Science and** [2]**Institute for Systems Research, Univ. of Maryland, College Park, MD, USA**
[3]Navy Center for Applied Research in AI, Naval Research Laboratory, Washington, DC, USA
[1]{patras}@terpmail.umd.edu {kulaksor, nau}@umd.edu [3]{first.last}@nrl.navy.mil

Abstract

The complexity of an environment and the difficulty of an actor's goals both impact task transfer in Reinforcement Learning (RL). Yet, few works have examined using the environment and goals in tandem to generate a learning curriculum that improves transfer. To explore this relationship, we introduce a task graph that quantifies the environment complexity using *environment descriptors* and the goal difficulty using *goal descriptors*; edges in the task graph indicate a change in the environment or the goal. For a delivery environment with up to ten skills, we introduce an algorithm that generates a *Task-Graph Curriculum* to train policies using the task graph and demonstrate that a planner can execute these trained policies to achieve long-horizon goals in increasingly complex environments. We also evaluate the task graph in two synthetic environments where we control environment and goal complexity. Our results demonstrate that (1) the task graph promotes skill transfer in the synthetic environments and (2) the Task-Graph Curriculum trains nearly perfect policies and does so significantly faster than learning a policy from scratch.

1 Introduction

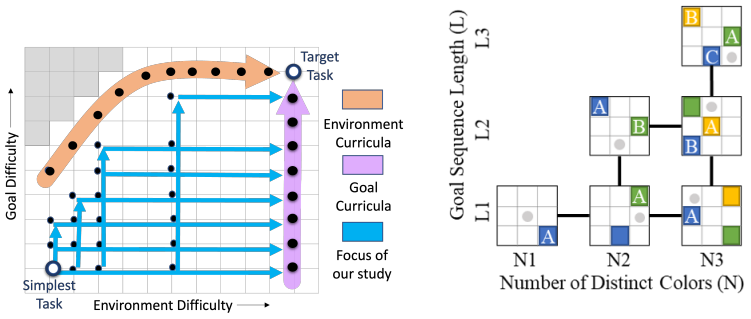
Using (sub)goals is a hallmark of intelligent behavior [1, 2, 3, 4, 5, 6]. In most AI planning literature (e.g., [7]), a goal is a partial world state an actor needs to achieve; that is, actors are told *explicitly* what goal to accomplish and the key task for planning is synthesizing a plan – a sequence of actions – to achieve the goal. Yet, most RL literature elicits behavior *implicitly* through the reward, though there are some recent initiatives that combine planning with RL (e.g., [8, 9, 10, 11]) or that provide abstractions similar to goals (e.g., [12, 13, 14])

Recent work by Patra et al. [15] links a goal to a reward using a data structure called a *goal skill*, which they formalize as an RL Option [16] with symbolic preconditions and effects. A collection of goal skills forms a *goal-skill network* (GSN), which we define in Section 2. Using a GSN allows a human expert to provide guidance for learning.

In this paper, we focus on the problem of training the goal skills of a GSN. The adaptability of a goal skill depends on the environment complexity and the goal complexity. Harder environments and more complex goals will tend to require learning with a more involved curriculum than learning each goal independently. We develop a process for developing curricula [17] of easier to harder tasks

[*]Work performed while at a postdoc at UMD; author is currently working at JP Morgan Chase

NeurIPS 2023 Workshop on Generalization in Planning (GenPlan 2023).



(a) (b)

Figure 1: (a) A notional plot showing several possible task-graph curricula (each path from the simplest to the target task in the space of blue arrows). (b) An example task graph for the VisitColors environment, described in Section 4.3, where the actor (gray circle) must use a single skill of going to a location to visit a sequence of colors (A, B, C) in order.

that use these two axes, as illustrated in Figure 1(a). The environment curricula (in orange along the top) only *implicitly* includes goals along the top but does not address distinct goal complexities as learning opportunities while the goal-only curriculum (in lavender on the right) does not vary the environment.

Our hypothesis is that training goal skills using a curriculum constructed from the blue paths of Figure 1(a) (left) will result in faster training on the target task. We assess this hypothesis using a step-wise approach for gridworld domains where we: demonstrate that the approach works for our target environment that has 10 goal skills (Section 4.1), demonstrate that the approach works for a simpler environment that alternates two goals skills (Section 4.2, and then construct a synthetic domain on a single goal skill to better control goal and environmental complexity (Section 4.3).

We represent the environmental complexity using a set of numeric values called *environment descriptors* and the difficulty of the (sub)goals using *goal descriptors*. We then construct a *task graph* where distinct tasks (nodes) in the task graph are separated by either an incremental change in the environment or the goal difficulty. Figure 1(b) shows a task graph for a synthetic environment where an agent (gray circle) must visit colored cells (A,B,C) in order; this problem is studied further in Section 4.3. The graph is distinguished by the number of distinct colors, an environmental attribute, and the sequence length, a goal attribute. Each path through the task graph is a *task-graph curriculum*, so named because the goals of the agent and the length of time spent on each task in the graph are design decisions provided by a human expert.

The contributions of this paper include: (1) formalizing the representation of a task as a set of environment and goal descriptors, which we combine into a task graph (Section 3); (2) developing an algorithm that uses the task-graph to train up to 10 goal skills and demonstrating, via plan execution, these goal skills to achieve a complex goal in a delivery environment (Section 4.1); (3) demonstrating the benefits of a task graph on an environment with two goal skills (Section 4.2); and (4) developing a synthetic domain that controls environment and goal complexity for a single goal skill to assess various curricula in the task graph (Section 4.3). Our results show that using the task graph to train goal skills results in faster training (usually through transfer). Although our results are limited to grid world domains, they show promise for future work that explores applications in continuous domains such as robotics.

2 Background: Goal Skills and the Goal Skill Network

Patra et al. [15] define a goal skill and Goal Skill Network that links a hierarchical goal network with the RL Options framework. In that work, goals are defined as a symbolic state of the world. A *goal skill* $g_\pi = (head, \breve{g}, \breve{\pi}, \breve{\mathcal{C}})$ is a tuple where *head* includes the *name* and *parameters* of the goal skill, $\breve{g}$ contains the goal the skill is meant to achieve, $\breve{\pi}$ is a goal-specific *policy* for this goal skill, and $\breve{\mathcal{C}}$ is a non-empty set of conditions that should be satisfied while executing $\breve{\pi}$. $\breve{\mathcal{C}}$ is partitioned into

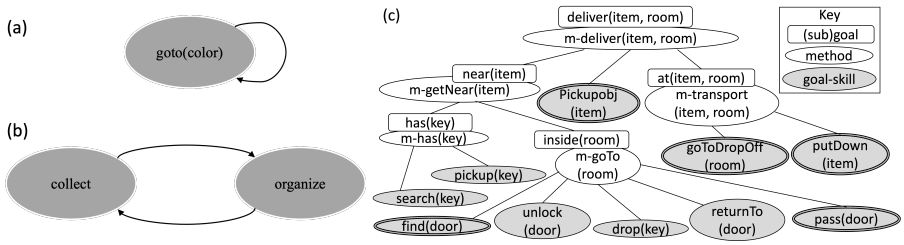


Figure 2: The Goal Skill Networks (GSNs) for (a) the VisitColors environment of Section 4.3; (b) the Snake environment of Section 4.2; (c) the KeyCorridor from Patra et al. (2022)[15] of Section 4.1.

$\{\mathcal{C}_{\text{start}}, \mathcal{C}_{\text{during}}, \mathcal{C}_{\text{end}}\}$, where $\mathcal{C}_{\text{start}}$ corresponds to the INIT of an RL Option, $\mathcal{C}_{\text{end}}$ corresponds to the TERM of an RL Option, and $\mathcal{C}_{\text{during}}$ indicates invariants during the execution of a skill. For example, a goal skill for the pickup(key) skill in Figure 2(c) would be (head: pickup(key), $\breve{\text{g}}$: `holding(key)`, $\mathcal{C}$: start+during:`holding(nothing)`, end:`holding(key)`, $\breve{\pi}$: pickup-key).

Patra et al. [15] further define a Goal Skill Network (GSN), which they use to generate hierarchical plans that have internal nodes describing what needs to be done with leaf nodes as goals skills. Figure 2 shows three GSNs used in this paper. Figure 2(a) is a single goal skill for getting to a specific cell location (See Section 4.3). Figure 2(b) uses two skills to solve a game (Section 4.2). Figure 2(c) is used in Section 4.1 and is from the experiments by Patra et al [15][2].

3 The Task Graph and Task-Graph Curriculum

Our concept of the task graph builds on prior work in transfer learning, specifically, the work of [18], [19] and [20] (see Related Work), which defined a set of related transfer tasks as a set of MDPs. Let a transfer domain $\mathscr{D}$ be a set of MDPs that represents the universe of possible tasks an actor can encounter. Further, let $t_m \in \mathscr{D}$ be a specific task from $\mathscr{D}$, where m designates the task descriptor. m describes t_m in terms of problem-specific features, and, in prior work, these task features were a set of environmental features.

In this work, we factor a task descriptor into an environment descriptor and a goal descriptor. For simplicity, we restrict all features of t_m to be non-negative integers (i.e., $\mathbb{Z}_0$). For example, a task descriptor in Figure 1(b) is a combination of a column (i.e., the environment) and a row (i.e., the goal). The task descriptor is underspecified with respect to the environment, in the manner of Unsupervised Environment Design [21, 22], so the initial state of a given task may differ from one environment instantiation to another. An environment descriptor is a vector of features that describe the environment.

Definition 1. *(Environment Descriptor) Let $\boldsymbol{\phi} = (\mathbb{Z}_0)^j$ be a vector of j features, where $\phi_i = \{0, .., k_i\}$ indicates the ith feature and k_i is a feature-specific maximum value. An* environment descriptor $\boldsymbol{env} = \phi$ *describes an environment in $\mathscr{D}$.*

Example Environment Descriptor. For Figure 1(b), ϕ consists of the number of colors in $\{1, 2, 3\}$. Here, k_i is 3 for the maximum number of colors, though it is easy to see how the max could be increased for this example. Together, these features result in $\boldsymbol{env} = (\#colors)$.

A goal descriptor is a vector of features that describe the goal; it may contain subgoals.

