# A Hierarchical Goal-Biased Curriculum for Training Reinforcement Learning

**Sunandita Patra**[1,*], **Mark Cavolowsky**[1], **Onur Kulaksizoglu**[1], **Ruoxi Li**[1], **Laura M. 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
*now at JP Morgan AI Research, New York, USA
[1]{patras, markcav, kulaksor, rli12314, nau}@umd.edu [2]{first.last}@nrl.navy.mil

**Abstract**

Hierarchy and curricula are two techniques commonly used to improve training for Reinforcement Learning (RL) agents. Yet few works have examined how to leverage hierarchical planning to generate a curriculum for training RL Options. We formalize a *goal skill* that extends an RL Option with state-based conditions that must hold during training and execution. We then define a *Goal-Skill Network* that integrates a Hierarchical Goal Network, a variant of hierarchical planning, with goal skills as the leaves of the network. An automatically generated plan for a *Goal-Skill Network* correctly orders goal skills such that (1) it is a *Goal-Biased Curriculum* for training the goal skills, and (2) it can be executed to achieve top-level goals. In a set of six distinct gridworld environments using up to ten goal skills, we demonstrate that these contributions train nearly perfect policies significantly faster than learning a whole policy from scratch.

## 1 Introduction

We study the problem of how to guide the training of Reinforcement Learned skills using a variant of Hierarchical Goal Networks. Using goals to focus learning has been studied in a variety of contexts (Ram and Leake 1995; Hawes 2011; Aha 2018). Recent work has shown that one of the benefits of these approaches is that they allow planning to guide the learning so that it is more focused even when reward signals are sparse (Illanes et al. 2020), a situation in which Reinforcement Learning (RL) alone can struggle. Yet no work we are aware of has examined how to link recent hierarchical planning approaches with RL policies.

Hierarchical Goal Networks (HGNs) (Shivashankar 2015) leverage the best of standard planning techniques (e.g., landmark heuristics, heuristic search) and Hierarchical Task Networks. An HGN consists of a set of partially ordered goal nodes that can be decomposed into subgoals or actions via *methods*. Similar to Hierarchical Task Networks (HTNs), the methods provide a substantial speed up for planning; in studies, speedups of 2-8 times were common (Shivashankar et al. 2013). However, HGNs can operate without decomposition methods and, in the absence of such methods, HGNs can simply "fall back" to calling a classical planner. Further, because the networks consist of goals (i.e., state descriptors rather than just task labels), HGNs can leverage advances in classical planning heuristics (Shivashankar et al. 2016).

In this paper, we synthesize a variant of the Hierarchical Goal Network with the approach of Illanes et al. (2020). The contributions of the paper include: (1) formalizing a *goal skill* that generalizes the RL Option (Sutton, Precup, and Singh 1999) and links information about a goal from the goal network with an RL policy, (2) defining a Hierarchical *Goal-Skill Network* (GSN) that decomposes top-level goals into subgoals and goal skills, (3) developing a *Goal-Biased Curriculum* that leverages a hierarchical planning system called `GTPyhop`, provides an intrinsic reward structure, and trains the goal skills using the GSN, (4) showing in six gridworld environments that the Goal-Biased Curriculum trains goal skills to near perfection significantly faster than it takes to learn a poorly-performing monolithic policy without the GSN, and (5) demonstrating an integrated actor that plans and executes a trained GSN is able to achieve goals. Our study shows that combining HGNs with RL policy learning leverages the strengths of each approach: (1) providing HGNs with the robust acting that RL policies can provide and (2) effectively training these RL policies in difficult or reward-sparse environments. Though the results are limited to gridworld domains, we believe they show promise and our future work will generalize this approach to other domains.

## 2 The Goal-Biased Curriculum

Our ultimate aim is to integrate a policy into a goal network that we can then use to develop a training curriculum. We accomplish this by: extending the RL Options framework into goal skills (defined in Section 2.1), integrating goal skills into a Goal Skill Network that extends Hierarchical Goal Networks (defined in Section 2.2), and using the Goal Skill Network for the Goal-Biased Curriculum (defined in Section 3). Background material is integrated with the discussion.