Definition 2. *(Goal Descriptor) Let $G = (g_1, .., g_j)$ describe an ordered set of j goal types the actor can achieve, where a specific goal $g_i = \{0, .., k_i\}$ indicates the ith goal type (e.g., `holding(item)`) with a maximum number of goals of that type being k_i. A* goal descriptor $\boldsymbol{g} = (\mathbb{Z}_0)^{|G|}$ *is a feature vector of length $|G|$ describing the number of goals of each goal type the actor should achieve.*

Example Goal Descriptor. For Figure 1(b), $|G| = 1$ and the environment descriptor $(N3)$ has three goal descriptors: $\boldsymbol{g_{L1}} = (1)$ or (`visit(blue)`), $\boldsymbol{g_{L2}} = (2)$ or (`visit(yellow), visit(blue)`), and $\boldsymbol{g_{L3}} = (3)$ or (`visit(green), visit(yellow), visit(blue)`). The colors to visit might change randomly with a new instance but will always be some subset of the number of colors.

[2]We thank the authors of that paper for sharing their code.

To specify goals, we assume a function that maps the state of the world to a symbolic state. For example, an actor achieves the goal `at(dest)` when its position equals the location parameter. For our study, this is accomplished with the implementation of problem-specific functions written in Python. We link a goal instance with the policy using the goal skill. The goal ordering for G is specified as part of the GSN, which can be provided by an automated planner, as we do in Section 4.1.

Now that we have defined the environment descriptor and the goal descriptor, we define the task descriptor m for a given $task_m \in \mathscr{D}$. A task descriptor is composed of the environment descriptor (Definition 1) and a goal descriptor (Definition 2).

Definition 3. *(Task Descriptor) A* task descriptor $\boldsymbol{m}^{(g)}_{(env)}$ *denotes a vector describing* $task_m \in \mathscr{D}$.

Example Task Descriptors. For Figure 1(b), there are six tasks, where the top right task would be $\boldsymbol{m}^{(3)}_{(3)}$, or a shorthand $\boldsymbol{m}$. Throughout we will use an abbreviated form of $env|goal$ or $3|3$.

Task descriptors can be joined in a graph that we call the task graph.

Definition 4. *(Task Graph) A Task Graph* $M = (V, E)$ *is a directed acyclic graph where* $V = \{v^1, v^2, .., v^{(|\mathscr{D}|)}\}|\ v^k = t_k$*, where k is a task descriptor, and* $E = \{(v^i, v^j)|\delta(\boldsymbol{m}_i, \boldsymbol{m}_j) \leq \theta \wedge i \neq j\}$.

In principle, δ can be any distance metric (e.g., the $L1$ distance). In practice, distances between goals and environments are problem-specific and hard to estimate. So we use unit distances with $\theta = 1$ for this study and leave examining other metrics for future work. Our focus in this study on achievement objectives and integer-valued features has not been too limiting, though future work may require maintenance objectives or continuous feature vectors, especially for applications such as robotics.

4 Results

We examine three grid world environments. KeyCorridor in Section 4.1 demonstrates a blend of up to 10 goal skills using an automatically constructed curriculum. Snake in Section 4.2 demonstrates a simple domain that alternates two goal skills. Finally, VisitColors in Section 4.3 is a synthetic domain where we can control features of the task graph. The results show that training with a task-graph dramatically improves performance.

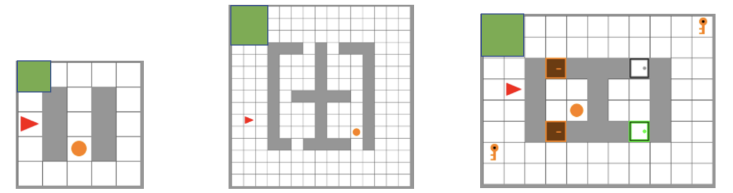
4.1 Training up to 10 goal skills in the KeyCorridor Environment (Automated Curriculum)

In KeyCorridor (see Figure 3) an actor (red triangle) must navigate rooms and hallways to deliver an orange ball to the green square. The discrete action space includes navigation (turn left, turn right, and move straight) and interaction (pick up, put down, and use object). The descriptor $\boldsymbol{env} = (\#rooms, \#items, status, size)$ where $status \in \{1, 2, 3\}$ for 1: no doors, 2: unlocked door, and 3: locked door. $size \in \{3, 4, 5\}$ indicates the room size.

We consider two test scenarios for this study. **Scenario 1** uses the source task of Figure 3(a) and the target task of Figure 3(b) and requires the actor to navigate to the ball, pick it up, move it to the top-left corner and deposit the ball. The curriculum includes the double-bordered skills in Figure 2(c): `find(door)`, `pass(door)`, `pickupobj (item)`, `goToDropOff(room)`, and `putDown(item)`. **Scenario 2** uses the same source task Figure 3(a) with the target task of Figure 3(c) where the actor must search for a key and unlock the door before entering the room. The actor can carry only one object at a time and must drop the key to carry the ball. The actor, key, ball and doors are initially placed in random locations. The curriculum involves all goal skills in Figure 2(c), which include all from Scenario 1 plus five additional goal skills: `search(key)`, `pickup(key)`, `unlock(door)`, `drop(key)`, `returnTo(door)`.

We develop a *Task-Graph Curriculum* using the task graph M (Definition 4). This procedure enables a policy within each goal skill to transfer to environments leading to the target task. The Task Curriculum proceeds in two phases: (1) it orders the task descriptors and (2) it orders and learns goal skills for each task descriptor.

Automatically constructing a Task-Graph Curriculum. We summarize here our algorithm for constructing a Task-Graph Curriculum; see appendix for details. Let $\boldsymbol{u}$ be the easiest task to learn and $\boldsymbol{f}$ the hardest. The algorithm finds the shortest path from $\boldsymbol{u}$ to $\boldsymbol{f}$ using greedy best-first search (GBFS) on the task graph, M. The heuristic used to choose among all neighbors at an equal distance



(a) Task Descriptor of **Source**:
Env: 1 room with no doors
room size = 3
Goal: find(hallway)

(b) Task Descriptor of **Target 1**:
Env: 4 rooms with no doors
room size = 5
Goal: deliver(ball)

(c) Task Descriptor of **Target 2**:
Env: 1 locked room, 1 unlocked room
room size = 3
Goal: deliver(ball)

Figure 3: The two scenarios of the Key Corridor environment considered in this study showing the starting (top) and final task descriptors (bottom). The environment descriptor is $\boldsymbol{env} = (\#rooms, \#items, status, size)$ and the goal skill network is from Figure 2(c).

from a node is the neighbor which is nearest to $\boldsymbol{f}$. As the algorithm proceeds along a path in M, it trains all the goal skills with positive values in the goal descriptor g of the current task descriptor, $\boldsymbol{m}$.

Plan Execution with Goal Skills. In order to execute an incoming stream of tasks, we developed an algorithm (full details in appendix) that takes a stream of tasks t_m (with task descriptors $\boldsymbol{m}(env|goal)$) and a sub-set of the trained set of trained goal skills Π. It queries a planner for a *plan*, i.e., a correct sequence of goal skills that accomplish $goal$ in the current state of the environment with descriptors env. It reads an incoming stream of tasks in a loop. For each new task, it obtains a *plan* for the current state. Then it iterates over the goal skills in the plan querying the next action from the trained policy of the current goal skill. It moves to the next goal skill in the plan if the current one succeeds and replans if the action from the current goal skill fails.

Evaluation. We evaluate five different curricula: (1) a goal only curriculum ΔGoal Only that trains all goal skills for the target; and (2) an environment only curriculum ΔEnv Only that trains one policy for the most difficult goal and only increases the environment descriptor. (3) the Task-Graph Curriculum constructed using our algorithm; (4) Three random permutations of the task curriculum; (5) the reverse of our proposed task curriculum; We give a *reward* of 1 when `deliver(item)` is successful, -1 when `deliver(item)` fails, and a penalty of 0.01 for each step.

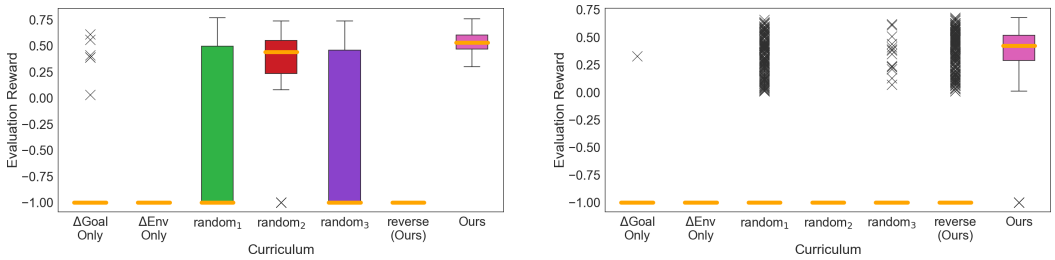
Figure 4 shows the evaluation on the target tasks for our two scenarios. In Scenario 1, our task curriculum has a length of 25 from the source to the target task. In all plots, each box summarizes the reward values for 100 runs and the bold line indicates the median score. The bottom and top box lines indicate the 25th and 75th quantiles. If not visible, they align with the median. The '$\times$' marks show outliers. We significantly outperform the two baselines, ΔGoal and ΔEnv (the left two columns in the plot corresponding to (1) and (2) above). In Figure 4(a), we see that ΔGoal is able to achieve a few successful runs, whereas the trying to learn `deliver(item)` as one policy in ΔEnv fails in all cases resulting in a reward of -1.

We wanted to compare the path in the task graph suggested by our curriculum generation algorithm to other orderings. To do this, we compare the performance with random and reverse orderings of the task descriptors in our curriculum. We see that one of the random orderings, random$_2$, performs fairly well compared to others. This is because it has a good overlap with our task curriculum in the initial stages. As expected, the reverse ordering performs the worst because it tries to learn the hardest task first and goes back to easier tasks.

In Scenario 2, our task curriculum has a length of 50. We see in Figure 4(b) that the impact of using our task curriculum is even more pronounced compared to Scenario 1. All curricula except ours have a median reward of -1 with ΔEnv and random$_2$ performing the worst. One of the random orderings, random$_1$, and the reverse ordering has the maximum number of successful outliers.

Our examination of the training patterns (see appendix for all training curves and detailed analysis) shows that our task curriculum learns significantly faster than other training strategies (1-5 above).

In summary, we observe that having distinct policies for each goal skill led to strong transfer learning between environments for KeyCorridor. Our task curriculum significantly outperforms other curricula which consider, either (a) changes only in the environment descriptor, (b) changes only in the goal descriptor, or (c) random/reverse permutations of the task curriculum.



(a) (b)

Figure 4: Comparison of the evaluation reward for the final task in Key Corridor for the two scenarios of Figure 3. Higher values of reward denote better performance. Our task curriculum significantly outperforms other curricula. ΔGoal curriculum trains the goal skill network directly for the final task. ΔEnv tries to learn deliver(item) directly without using the goal skill network. (a) Scenario 1: Evaluation reward on the final (hardest) task with descriptor, (4, 1, 0, 5)|`deliver(ball)`. (b) Scenario 2: Evaluation reward on the final (hardest) task with descriptor, (2, 1, 2, 3)|`deliver(ball)`.

4.2 Training Alternating Goal Skills in the Snake Environment (Alternating Curriculum)

Our next study examines the problem where two goal skills must alternate to achieve an objective. The Snake environment involves an actor eating apples and staying alive as long as possible in a 2D grid. Actions include moving up, right, down, left. Each time the head "eats" an apple the length grows by one and the body follows the path of the head, similar to a conga line. A task fails when the head runs into the body or the wall, and it is successful when the entire grid is filled with the body. Figure 5(a) shows an example of initial states for a 10x10 environment with a length of 25. The environment descriptor is (N, L), where the size of the grid is $N \times N$, and the length is L.

One challenge with this environment is keeping the body organized as length increases. A good heuristic is circling the head around the perimeter or doubling back on the body often, but doing this may compete with apple consumption. Thus there are two essential skills for the environment, `organize(body)` and `collect(apple)`, and these skills have different, complementary behaviors. Our study focuses on examining how the goal skills interact as environmental complexity increases.

Learning Snake with a single policy rewarded to collect apples and stay organized does not seem to do well. Summarizing our findings (full details in appendix), the snake does not achieve very high success and we did not notice a substantial difference in transfer to harder environments at snake lengths of only 10 or 15. Even using a reward structure of the number of collected apples (i.e., the final score) resulted in a score of 35 apples in a 10x10 grid when the best score is 100.

At first, we thought this might just mean that we needed to alternate training for each skill. But Figure 5(b) demonstrates what happens when the training collect (blue lines) is alternated with organize (red lines); here, we trained skills with an initial length of 3 and growth turned on for collecting apples where, for organize, the snake grows every 15 timesteps. Training one skill regresses the other, suggesting that each goal skill must be trained in isolation.

Training each skill separately works much better. Instead of a combined reward structure, we reward collect 1 for each collected apple and organize for how long it stays alive. Figure 5(c) shows the results for transferring the collect skill through lengths of $5, 10, 25, 50, 75, 95$ where the episode ends when collecting the apple. As the length increases, the snake collects fewer apples overall (until length 95) but it is still able to collect apples. This is in contrast to our earlier experiments where moving past a length of 15 was challenging. The high values at length of 95 make sense when one considers that only 5 empty spots remain and the snake has very few choices; sampling can be very efficient. Collecting apples at longer lengths is still challenging, but this is likely due to the fact that organize needs to be leveraged at those points.

Figure 5(d) shows the results for transferring the organize skill to longer lengths $\in \{5, 10, 25, 50, 75, 95\}$. Similar to the previous results, the transfer is successful and exhibits positive recovery. Even at its worst (length 50, green), the snake is able to take about 400 steps in the environment compared to about 100 steps in the single-policy version.

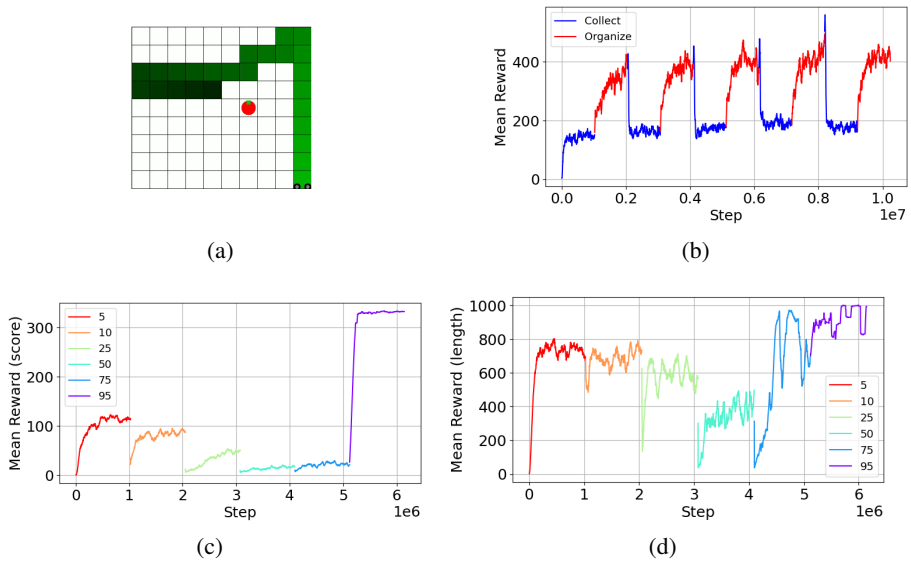


Figure 5: (a) Example 10x10 Snake problem. (b) Poor recovery when alternating the collect (blue) and organize (red) skills. (c) Transfer for the collect skill. (d) Transfer for the organize skill.

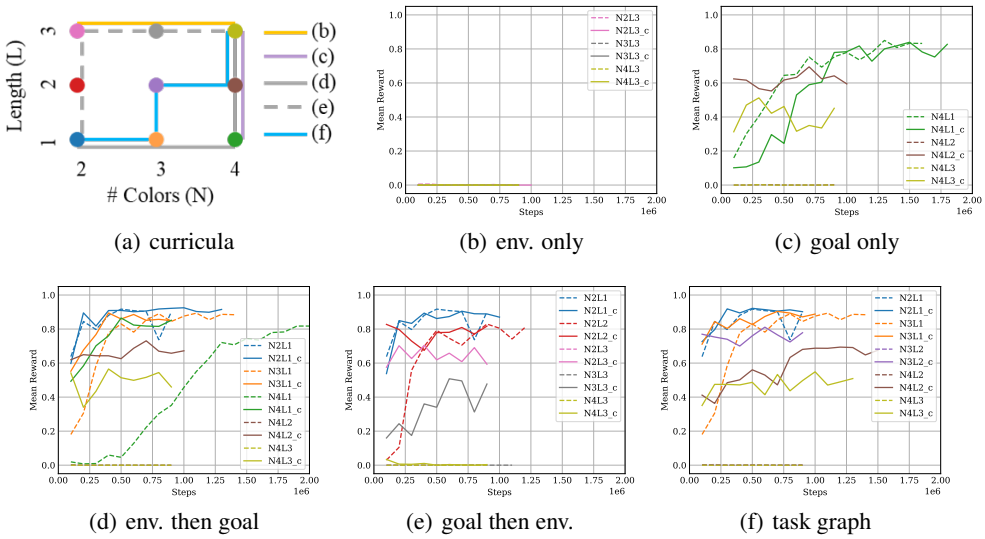
To summarize, the results in Snake suggest that simultaneously learning two implicit goals in a single policy, a practice commonly done in the RL literature, performs poorly in this domain and fails to transfer to harder environments. We provided a possible reason for this as resulting from the two skills causing catastrophic forgetting (see Figure 5(b)). In further experiments, we demonstrated that using distinct goal skills resulted in better overall performance. The evidence from these studies provides further weight that using goals to structure the reward presents a natural way to factor an RL problem when the objectives of the agent compete.

4.3 A Synthetic Domain: The VisitColors Environment (Manual Curriculum)

In this section, we use a synthetic domain to demonstrate how the task graph can be used to train a single goal skill. We created a custom MiniGrid [23] environment called VisitColors where the actor must visit colored floors in a specified, observable sequence. The actions are move forward, turn left, and turn right, and the observation combines a 2-D array of the grid from the actor's perspective, an encoding of the actor's direction, and an encoding of the current goal color. Figure 6(a) shows an example task graph for this environment and the specific curricula that we tested. The task descriptor is $N|L$, where N is the number of distinct colors and L is the sequence length. The agent receives the sparse reward of 1 only if the correct color order is achieved; reaching a color other than the next color terminates the episode with no reward. This environment allows comparison of curricula while varying N and L to control for environment and goal complexity, respectively.

Figure 6 shows the results of five curricula we explored. Each line is a particular task in the task graph NxLy where x indicates the number of colors and y indicates the sequence length. For each task, a dashed line shows the results of training from scratch (without transfer) while a solid line indicates continuing the curriculum from the top of the legend down. In all cases, the target task is $4|3$ (or N4L3). Each data point indicates the average undiscounted return of the policy over 400 random rollouts after a given training duration; note that different training runs have different durations due to the use of early-stopping criteria. See Section 1 of the appendix for complete algorithm details, as well as for additional plots visualizing standard deviation.

Each task (i.e., each vertex) in the task graph of Figure 6(a) has a unique color that matches the performance plots; this allows for easier comparison of the relative value of a curriculum for a task; For example, N4L2 (magenta) shows up in different points of curricula (c), (d), and (f). shows the specific curricula we test on a visual representation of the task graph. Figure 6(b) demonstrates that the typical curriculum of training only for environmental complexity does not perform well.



(a) curricula (b) env. only (c) goal only

(d) env. then goal (e) goal then env. (f) task graph

Figure 6: Comparison of curricula for VisitColors: (a) a visual representation on the task graph of the curricula tested where line colors match Figure 1(a) and vertices match the performance lines in (b)-(f); (b) environment only; (c) goal only; (d) environment then goal; (e) goal then environment; (f) task graph. See prose for details. The variance decreases as the reward approaches 1.

Figure 6(c) demonstrates improvement when using a goal only curriculum. It is evident here that a curriculum using goals works better than (b).

The bottom three plots show variations of varying environmental and goal complexity together in a curriculum. Figure 6(d) shows the result of increasing the environment complexity first and then the goal complexity. Early steps in the curriculum (N2L1 and N3L1) do not perform differently from the baseline, but the curriculum results in a noticeable improvement in N4L1, N4L2, and N4L3 when compared to (c), which is a suffix of (d). Figure 6(e) shows the result of increasing the goal complexity and then the environment complexity. The final task N4L3_c of (e) performs poorly against (c) or (d). However, (e) outperforms N2L3 and N3L3 from (b) using the prefix of {N2L1, N2L2}. Figure 6(f) shows the results of alternating a step in goal and environment complexity. This curriculum performs similar to Figure 6(d) and (c). Some steps recover more quickly (e.g., N3L1, N3L2) while others take slightly longer.

In summary, our use of a synthetic domain allowed us to control for environment and goal complexity for a simple, sparse-reward problem to evaluate different curricula. The results show that varying environmental and goal complexity together often result in better performance on the target task over simply varying the environment or goal alone.