**Running Example.** Our study uses the Minimalistic Gridworld (MiniGrid) simulator (Chevalier-Boisvert et al., 2018). Figure 1 (left) shows our variant of the Keycorridor Task from Minigrid, where an agent (red triangle) must move an item (yellow) to the top-left corner, with six variants differing in whether the agent must simply find the object, as in Env 1 and 4, pass through unlocked doors, as in Env 2 and 5, or unlock a door by using a key, as in Env 3 and 6.

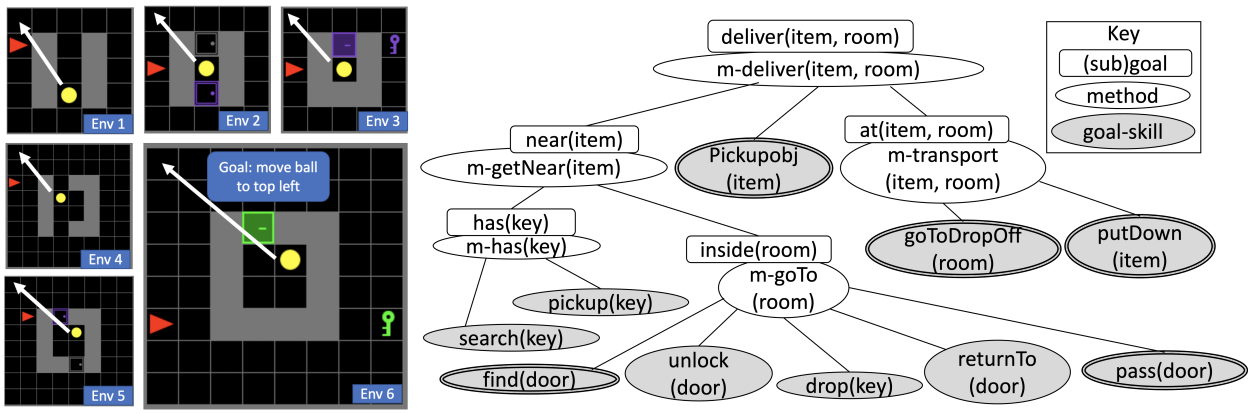


Figure 1: Left: The six Minigrid environments (Env) we evaluate. Right: The Goal Skill Network we use in the study. The double bordered goals skills are used in the curricula for Env 1, 2, 4, 5. All goal skills are used for Env 3 and 6.

**Background: RL Options.** We consider the problem of learning a set of policies for achieving subgoals for a Markov Decision Process (MDP). An MDP is a tuple MDP $= (S, A, T, R, \gamma)$, where $S$ is the set of states, $A$ is the set of actions, $T : S \times A \rightarrow S$ is the transition function, $R$ is the reward function, and $\gamma$ is the discount rate. A solution to the MDP is a policy $\pi : S \rightarrow A$ that maps each state to an action.

An option (Sutton, Precup, and Singh 1999) over an MDP is a tuple $o = (\pi, \text{INIT}, \text{TERM})$, where the policy $\pi$ can begin executing when the INIT $\subset S$ is valid and terminates its execution when TERM $\subset S$ is true. An option is a way to define multiple policies for an MDP that can span more than a single action. They are often called temporally extended actions, macro actions, or skills.

## 2.1 Goal Skills

We define goals as a symbolic representation of state or states the actor can achieve. Figure 1 (right) shows several kinds of goals the agent may need such as having a key or being in a room. We will describe the specific representation we use below. To link a policy to a goal, we define a goal skill:

**Definition 1.** A *goal skill* $g_\pi = (head, \breve{g}, \breve{\pi}, \mathcal{C})$ is a tuple where *head* includes the *name* and *parameters* of the goal skill, $\breve{g}$ contains the goal information the skill is meant to achieve, $\breve{\pi}$ is a *micro-policy* for this goal skill, and $\mathcal{C}$ is a non-empty set of conditions that should be satisfied while executing $\breve{\pi}$.