5 Related Work

Curriculum learning for RL has a rich history, as evidenced by a recent survey by Narvakar et al. [24]. This existing literature has explored both environment and goal complexity, largely independently. To the best of our knowledge, no other approaches have explicitly considered both the environment and goal difficulty together to build a learning curriculum. For learning complex goals, there is work on hierarchical RL, RL options, and other approaches, such as [25, 26, 8]. For learning one goal in a variety of different environments, there is work on transfer learning [18, 19, 20]. However, there are several works that relate to factored representation, task descriptors, lifelong learning, transfer learning, RL, hierarchy and planning in different combinations.

Several approaches have examined factored representation of states, actions, or the reward structure, which overlap with the proposed task graph of this paper. For example, Mirsky et al. [27] developed a concept of task factorization of a task into the agent, environment, and mission. The mission is equivalent to the reward but also includes a hierarchical structure and sequential structure. Agent and

environment states are partitioned over the powerset of the world state while actions are partitioned as actions of the agent or events "taken" by the environment. They showed that these various ways of partitioning a curriculum impacted agent performance. This paper examines a particular kind of environment and mission. Another example is the work of Able et al. [28] that showed using a goal to bias learning can reduce sample complexity.

Our work builds on the use of **task descriptors** to construct automated curricula, which has been explored in a variety of important contexts (e.g., [29], [30], [19]). Much of this work has focused on learning an encoding for the tasks. As the evidence shows, this works incredibly well for situations where one wants to perform one-shot transfer, for lifelong learning, or where the tasks are not known in advance. Our approach complements this literature by allowing a human expert to provide a goal hierarchy that can dramatically speed up learning.

The use of hierarchical RL has the potential to accelerate learning because hierarchical actors can decompose problems into smaller subproblems while learning at multiple levels of abstraction [31, 32, 33, 34]. The options framework [16] provides methods for learning and planning using high-level actions, or options, in a *Semi-Markov Decision Process* (SMDP). Potentially, options that select other options as macro-actions in their policies can give rise to a hierarchical structure with arbitrary levels, but most work assumes a fixed 2-layer hierarchy in practice (e.g., [35]); very few works allow a 3-layer hierarchy (e.g., [31]). Our work does not presume the number of layers in the hierarchy. [36] demonstrated an approach to use symbolic option discovery. Our approach uses symbolic structure, but combining it with this approach could provide complementary benefits.

Andreas, Klein, and Levine ([26]) proposed a two-level hierarchy, where each subpolicy is trained with a different neural network. They test their algorithm in a similar grid-based RL environment where it performs better than monolithic, single-network approaches. This approach mainly differs from ours in the structure and the level of hierarchies. Additionally, their subpolicies are not trained by explicit terminal goal states, instead their subpolicies try to learn the termination states on their own. In contrast, our approach provides explicit termination states for each goal skill during training.

The Universal Value Function Approximator (UVFA) is a popular baseline method for training neural networks with multiple goals [37]. The UVFA learns RL policies by training two separate neural networks for goal and environment embeddings. This method has some downsides when the rewards are sparse in the environment. [31] mitigates this problem using a data augmentation technique called hindsight experience replay [38]. While all these approaches try to solve the multi-goal RL problem, their approaches don't include a hierarchy of goals and planning associated with them.

Konidaris et al. [39] proposed a technique that segments a demonstration trajectory into a chain of component skills (each skill is an abstracted goal). Chains from multiple trajectories are then merged into a skill tree. This work was extended to arbitrary goals by Baggaria et al. ([40]), who created Deep Skill Graphs. Future work will explore how to combine the goal-skill network with the skill graph or skill trees.

Another related line of work is the approaches that try to integrate planning and RL. [8] and [41] implemented a hierarchical RL framework with symbolic planning, which achieves better sample efficiency for training and makes it relatively easy to specify high-level goals. They do not consider different environment or goal complexities. Reprel[42] combines HTN planning and reinforcement learning for skills. They provide a relational RL framework for zero shot transfer across skills but do not consider different levels of environment complexity.

Curriculum learning is a well-known technique in RL. [17, 43] demonstrated that curriculum learning can reduce training times and improve a network's resilience to overfitting by gradually increasing the difficulty of samples as the network converges. Our proposed curriculum uses the task graph to partition and order the skills that need to be learned. [20] curriculum learning for source task creation demonstrated the benefit of building agent-specific source tasks. Our approach does not preclude the use of this focus, and future work should explore this connection. [44] automated curriculum generation based on task similarity leading to improved performance. While their focus was on implicit goals, their automated construction of the curriculum was a key benefit. Although many aspects of our proposed approach are automated, the hierarchy of the goal skill network must be constructed by hand. However, automated techniques for hierarchy learning exist and could complement our curriculum approach. Future work should explore this possibility.

Several studies, including [45], [46], and [47], primarily concentrate on decomposing target tasks into smaller waypoints and training them within a fixed environment, however, there is no notion of changing the environment difficulty. In contrast, we are trying to learn the tasks in increasing levels of environment difficulty.

6 Summary and Future Work

Our results showed that the task graph is a powerful conceptual tool for designing goal-based RL agents that combine planning and execution and that the task graph curriculum shows promise as a way to train goal skills. Although we focused in this paper on externally provided goals and curricula, our work lays a foundation for future work to automate the construction of a task graph using techniques such as option discovery ([48, 49, 50]) and planning model acquisition [51].

We argued that goals and environment are strongly correlated and that a curriculum that leverages this relationship will learn better policies than a curriculum focused on one aspect alone. We formalized a task graph composed of environment descriptors and goal descriptors. We examined the performance of various curricula in light of the task graph for three environments: KeyCorridor, Snake, and VisitColors. We proposed two algorithms that (1) automatically generate a task curriculum from the task graph to guide the training of goal skills, and (2) use a planner to execute the trained goal skills. During execution in KeyCorridor, the most complex environment, the actor was able to accomplish difficult tasks using a hierarchical planner and the learned goal skills. Our experiments demonstrated that using a task graph curriculum often resulted in better performance than training in only the environment or goal complexity alone. Our next steps include using multi-task learning strategies to transfer knowledge between goal skills. We also look forward to learning the goal skill network automatically without involving a human expert.

Acknowledgements

This work has been supported for UMD in part by ONR grant N000142012257 and AFRL contract FA8750-23-C-0515. PR, KJ, KH, LH, and MR thank ONR and NRL for funding this research. The views expressed are those of the authors and do not reflect the official policy or position of the funders.

References

[1] Nick Hawes. A survey of motivation frameworks for intelligent systems. *JAIR*, 175(5-6):1020–1036, April 2011.

[2] David W. Aha. Goal reasoning: Foundations, emerging applications, and prospects. *AI Magazine*, 39(2):3–24, 2018.

[3] Vikas Shivashankar. *Hierarchical Goal Networks: Formalisms and Algorithms for Planning and Acting*. PhD thesis, Computer Science Dept., University of Maryland, 2015.

[4] Herbert A Simon and Allen Newell. Human problem solving: The state of the theory in 1970. *American psychologist*, 26(2):145, 1971.

[5] A Ram and DB Leake. Learning, goals, and learning goals, goal driven learning. *Art Intel Rev*, pages 9387–422, 1995.

[6] Michael T Cox and Chen Zhang. Mixed-initiative goal manipulation. *AI Magazine*, 28(2):62–62, 2007.

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

[8] León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A. McIlraith. Symbolic Plans as High-Level Instructions for Reinforcement Learning. *Proc. ICAPS*, 30:540–550, June 2020.

[9] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208, 2022.

[10] Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomas Lozano-Perez, Leslie Pack Kaelbling, and Joshua Tenenbaum. Inventing relational state and action abstractions for effective and efficient bilevel planning. *arXiv preprint arXiv:2203.09634*, 2022.

[11] Rohan Chitnis, Tom Silver, Joshua B Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Glib: Efficient exploration for relational model-based reinforcement learning via goal-literal babbling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11782–11791, 2021.

[12] David Abel, Dilip Arumugam, Kavosh Asadi, Yuu Jinnai, Michael L Littman, and Lawson LS Wong. State abstraction as compression in apprenticeship learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3134–3142, 2019.

[13] David Abel, Nate Umbanhowar, Khimya Khetarpal, Dilip Arumugam, Doina Precup, and Michael Littman. Value preserving state-action abstractions. In *International Conference on Artificial Intelligence and Statistics*, pages 1639–1650. PMLR, 2020.

[14] Haotian Fu, Shangqun Yu, Saket Tiwari, George Konidaris, and Michael Littman. Meta-learning transferable parameterized skills. *arXiv preprint arXiv:2206.03597*, 2022.

[15] Sunandita Patra, Mark Cavolowsky, Onur Kulaksizoglu, Ruoxi Li, Laura Hiatt, Mark Roberts, and Dana Nau. A hierarchical goal-biased curriculum for training reinforcement learning. In *The International FLAIRS Conference Proceedings*, volume 35, 2022.

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

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

[18] David Isele, Eric Eaton, Mark Roberts, and David W Aha. Modeling consecutive task learning with task graph agendas. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1965–1967, 2018.

[19] Mohammad Rostami, David Isele, and Eric Eaton. Using task descriptions in lifelong machine learning for improved performance and zero-shot transfer. *JAIR*, 67:673–703, 2020.

[20] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source task creation for curriculum learning. In *Proceedings of the 2016 international conference on autonomous agents & multiagent systems*, pages 566–574, 2016.

[21] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 13049–13061. Curran Associates, Inc., 2020.

[22] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *Proceedings of the 39th International Conference on Machine Learning*, page 17473–17498. PMLR, June 2022.

[23] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for OpenAI gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[24] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020. Citation Key: JMLR:v21:20-212.

[25] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[26] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. *CoRR*, abs/1611.01796, 2016.

[27] Reuth Mirsky, Shahaf S. Shperberg, Yulin Zhang, Zifan Xu, Yuqian Jiang, Jiaxun Cui, and Peter Stone. Task factorization in curriculum learning. In *Decision Awareness in Reinforcement Learning Workshop at ICML 2022*, June 2022.

[28] David Abel, David Hershkowitz, Gabriel Barth-Maron, Stephen Brawner, Kevin O'Farrell, James MacGlashan, and Stefanie Tellex. Goal-based action priors. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 25, pages 306–314, 2015.

[29] Jivko Sinapov, Sanmit Narvekar, Matteo Leonetti, and Peter Stone. Learning inter-task transferability in the absence of target task samples. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, page 725–733, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.

[30] Felipe Leno Da Silva and Anna Helena Reali Costa. Object-oriented curriculum generation for reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, page 1026–1034, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

[31] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. *Proc. ICLR*, 2019.