For example, a goal skill to unlock a door would be:

head: unlock(door)
$\breve{g}$: unlocked(door)
$\mathcal{C}$: start+during:have(key), end:unlocked(door)
$\breve{\pi}$: unlock-door

Similar to an option, learning $\breve{\pi}$ still occurs in the full state and action space of the MDP. However, a goal-specific reward is used instead of $R$. Including the name and parameters allows more than instantiation of a goal skill to exist in the same goal network. Goal parameters allow a goal skill to be used over a variety of similar objects.

We partition $\mathcal{C}$ into $\{\mathcal{C}_{\text{start}}, \mathcal{C}_{\text{during}}, \mathcal{C}_{\text{end}}\}$. $\mathcal{C}_{\text{start}}$ declares a function over states that declares whether the skill can start; it aligns with an option's INIT. $\mathcal{C}_{\text{end}}$ defines a function over states that designate the desire outcome of the goal skill; it aligns with an option's TERM set but for the end of the duration. $\mathcal{C}_{\text{during}}$ declares the invariants of skill execution; it also aligns with an option's TERM but for specific invariants that, when violated, should cause failure. Together, these conditions provide clarity about when to start, continue, or terminate a goal skill. Those familiar with temporal planning will recognize $\mathcal{C}$ as similar to the conditions of a temporal action, which define constraints over the feasible trajectories of an action during execution. Our study in Section 4 uses a small set of $\mathcal{C}$, and extending the implementation to include richer conditions is an area for planned future work.

## 2.2 The Goal Skill Network (GSN)

To generate hierarchical plans, we adapt the Hierarchical Goal Network (HGN) by Shivashankar et al. (2013), which allows for a direct correspondence between the states of the MDP and the goals to be decomposed[1].

**Background: A Hierarchical Goal Network** is a pair $hgn = (G, \prec)$, where $G$ is a set of nodes representing goals, each $g \in G$ is a goal a predicate statement of ground literals in disjunctive normal form, and $\prec$ is a partial order over $G$. Nodes in the network are distinguished by whether they are compound or primitive. Compound nodes are not directly executable and require further decomposition. Primitive nodes (i.e., actions) describe what needs to be accomplished and can be executed by a controller.

Methods allow a domain designer to write domain-specific decompositions for goals. A *method* $m$ decomposes a non-primitive goal $g$ by inserting its subgoals into *hgn* and ordering them to occur before $g$. A method is a tuple $(head, g, pre, m_{hgn})$, where *head* is the method's name and parameter list, $g$ is the goal that $m$ is relevant for achieving, *pre* is the preconditions that determine whether $m$ is applicable in the current state, and $m_{hgn}$ is the goal network to

[1]The Hierarchical Task Network (HTN) formalism would be much more complicated because HTN methods are not required to refer to actual states of the world, only actions must do that.

be used to achieve $g$. Shivashankar et al. (2013) showed that methods substantially improve search.

**A Goal Skill Network** links a goal network with skills. Let $G$ be a set of goal nodes where a goal is described in state-variable representation from (Ghallab, Nau, and Traverso 2016). Further, let $\check{\mathrm{G}} = G \cup \mathcal{G}_\Pi$ be a set of goal nodes that is a union of the $G$ and the set of goal skills $\mathcal{G}_\Pi$.

**Definition 2.** A *Goal Skill Network* $gsn = (\check{\mathrm{G}}, \prec)$ is a tuple where $\check{\mathrm{G}} = G \cup \mathcal{G}_\Pi$ is a set of goal nodes and goal skills, and $\prec$ describes a partial order over $\check{\mathrm{G}}$.

We will refer to this class of Goal-Skill Networks as GSNs and specific network instances as a $gsn$.

Similar to the Hierarchical Goal Network, methods decompose nodes in $G$. However, instead of reaching primitive goals that can unify with actions, the GSN decomposes primitive goals into goal skills, as shown in Figure 1 (right). The main benefit of the GSN is the goal ordering it provides. The GSN ordering is used to provide a curriculum for learning goal skills, and it is used as a plan during execution.