[32] Thomas G Dietterich. The maxq method for hierarchical reinforcement learning. In *Proc. ICML*, volume 98, pages 118–126, 1998.

[33] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1):41–77, 2003.

[34] Nicholas K Jong and Peter Stone. Hierarchical model-based reinforcement learning: R-max+ maxq. In *Proc. ICML*, pages 432–439, 2008.

[35] Jhelum Chakravorty, Patrick Nadeem Ward, Julien Roy, Maxime Chevalier-Boisvert, Sumana Basu, Andrei Lupu, and Doina Precup. Option-critic in cooperative multi-agent systems. In *Proc. AAMAS*, pages 1792–1794, 2020.

[36] Mu Jin, Zhihao Ma, Kebing Jin, Hankz Hankui Zhuo, Chen Chen, and Chao Yu. Creativity of ai: Automatic symbolic option discovery for facilitating deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(66):7042–7050, Jun 2022.

[37] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proc. ICML*, pages 1312–1320, 2015.

[38] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Proc. NeurIPS*, 2017.

[39] George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. *The Int'l Journal of Robotics Research*, 31(3), 2012.

[40] Akhil Bagaria, Jason K. Senthil, and George Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *Proc. ICML*, page 521–531, July 2021.

[41] León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila McIlraith. Symbolic planning and model-free reinforcement learning: Training taskable agents. In *Proc. RLDM*, pages 191–195, 2019.

[42] Harsha Kokel, Arjun Manoharan, Sriraam Natarajan, Balaraman Ravindran, and Prasad Tadepalli. Reprel: Integrating relational planning and reinforcement learning for effective abstraction. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 533–541, 2021.

[43] Çağlar Gülçehre and Yoshua Bengio. Knowledge matters: Importance of prior information for optimization. *The Journal of Machine Learning Research*, 17(1):226–257, 2016.

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

[45] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[46] Lunjun Zhang, Ge Yang, and Bradly C Stadie. World model as a graph: Learning latent landmarks for planning. In *International Conference on Machine Learning*, pages 12611–12620. PMLR, 2021.

[47] Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. Goal-conditioned reinforcement learning with imagined subgoals. In *International Conference on Machine Learning*, pages 1430–1440. PMLR, 2021.

[48] Marlos C Machado, Marc G Bellemare, and Michael Bowling. A laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning*, pages 2295–2304. PMLR, 2017.

[49] Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In *International conference on machine learning*, pages 316–324. PMLR, 2014.

[50] Akhil Bagaria and George Konidaris. Option discovery using deep skill chaining. In *International Conference on Learning Representations*, 2020.

[51] Ethan Callanan, Rebecca De Venezia, Victoria Armstrong, Alison Paredes, Tathagata Chakraborti, and Christian Muise. Macq: A holistic view of model acquisition techniques. In *ICAPS Wksp. on Knowledge Engineering for Planning and Scheduling (KEPS)*, 2022.

Appendix

In this appendix, we provide more details of our methodology and experiments. Section 1 discusses more detail for the VisitColors environment. Section 2 provides additional detail for the Snake environment on several sets of experiments summarized in the paper and extends the results of the paper. Section 3 details the algorithms we introduced in the paper as well as provides more details on the evaluation in the KeyCorridor environment.

1 Further details of the Visit-Colors Environment

We used the Stable-Baselines3[1] Python package on a server with 255 cores at 1.5GHz, 2TB RAM, and 8x NVIDIA A100 80GB GPUs. We trained all task graph nodes with the same policy architecture and learning parameters, which were tuned with Optuna[2]. The policy is a feed-forward neural network that flattens and stacks the separate observation tensors before processing with four sequential hidden layers of 256 units, each followed by rectified linear unit nonlinearities. We trained the policies using the PPO algorithm [1] with a separate value network with the same hidden layer specifications as the policy. The algorithm collects 4096 steps of state-action-reward experience before updating network parameters, with 10 epochs of size 512 batches per update. It also performed generalized advantage estimation[2] with a discount factor $\gamma = 0.99$ and $\lambda = 0.95$. The policy optimization loss function includes value error and entropy summands, scaled by 0.5 and 0.0001, respectively.

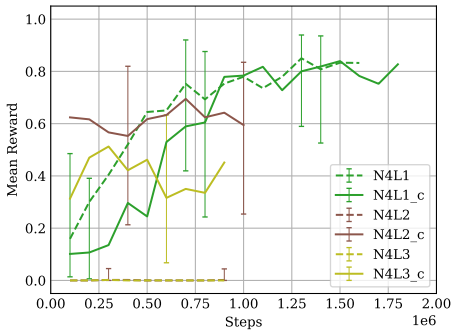
The plots in the VisitColors section demonstrate reward evaluated on a separate environment instance, averaging undiscounted return over 400 episodes. The plots show results of evaluating the polices after every 100,000 steps of training experience, with early training termination if a new best is not realized after three consecutive checks. To visualize the variability of evaluation rewards, Figure 1 repeats several curricula plots from Figure 3 of the paper with standard deviation error bars with the average reward; only select bars are plotted for visibility. Even well-performing policies exhibit high variance due to the sparse reward structure of the VisitColors environment. The lines of each figure are best understood by comparing pairs of the same color where the dashed line indicates learning without a curriculum and the solid line indicates learning with a curriculum. For example, consider the final task N4L3 for Figure 1(a), where the orange and green dashed lines (along the bottom) indicate failure to learn but the solid lines of the same color show that transfer was helpful. In contrast, the purple line (target task) in Figure 1(b) shows the benefit of the environment-then-goal curriculum but the dashed green line (i.e., middle of the curriculum) shows that the baseline approach is still performing well for the N4L1 task; this is not surprising because the sequence length is still one even though there are four colors. Figure 1(c) shows that the goal-then-environment curriculum (solid line) did not perform better than the baseline (dashed line) on the target task. The task-graph curriculum shown in Figure 1(d) shows that the performance of the N4L3 target task benefited from the curriculum.

2 Further details of the Snake Environment

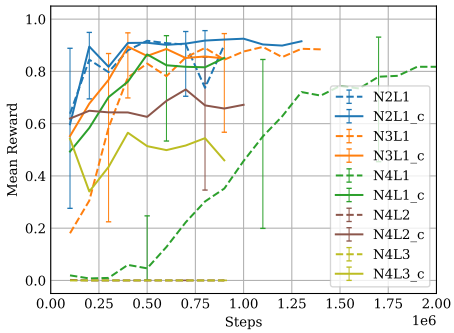
As discussed in the paper, Snake was a particularly challenging environment to demonstrate the value of the task-graph curriculum because it turned out we needed to train two goal skills – collect and

[1] https://stable-baselines3.readthedocs.io/

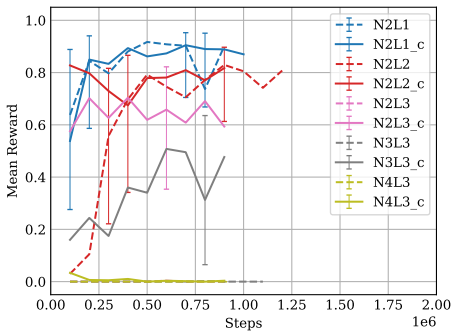
[2] https://optuna.readthedocs.io/



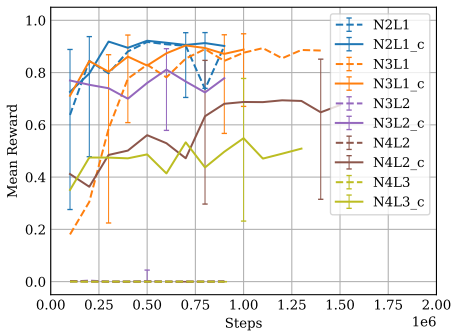
(a) Fig. 3 (c) from paper with s.d.



(b) Fig. 3 (d) from paper with s.d.



(c) Fig. 3 (e) from paper with s.d.



(d) Fig. 3 (f) from paper with s.d.

Figure 1: Plots from the paper that include the evaluation reward standard deviation for different curricula: (a): Goal-only; (b) Environment, then goal; (c) Goal, then environment; (d) Task graph. Each line is a particular task in the task graph NxLy where x indicates the number of colors and y indicates the sequence length. For each task, a dashed line shows the results of training from scratch (without transfer) while a solid line indicates continuing the curriculum from top of the legend down.

gather – that have competing needs. We also attempted several iterations of the observation space in order to facilitate learning.

Section 2.1 discusses the initial results for a 20x20 grid and shorter snake lengths. In that work we used a sophisticated reward structure that encouraged staying organized and collecting apples. As mentioned in the paper, we noticed that transfer was not very strong for shorter snake lengths in such a large grid, which led us to consider smaller grids and a simpler reward structure. Section 2.2 details results for a 10x10 grid with a basic reward structure, while Section 2.3 examines the impact of this problem variant in a transfer situation.

2.1 Initial results for transfer with Snake in a 20x20 grid

Our first attempt to learn in Snake involved learning a single, shared policy for `organized(body)` and `collected(apple)` and using a sophisticated reward structure to capture the two skills. We use a hybrid reward structure during training. The hybrid reward structure promotes both obtaining apples, and keeping the snake body organized. The overall reward is calculated at each step. To promote obtaining apples, a large reward of 10 is given for every apple collected, and a reward of 1 is given at every step if the snake head moves closer to the apple. To encourage the snake to keep alive, a penalty of -1000 for every time the game ends due to snake collision with the exterior walls or the snake body. To promote snake body organization, wall-following and zig-zagging is rewarded. To prioritize wall-following, a reward of 1 is given if the snake head is closer to the exterior walls at each step. To prioritize zig-zagging, a reward of 1 is given if the snake head takes a step adjacent to the snake body.

The observation space for the snake is 16 states updated at each step. It includes the apple location relative to the snake head (above, below, right, or left), if there is a snake body obstacle to the snake head (above, below, right, or left), if there is a wall obstacle to the snake head (above, below, right, or left), and what direction (up, down, right, or left) the snake moved at the last step. Additionally, the action space for the snake is four options representing which one direction (up, down, right, or left) the snake head will move at the step next.

Our hypothesis for this study is that this form of implicit task transfer learning with our proposed curriculum will result in better performance over learning the most difficult task directly, but this ended up not being the case. We evaluate the trained policies by measuring the score (i.e., the number of apples collected before terminating) for 1000 rollouts. We also compare the training curves to examine recovery time under transfer. We don't increase the length after consuming apples (in contrast to the classic game) because we want to study whether policies for shorter lengths transfer to longer lengths. We used the Stable-Baselines3[1] Python package on a server with a 1.2 GHz 8-core Intel-based processor. We trained the policies using the PPO algorithm [1] with a standard multilayer perceptron policy that saved at every 10,000 steps.

Figure 2 shows boxplots of the evaluation scores for a length of 10 (top) and 15 (bottom) on a grid size of 20x20. The line inside the box indicates the median score and the triangle indicates the average score; the bottom and top box lines indicate the 25th and 75th quantiles.

The left column (blue box) of each plot shows the baseline result of learning without task curriculum. The starting task descriptor for both plots is (20, 5)|(1, 1). For the left plot, the policy 10_T for the task t_{10} (descriptor $= (20, 10)|(1, 1)$) was transferred from the policy learned for a length of 5. The results show that the mean score increases from 155 to 213 (using a t-test this is a significant difference; $p \ll 0.0001$) indicating that transfer was effective from length 5 to length 10.

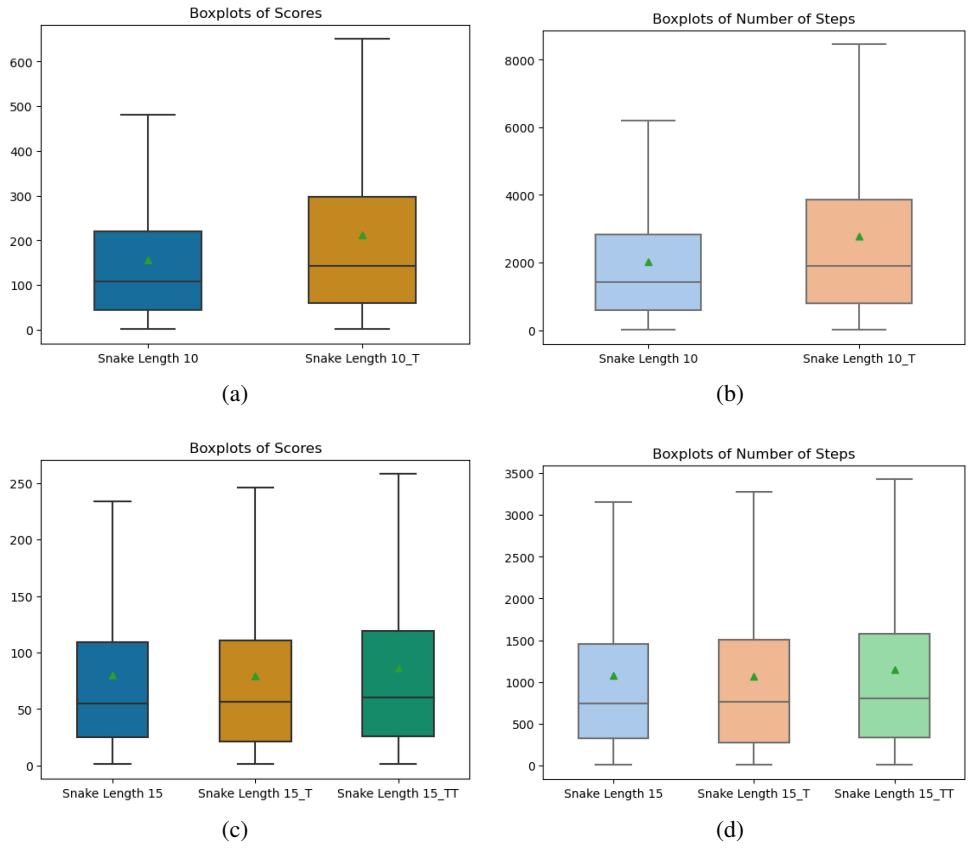


Figure 2: Comparison of evaluation scores (number of apples collected) and number of steps, with and without task curriculum in the Snake environment for two target tasks: (20, 10)|(1, 1) (top) and (20, 15)|(1, 1) (bottom). Higher scores indicate better performance.

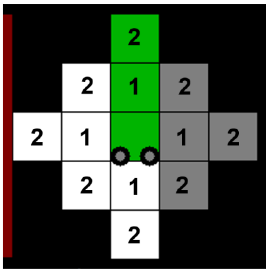


Figure 3: The Snake observation space for the studies in a 10x10 grid. Numbers on the grid represent the Manhattan distance from the head. Green squares contain the snake with the head pointing facing the eyes, gray squares contain the grid border, and the red section of the border indicates the apple direction.

For the task t_{15} (descriptor $= (20, 15)|(1, 1)$), we examined two cases: (a) policy 15_T is transferred from a vanilla policy that learned t_{10} directly, and (b) policy 15_TT transferred from policy 10_T. The results show that the mean score remains the same from a baseline of 80.0 to 80.0 for the 15_T result ($p \approx 0.88$) and to 86.4 for the 15_TT result ($p \approx 0.08$). Although the result is less significant for this test, the trend is improving. One explanation for the low improvement is that higher lengths require a different emphasis between goals, which we explored in the paper and detail below in the experiments of Section 2.3.

We also considered the recovery time of the transfer learning, which is the difference in the number of iterations it takes to return to an equal or better evaluation after transfer learning begins. For the 10 and 10_T policies, we observed that transfer results in a nearly immediate boost after recovering, reaching a new high score of 8000 within a million steps, whereas the typical score was around 6000 for the baseline learning. Note that our training scores are an order of magnitude higher than our evaluation scores because during training we let the episode continue for much longer. With the 15_T policy, a typical score of 2500 is achieved within two million steps, whereas the baseline takes about three million steps to arrive at that point. Based on these results, we can say that the recovery time is definitely improved with transfer.

In summary, though recovery seems to do well, transfer is not as strong as we had hoped it would be for this particular setup. So we focused our attention to a smaller version of the problem, a 10x10 grid, as well as began to understand more carefully the two aspects of the snake behavior we discussed in the paper.

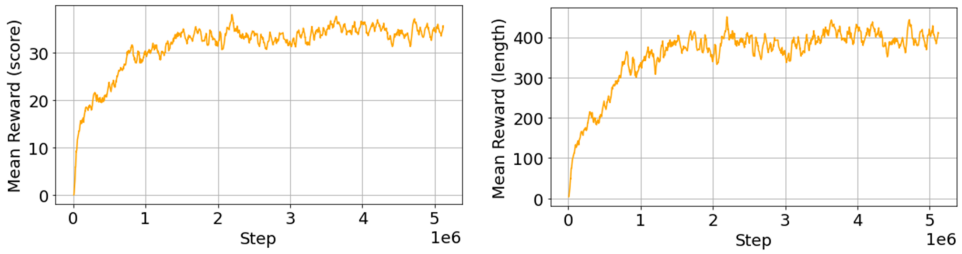
2.2 Learning Snake on the collect apple reward alone for a 10x10 grid

We also tried to learn snake using a simple reward of getting 1 for each apple collected. In this study, the snake grows by a length of 1 each time the apple is collected. The observation space of these results is more direct as well, as shown in Figure 3. For these experiments, we trained PPO on a Windows Linux subsystem OS with 4 cores at 3GHz and 64GB RAM. The parameters for PPO included the MLP policy type, a learning rate of 0.0003, and a discount factor of γ = 0.99 and λ = 0.95. Also, the algorithm collects 2048 steps of state-action-reward experience before updating network parameters, with 10 epochs of batches of size 64 per update.

Figure 4 (top) shows that PPO only manages to achieve about 35 apples eaten, which is not very high. Similarly, the bottom figure shows that episode length tends to hover around 375. While the episode length is longer than some values we saw in the paper, apples are not being collected past 35, suggesting that the snake is sacrificing score for staying alive.

2.3 Evaluation of transfer for collect and organize in a 10x10 grid

In the paper, we showed the results of performing transfer for different snake lengths (cf. Figure 4 in the paper). However, that work focused on the results during training. We wondered how well the trained policy would perform during evaluation. To accomplish this, stopped training every periodically, generated 10 random environments for the current task, and ran 100 trials of each environment, resulting in 1000 runs. For collect, we capture the number of apples collected, while for

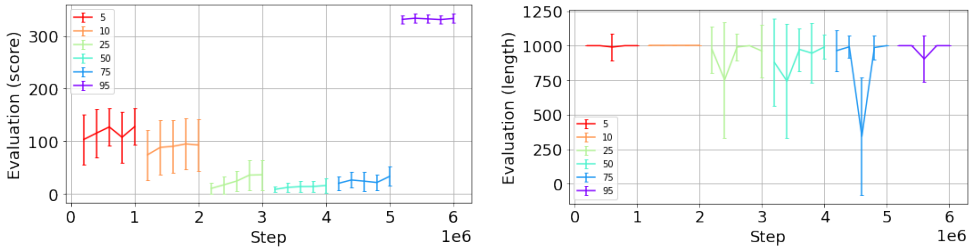


(a) (b)

Figure 4: Learning Snake using PPO on the score alone. The snake grows each time the apple is collected. The left plot shows the number of apples collected with a score of 1 for each apple, while the right plot shows the episode length with a score of 1 for each step.

the organize we capture the number of steps taken. However, we stopped the evaluation at 1000 steps, which is 10 times the maximum size of the 10x10 grid.

Figure 5(a) shows the result of testing the collect skill. Recall that the baseline PPO system collected about 35 apples and had an average episode length of around 375 (shown in Figure 4). As can be seen, all skill variants perform in episode length, often reaching the 1000 step limit. The collect skill does very well for lengths of 5, 10, and 95 and performs reasonably well for other lengths. The problem is more challenging for lengths 25, 50, and 75, but it still performs well. Figure 5(b) shows a similar result for the collect skill when the training is stopped every 200K steps.



(a) (b)

Figure 5: Evaluating (a) collect and (b) organize transfer for the Snake environment.

3 Further Details of the Key Corridor Environment

In this section, we present an example task graph for the KeyCorridor Environment and describe our algorithms for automated curriculum generation and plan execution in detail.

3.1 An example task graph for the KeyCorridor

Figure 6 shows another example task graph based on the KeyCorridor environment. The actor's (red triangle) goal is to deliver an item (orange ball) to the destination (the upper left green box), stated as `at(item, dest)`. This goal has a set of subgoal achievements, namely row 3, `holding(item)`, row 2, carrying it to destination (i.e., `holding(item)` $\wedge$ `at(dest)`), and row 1, placing the object (`at(item, dest)`).

Example Environment Descriptor For Figure 6, ϕ consists of the number of items in $\{0, 1\}$ and the number of rooms in $\{0, 1\}$. Here, k_i is 1 for maximum number of items and rooms, though it is easy to see how the max could be increased for this example. Together, these features result in $\boldsymbol{env} = (room, item)$.