### 2.3 The Goal-Biased Curriculum

The Goal Biased Curriculum uses a *plan* produced by `GTPyhop`[2] (Nau et al. 2021), which extends the Pyhop HTN planner (Nau 2013a; 2013b) with the ability to do goal decomposition in a manner similar to GDP (Shivashankar et al. 2012). `GTPyhop`'s planning algorithm recursively decomposes a goal or task using whichever methods are both relevant and applicable.

`GTPyhop` generates an ordered sequence of actions $\langle a^1, a^2, \ldots, a^n \rangle$. This sequence of actions corresponds to a sequence of goal skills, $plan = \langle g_\pi^1, g_\pi^2, \ldots, g_\pi^n \rangle$. This *plan* guides the learning of goal skills and the acting algorithm. Each goal skill has a corresponding policy that is learned using an offline procedure called `TrainGSN`. The actor executes the learned policies via its execution platform.

## 3 Using the Goal-Biased Curriculum

This curriculum for training the goal skills is biased by the ordering of the methods that were used to refine it. For this study, the plan is a totally-ordered sequence of goal skills. In future work we will extend the curriculum to include partial orders and more sophisticated plans. The curriculum also includes the training procedure and the reward structure.

The `TrainGSN` algorithm, shown in Algorithm 1, uses the *plan* to train goal skills. Lines 1-3 initialize data structures and obtain the *plan*. Each goal skill in the *plan* is trained sequentially (Line 4). Before training, the set of initiation states is copied from the last goal skill's termination set (Line 5). Until the algorithm converges (Line 6), training proceeds as a standard actor-critic architecture by collecting the experiences storing them in a replay buffer (Line 7) and updating the policy for each iteration (Line 8). If the episode is finished (Line 9), the current terminal state is saved into the termination set (Line 10), which will be used for the next goal skill at Line 5, and the environment is reset to a new state sampled from the initiation set of $g_\pi$ (Line 11). Training

[2]https://github.com/dananau/GTPyhop

**Algorithm 1:** `TrainGSN`

**Input:** $env$ - the environment, $gsn$ - GSN to be trained.

1 **begin**
2 $gsn' \leftarrow gsn$; $g \leftarrow gsn'$.root(); $s_{init} \leftarrow env.s_{init}$
3 *plan* $\leftarrow$ `GTPyhop` $(g, gsn', s_{init})$
4 **for** $g_\pi \in$ *plan* **do**
5 $g_\pi$.*initiation_set* := $\hat{S}_{\text{TERM}}(pred(g_\pi))$
6 **while** *not* converged($g_\pi$) **do**
7 *exp* $\leftarrow$ CollectExperience($env, g_\pi$)
8 train($g_\pi$, *exp*)
9 **if** *env.done* **then**
10 $\hat{S}_{\text{TERM}}(g_\pi)$.add(*env.state*)
11 *env*.reset(Sample($g_\pi$.*initiation_set*))
12 **return** $gsn'$ *# trained GSN*

of a $g_\pi$ ends when converged($g_\pi$) returns *True*, which is determined by the performance of the policy during training.

An important step of this process is when `TrainGSN` stores the states that meet the initiation and termination conditions of each goal skill. These sets of starting and ending points serve two purposes. First, they sample the initiation (INIT) and termination conditions (TERM) of the managed Options Framework. Second, they provide a natural starting point for each successive goal skill in the Goal-Biased Curriculum.

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 $pred(g_\pi) = \{g'_\pi \in \mathcal{G}_\Pi | name(g'_\pi) = name(g_\pi)\}$ 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.

Another important distinction of the Goal-Biased Curriculum is the reward structure. RL algorithms usually receive reward from the environment, but `TrainGSN` generates a standard reward for each goal skill based on the conditions $\mathcal{C}$. The reward can be application dependent, so we describe for each study how the reward structure was created for each goal skill. However, this kind of intrinsic reward sets the stage for allowing the actor to direct its own learning.