Example Goal Descriptor The rows of Figure 6 vary the goal complexity, matching the layout of Figure 1 in the main paper. In the column containing the environment descriptor $(0, 1)$ there are three

goals to complete in order: g_1 : `holding(item)`, g_2 : `at(dest)`, and g_3 : `at(item,dest)`. The column containing the environment $(1, 1)$ prepends an additional goal of g_0 : `inside(room)`.

The actor achieves the goal `at(dest)` when its position equals the location parameter.

Example Task Descriptors For Figure 6, there are seven tasks, where the top left task would be $\boldsymbol{m}^{(0,1,1,1)}_{(0,1)}$ or $(0, 1)|(0, 1, 1, 1)$.

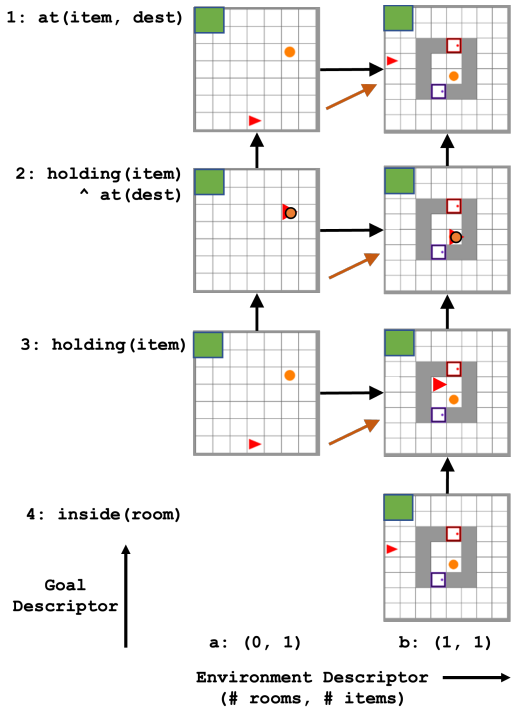


Figure 6: Some possible initial states of an example Task Graph for a series of subgoals (goal descriptors changing vertically) of a plan that results in an actor (red triangle) delivering an item (orange ball) to a destination (upper left green corner). The environments in the left column have no rooms while the ones on the right have one room. As a result, the right environments need a new subgoal to be inside the room first, `inside(room)`. The orange arrows show potential for task transfer within a specific subgoal.

3.2 Automated Task-Graph Curriculum Generation

Algorithm 1 shows the pseudocode for our task curriculum generation and goal skill training algorithm. We want to learn starting with easier tasks and slowly move to harder tasks. Let $\boldsymbol{u}(env_u|g_u)$ be the easiest task to learn and $\boldsymbol{f}(env_f|g_f)$ the hardest. Line 1 finds the shortest path from $\boldsymbol{u}$ to $\boldsymbol{f}$ using greedy best-first search (GBFS) on the task graph, M. The heuristic used to choose among all neighbors at equal distance from a node is the neighbor which is nearest to $\boldsymbol{f}$. Line 3 walks the $path$ and trains all the goal skills with positive values in the goal descriptor g of $\boldsymbol{m}$.

To train the goal skills for $\boldsymbol{m}$, TrainWithTaskCurriculum uses the sub-routine, `TrainGoalSkills`. It first selects an initial state from the environment and then uses a planner, `GTPyhop`[3], to recursively decompose goals using methods are both relevant and applicable. `GTPyhop` generates $\langle a^1, a^2, \ldots, a^n \rangle$, which is an a ordered sequence of actions. Every action has a corresponding goal skill, so this sequence of actions corresponds to a totally-ordered sequence of goal skills, *plan* $= \langle g^1_\pi, g^2_\pi, \ldots, g^n_\pi \rangle$. This *plan* guides the learning of goal skills in a particular order that is biased by the ordering of the methods that were used to create the $plan$. In future work we will extend the curriculum to include

[3] `GTPyhop` [3] extends the Pyhop HTN planner [4, 5] with the ability to do goal decomposition in a manner similar to GDP [6]; the source is available at https://github.com/dananau/GTPyhop

Algorithm 1: TrainWithTaskCurriculum

Input: $\boldsymbol{u}(env_u|g_u)$ - the starting task descriptor (easiest task),
$\boldsymbol{f}(env_f|g_f)$ - the final task descriptor (hardest task)
M- the task graph
Result: Π: A set of trained goal skills from $\boldsymbol{u}$ to $\boldsymbol{f}$

1 $path \leftarrow$ Shortest path in M from $\boldsymbol{u}$ to $\boldsymbol{f}$ using greedy BFS
2 $\Pi \leftarrow$ A set of empty policies $\check{\pi}$ for each goal skill $\in g_u \cup g_f$
3 **for** *each task descriptor* $\boldsymbol{m}(env|g)$ *in path* **do**
4 $\quad \Pi \leftarrow$`TrainGoalSkills(`$\boldsymbol{m}(env|g)$, Π`)`
5 **return** Π
6 **Function** `TrainGoalSkills(`$\boldsymbol{m}(env|g)$, Π`):`
7 $\quad s_{init} \leftarrow env.s_{init}$
8 $\quad$ *plan* $\leftarrow$ `GTPyhop` (g, s_{init})
9 $\quad$ **for** *goal skill* $g_\pi \in$ *plan* **do**
10 $\quad\quad g_\pi$*.initiation_set* := $\hat{S}_{\text{TERM}}(pred(g_\pi))$
11 $\quad\quad \check{\pi} \leftarrow \Pi[g_\pi]$
12 $\quad\quad$ **while** *not* converged($\check{\pi}$) **do**
13 $\quad\quad\quad$ *exp* $\leftarrow$ CollectExperience(env, g_π)
14 $\quad\quad\quad$ train($\check{\pi}$, *exp*)
15 $\quad\quad\quad$ **if** *env.done* **then**
16 $\quad\quad\quad\quad \hat{S}_{\text{TERM}}(g_\pi)$.add(*env.state*)
17 $\quad\quad\quad\quad$ *env*.reset(Sample(g_π*.initiation_set*))
18 $\quad$ **return** Π *# trained policies for goal skills*

partial orders and more sophisticated plans. Each goal skill in the *plan* is trained sequentially (Line 9). Before training, the set of initiation states is copied from the last goal skill's termination set (Line 10).

A critical step for task transfer happens in Line 11. Instead of initializing an empty policy, `TrainGoalSkills` looks up the current set of trained policies Π and chooses the existing policy for goal skill g_π to begin training. This results in the knowledge learned in previous calls to `TrainGoalSkills` to be transferred to the current training step. Until the algorithm converges (Line 12), training proceeds as a standard actor-critic architecture by collecting the experiences storing them in a replay buffer (Line 13) and updating the policy for each iteration (Line 14). If the episode is finished (Line 15), the current terminal state is saved into the termination set (Line 16), which will be used for the next goal skill at Line 10, and the environment is reset to a new state sampled from the initiation set of g_π (Line 17). Training of a g_π ends when converged(g_π) returns *True*, which is determined by the performance of the policy during training.

`TrainGoalSkills` stores the states that meet the $\mathcal{C}_{\text{start}}$ and $\mathcal{C}_{\text{end}}$ conditions of each goal skill. These sets of starting and ending points sample the initiation (INIT) and termination conditions (TERM) for training the policies and ensure a natural starting point for training each successive goal skill.

For a *plan*, let the termination set of a goal skill be defined as $S_{\text{TERM}}(g_\pi) := \{s \in S | \mathcal{C}_{\text{end}}^{g_\pi}(s)\}$ and let $\hat{S}_{\text{TERM}}(g_\pi)$ be a sampled set from $S_{\text{TERM}}(g_\pi)$. In a plan, a goal skill may occur multiple times each with distinct predecessor goal skills. Let $pred(g_\pi) = \{g'_\pi \in \mathcal{G}_\Pi | \textit{name}(g'_\pi) = \textit{name}(g_\pi)\}$ be the collected set of predecessors for all goal skills with the same name. Before training each goal skill, the algorithm queries the goal predecessors from the plan and concatenates the termination sets of the predecessors to build the initiation set of the current goal skill.

We provide a distinct, intrinsic reward structure for each goal skill, in contrast to the environment providing the reward after each action. TrainWithTaskCurriculum generates a fixed reward for each goal skill based on whether the conditions $\mathcal{C}$ have been satisfied. The rewards are goal-oriented and application dependent, so we describe for each environment how the reward structure was created for its goal skills. This kind of intrinsic reward sets the stage for allowing the actor to direct its own learning depending upon it's current goals.

3.2.1 Plan Execution With Goal Skills

In order to accomplish an incoming stream of tasks, t_m (with task descriptors $\boldsymbol{m}(env|goal)$), the actor uses the hierarchical planner, `GTPyhop` and a sub-set of the trained set of goal skills, Π (trained

Algorithm 2: ActingWithGoalSkills

Input: Π - the set of learned policies for goal skills
Result: accomplished task, or failure.

1 **begin**
2 **foreach** new task t_m **do**
3 $\boldsymbol{m}(env|goal) \leftarrow$ task descriptor of t_m
4 $s \leftarrow$ Observe env state
5 *plan* $\leftarrow$ `GTPyhop` $(goal, s)$
6 *plan.active_goal_skill* $\leftarrow$ *plan*.first()
7 **while** *plan.active_goal_skill* is not *None* **do**
8 $s \leftarrow$ Observe env state
9 *action* $\leftarrow$ `ProgressSkill` $(s, goal, \Pi,$ *plan*$)$
10 **if** *action* is *failure* **then**
11 `output` (*failure* for task t_m); Move to next task
12 **else** Send *action* to the execution platform
13 **Function** `ProgressSkill`$(s, goal, \Pi,$ *plan*`):`
  Result: *action* suggested by learned skill or failure
14 **begin**
15 **if** *plan.active_goal_skill* is successful **then**
16 *plan.active_goal_skill* $\leftarrow$ *plan.next()*
17 **else if** *plan.active_goal_skill* has failed **then**
18 *plan* $\leftarrow$ `GTPyhop` (g, s)
19 **if** *plan* = *failure* **return** *failure*
20 **else** *plan.active_goal_skill* $\leftarrow$ *plan*.first()
21 **return** Π[*plan.active_goal_skill*](s)

using the task curriculum in Algorithm 1). The acting engine queries `GTPyhop` for a *plan*, i.e., a correct sequence of goal skills to accomplish $goal$ in the current state of environment with descriptors env. Algorithm 2 details the acting engine, ActingWithGoalSkills. The algorithm receives as input the set of trained goal skills, Π. It reads an incoming stream of tasks in a loop. For each new task, it obtains a *plan* from `GTPyhop` with the current state (Line 5). Then, it executes the plan, starting with the first goal skill (Line 6). For executing the goal skill, it queries the trained policy for the correct action given the current state using the procedure `ProgressSkill` (Line 9). It moves on to the next goal skill in the plan, if the current one succeeds. However, if the current goal skill fails by violating any of its conditions $\mathcal{C}$, `ProgressSkill` replans by calling `GTPyhop` again to create a new *plan* for recovery. Otherwise, the returned action is sent to the execution platform (Line 12).