## 4 Experimental Evaluation

To demonstrate the ability to plan, learn, and execute in gridworld environments using goal skill networks, we implemented and tested our approach on six experimental domains built using the MiniGrid simulator (Chevalier-Boisvert, Willems, and Pal 2018).

**Domains.** For all the domains, the goal is to retrieve a ball from inside a room and deliver it to a pre-defined location (Figure 1, left). Our actor (red triangle) must deliver the yellow ball to the top-left corner. To accomplish this in Env 1 and 4, the actor must navigate to the ball, pick it up, move it to the top-left corner and deposit the ball. In Env 2 and 5, the actor can only reach the ball by passing a door. In Env 3

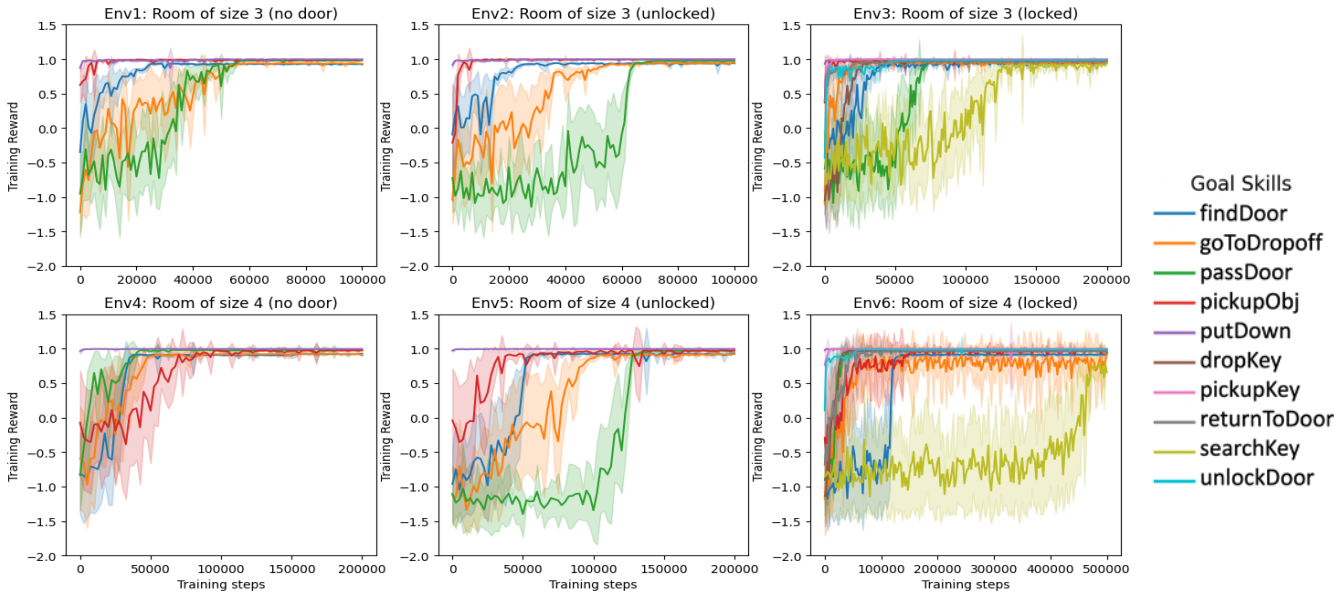


Figure 2: Learning Curves for up to ten goal skills in six environments. We learn the skills by following a goal-biased curriculum outlined in Algorithm 1.

and 6, the actor must search for the key and unlock the door before entering the room. Since the actor can carry only one object at a time, it must set the key aside before picking up a ball. The actor, key, ball and doors are placed in random locations. Env 4, 5, and 6 have larger grids than 1, 2 and 3.

The domains are fully-observable, with the actor able to observe the contents, properties, and status of each element in the grid. The action space is discrete, with 3 navigation actions (turn left, turn right, and move straight) and 3 interaction actions (pick up, put down, and use object).

## 4.1 Training Goal Skills

To solve our domains, we used the hierarchy in Figure 1 (right) with the goal skills: searchKey, pickupKey, findDoor, unlockDoor, dropKey, returnToDoor, passDoor, pickupObj, goToDropoff, and putDown. We trained each of Deliver's goal skills on a 2.3-GHz, 2-core Intel Core i5-8750H processor. We trained for a total of $1.5 \times 10^6$ time steps across all 10 goal skills. One timestep corresponds to one action taken by our actor. We reward the actor as follows: Goal Skill Achieved = +1, Goal Skill Failed = -1, Step Reward = -0.01.

Goal skill training ordering was determined through `TrainGSN`, the `GSN`, and `GTPyhop`. Figure 1 (right, all goal skills with a double border using in-order traversal) shows the curriculum for Env 1, 2, 4 and 5, which is findDoor, passDoor, pickupObj, goToDropoff, and putDown. Figure 1 (right, all goal skills using in-order traversal) shows the curriculum for Env 3 and 6, which is searchKey, pickupKey, findDoor, unlockDoor, dropKey, returnToDoor, passDoor, pickupObj, goToDropoff, and putDown.

The agent learns goal skills by approximating the optimal policy using a Deep RL network. Our implementation is based on the RL Starter Files (Willems 2021) and PyTorch (Paszke et al. 2017). The policy implementation is an Actor-Critic architecture (Sutton and Barto 2018), and the network is trained using PPO (Schulman et al. 2017). The Neural Network (NN) uses three convolutional layers to extract a latent representation of the environment. From this latent representation, we calculate the policy output and the value approximation in two separate fully-connected heads.

## 4.2 Results

Figure 2 shows the learning curves for our goal skills in all of our six domains. We observe that learning the goal skills, pickupKey, dropKey, unlockDoor, and putDown are relatively easy. This is expected because the solution consists of just one action if the preconditions are satisfied. TrainGSN is able to learn them within 50K training timesteps. The searchKey goal skill is the most difficult goal skill to learn for our actor, and it takes about 500K timesteps. We attribute this to searchKey being the goal skill with the highest variability and longest solution length. The findDoor, returnToDoor, passDoor, pickupObj, and goToDropoff skills are non-trivial and have relatively short solution lengths, which enables them to converge to maximum reward in less than 150K timesteps.

To demonstrate the efficacy of the trained policies, we tested the `TrainGSN` implementation against a single monolithic policy trained on the top-level deliver goal. After a total of $10^7$ timesteps, the policy generated a worst-possible rewards (in the range [-12, -9]) for all six environments, indicating that the monolithic policy was unable to learn an effective policy. This shows that `GTPyhop`-derived curriculum is an effective method for improving training convergence and sample efficiency.

We tested the trained goal skills achieve the higher-level goals in the `GSN`. This includes the top-level goal used to

train the GSN (deliver) and every compound goal of our domains: deliver, getNear, hasKey, goToRoom, and transport. The results are summarized in Table 1. For each goal, we ran our tests 25 times for statistical significance. We gave each test case a maximum of 500 timesteps to succeed, and the inability to succeed within this limit counts as failure.

The goal skills always succeed in achieving the desired state (i.e., its $\mathcal{C}_{\text{end}}$ condition) $100\%$ of the time. The Average Reward and Average Timesteps columns in Table 1 show the tradeoff of how many steps the agent must do and its maximum reward. This is because the step penalty reduces how well an agent could do. For example, goToDropOff skill completes in 20 steps, on average, which means a total penalty of $0.01 * 20 = 0.20$. So the best the agent could do is a reward of $0.80$, which is reflected in the average reward. Generally, the sum, Average Reward + step penalty * Average Timesteps provides a sense of how well the agent is learning. As can be seen, the sum is very close to 1, suggesting the agent is learning well.