3.3 Details of Training Goal Skills in KeyCorridor Environment

We ran our training and evaluation on a 3.2 GHz 10-core ARM-based processor with 16 GB RAM. We train each of the ten goal skills up to a total of 3 million time steps for all the six variants of curricula. We use the the RL starter files[4] to train each individual goal skill. The environment is built using minigrid[5]. The training curves for the five goal skills in Scenario 1 is shown in Figure 7, and the training curves for the ten goal skills in Scenario 2 is shown in Figure 8. During training, we continuously check a convergence condition to decide on continuing or moving to the next step in the training curriculum. Our convergence condition is that in five consecutive training batches, (a) L2 Norm of the gradients is less than $t_1 = 0.05$, so that model parameters are not changing drastically, and (b) t_2 (= all) episodes in the training batch must return success, to make sure the actor is doing well in training. These hyperparameters t_1 and t_2 are not completely optimized for this study but can be tuned to improve performance or efficiency. In several places along the training curve for a goal skill, the reward can drop suddenly then recover, which happens when training for a particular curriculum step is complete and the next curriculum step begins.

[4] https://github.com/lcswillems/rl-starter-files
[5] https://minigrid.farama.org/

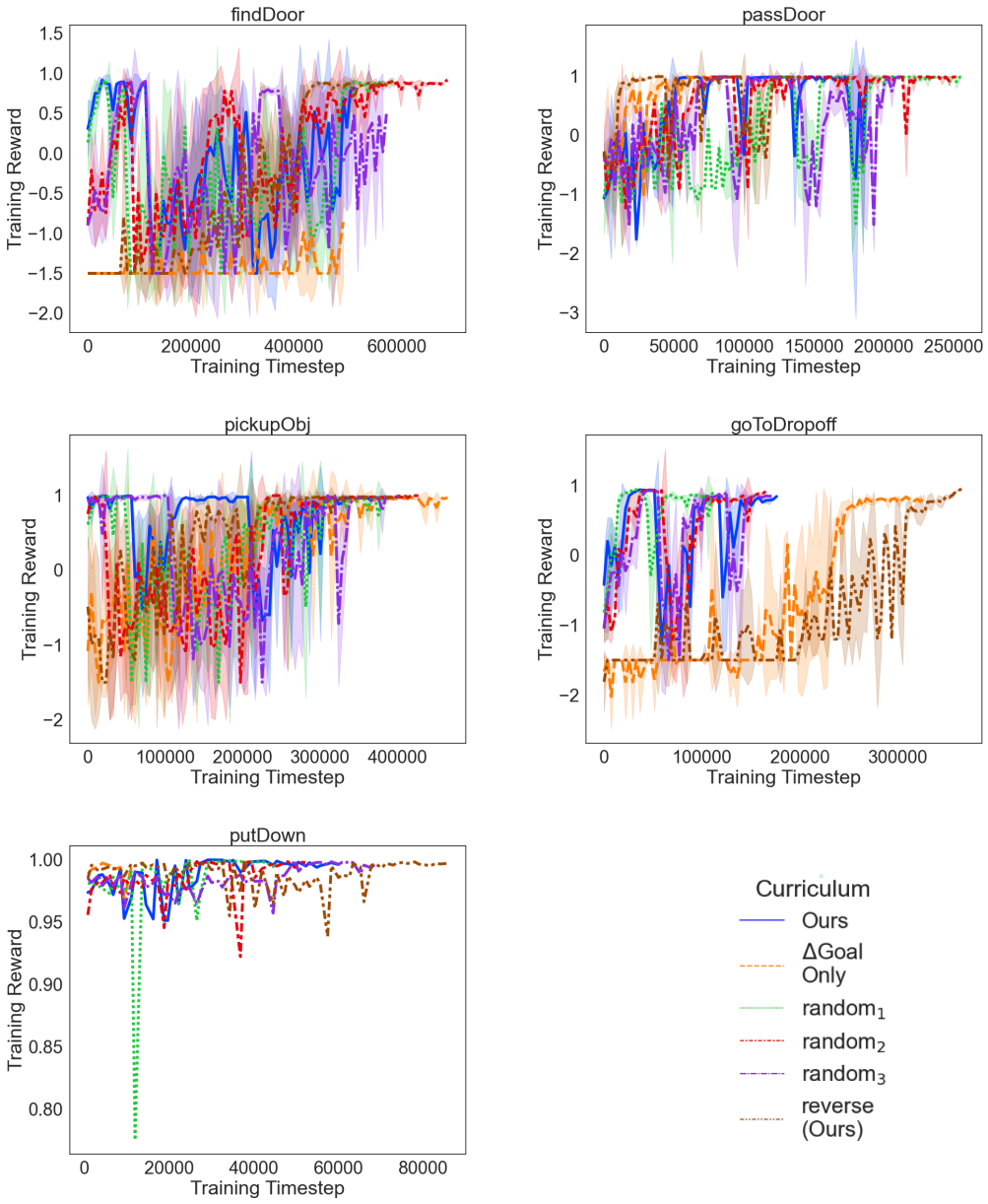


Figure 7: Comparison of training five goal skills in Scenario 1 (see main paper) of the KeyCorridor with our task curriculum vs ΔGoal, random, and reverse curricula.

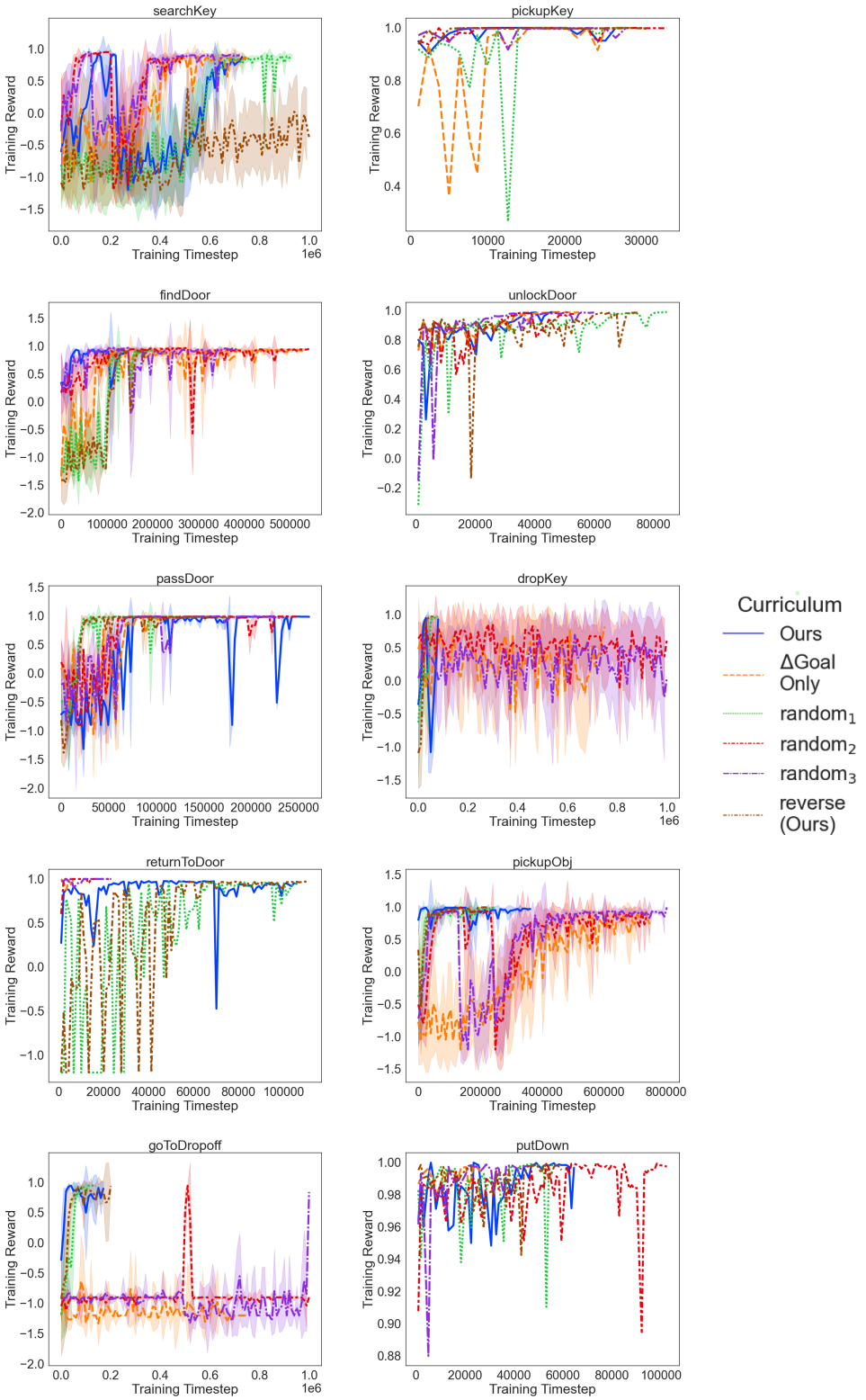


Figure 8: Comparison of training ten goal skills in Scenario 2 (see main paper) of the Key Corridor domain with our task curriculum vs ΔGoal, random, and reverse curricula.

References

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[2] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[3] Dana Nau, Sunandita Patra, Mak Roberts, Yash Bansod, and Ruoxi Li. GTPyhop: A hierarchical goal+task planner implemented in Python. In *ICAPS Wksp. on Hierarchical Planning (HPlan)*, 2021.

[4] Dana Nau. Game applications of HTN planning with state variables. In *ICAPS Workshop on Planning in Games*, 2013. Keynote talk.

[5] Dana S. Nau. Pyhop, version 1.2.2: A simple HTN planning system written in python. Software release, 2013.
`https://bitbucket.org/dananau/pyhop/src/master/`.

[6] Vikas Shivashankar, Ugur Kuter, Dana S. Nau, and Ron Alford. A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proc. AAMAS*, pages 981–988, 2012.