| Goal | Env/ Domain | Plan Length | Test Reward $\bar{x}$ | Test Reward $\sigma$ | Timesteps $\bar{x}$ | Timesteps $\sigma$ |
|---|---|---|---|---|---|---|
| pickupKey | 3,6 | 1 | 0.99 | 0.01 | 1 | 1 |
| putDown | all | 1 | 0.99 | 0.01 | 1 | 1 |
| pickupObj | all | 1 | 0.99 | 0.01 | 2 | 1 |
| unlockDoor | 3,6 | 1 | 0.99 | 0.01 | 2 | 1 |
| passDoor | all | 1 | 0.97 | 0.01 | 3 | 1 |
| returnToDoor | 3,6 | 1 | 0.97 | 0.01 | 4 | 1 |
| dropKey | 3,6 | 1 | 0.97 | 0.01 | 4 | 1 |
| findDoor | all | 1 | 0.93 | 0.03 | 8 | 3 |
| searchKey | 3,6 | 1 | 0.86 | 0.11 | 15 | 11 |
| goToDropOff | all | 1 | 0.79 | 1.21 | 20 | 110 |
| getNear | 1,2,4,5 | 2 | 0.89 | 0.02 | 11 | 2 |
| goToRoom | 1,2,4,5 | 2 | 0.88 | 0.03 | 12 | 3 |
| hasKey | 3,6 | 2 | 0.84 | 0.09 | 16 | 9 |
| transport | all | 2 | 0.77 | 1.49 | 22 | 135 |
| goToRoom | 3,6 | 5 | 0.80 | 0.06 | 21 | 6 |
| deliver | 1,2,4,5 | 5 | 0.67 | 1.05 | 33 | 96 |
| getNear | 3,6 | 7 | 0.62 | 0.15 | 39 | 15 |
| deliver | 3,6 | 10 | 0.30 | 1.45 | 69 | 130 |

Table 1: Results of testing goal execution using a combination of `GTPyhop` and learned goal skills in six domains. 100% of the goals are successfully achieved. The difficulty level of a goal is directly related to the length of the plan.

**Limitations.** There are several limitations we will address in future work. The study we performed relies on a hand-coded GSN. There are several approaches from the literature for learning hierarchies similar to the GSN, so we plan to explore this in future work. The problem we study is based on a discrete gridworld; we plan to examine the extent to which this technique might apply to harder RL problems, for example, robotics. Finally, the GSN provided by GTPyhop did not change during execution. As a proof of concept, this is sufficient, but future work needs to examine how to adapt these approaches to replanning approaches.

## 5 Related Work

To our knowledge, no other approaches have integrated RL with HGN planning to learn an actor's goal-skills. However, there are several works that relate hierarchy, planning, and reinforcement learning in different combinations, and there are many works that relate to the curriculum learning.

**Hierarchy in reinforcement learning.** Introducing hierarchy to RL has the potential to accelerate learning because hierarchical agents can decompose problems into smaller subproblems, or, in other words, learn at multiple levels of abstraction (Levy et al. 2019; Dietterich 1998; Barto and Mahadevan 2003; Jong and Stone 2008). The options framework (Sutton, Precup, and Singh 1999) 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., Chakravorty et al., 2020); very few works allow a 3-layer hierarchy (e.g., Levy et al., 2019). Our work does not presume the number of layers in the hierarchy.

Andreas, Klein, and Levine (2016) 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 to this we provide explicit termination states for each goal skill during training.

Universal Value Function Approximator (UVFA) is a popular baseline method for training neural networks with multiple goals (Schaul et al. 2015). UVFA method 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. Levy et al. (2019) mitigates this problem using a data augmentation technique called hindsight experience replay (Andrychowicz et al. 2017). While all these approaches try to solve the multi-goal RL problem, their approaches don't include hierarchy of goals and planning associated with them.

Konidaris et al. (2012) proposed a technique that segments a demonstration trajectory into a chain of component skills, where each skill is an abstracted goal. Chains from multiple demonstration trajectories are then merged into a skill tree. This work was extended to arbitrary goals by Baggaria et al. (2021), who created Deep Skill Graphs. Future work will explore how to combine the goal-skill network with the skill graph or skill trees.

**Planning and Reinforcement learning.** Illanes et al. (2020, 2019) implemented a hierarchical RL framework with symbolic planning, which achieves sample efficiency for training and is relatively easy to specify high-level goals. The planning representation uses an options-like framework with a two level hierarchy, unlike our more flexible goal networks. Additionally, in their approach, users specify the high-level predicates and actions; in ours, users specify the refinement of the goals and low-level achievement functions.

**Curriculum Learning**. Curriculum learning is a well-known technique in RL. Bengio et al. (2009) show using curriculum learning reduces training times and makes networks more resilient to overfitting. Curriculum learning framework trains the network with the easiest samples at the beginning and then with increasingly harder samples as the network converges. The Goal-Biased Curriculum uses the goals to partition and order the skills that need to be learned. In future work, we plan to explore ways to extend the use of this curriculum.

## 6 Final Remarks

We presented an adaptation of Hierarchical Goal Networks, called hierarchical Goal-Skill Networks, where the leaves of the hierarchical goal-skill network are called goal skills. Every goal skill has a corresponding MDP and policy. Our `TrainGSN` algorithm learns the optimal policies for every goal skill. We used a hierarchical goal planner, `GTPyhop`, to guide `TrainGSN` with the training order of the goal skills. Our experiments show that the resulting curriculum is an effective way to learn and execute the policies in six gridworld environments. Currently, the policies are arbitrarily initialized for each new goal skill, but we hope to explore transfer learning to speed the convergence of subsequent goal skill training across environments.

**Acknowledgments.** This work has been supported for UMD in part by ONR grant N000142012257 and NRL grants N0017320P0399 and N00173191G001. MR thanks ONR and NRL for funding this research. The information in this paper does not necessarily reflect the position or policy of the funders, and no official endorsement should be inferred. [3]

## References

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

Andreas, J.; Klein, D.; and Levine, S. 2016. Modular multitask reinforcement learning with policy sketches. *CoRR* abs/1611.01796.

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; and Zaremba, W. 2017. Hindsight experience replay. In *Proc. NeurIPS*.

Bagaria, A.; Senthil, J. K.; and Konidaris, G. 2021. Skill discovery for exploration and planning using deep skill graphs. In *Proc. ICML*, 521–531.

Barto, A. G., and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems* 13(1):41–77.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. *Proc. ICML* 41–48.

Chakravorty, J.; Ward, P. N.; Roy, J.; Chevalier-Boisvert, M.; Basu, S.; Lupu, A.; and Precup, D. 2020. Option-critic in cooperative multi-agent systems. In *Proc. AAMAS*, 1792–1794.

[3] **Disclaimer** This paper was prepared for informational purposes in part by the Artificial Intelligence Research group of JPMorgan Chase & Co. and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

Chevalier-Boisvert, M.; Willems, L.; and Pal, S. 2018. Minimalistic gridworld environment for OpenAI gym. https://github.com/maximecb/gym-minigrid.

Dietterich, T. G. 1998. The maxq method for hierarchical reinforcement learning. In *Proc. ICML*, volume 98, 118–126.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

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

Illanes, L.; Yan, X.; Icarte, R. T.; and McIlraith, S. 2019. Symbolic planning and model-free reinforcement learning: Training taskable agents. In *Proc. RLDM*, 191–195.

Illanes, L.; Yan, X.; Icarte, R. T.; and McIlraith, S. A. 2020. Symbolic Plans as High-Level Instructions for Reinforcement Learning. *Proc. ICAPS* 30:540–550.

Jong, N. K., and Stone, P. 2008. Hierarchical model-based reinforcement learning: R-max+ maxq. In *Proc. ICML*, 432–439.

Konidaris, G.; Kuindersma, S.; Grupen, R.; and Barto, A. 2012. Robot learning from demonstration by constructing skill trees. *The Int'l Journal of Robotics Research* 31(3).

Levy, A.; Konidaris, G.; Platt, R.; and Saenko, K. 2019. Learning multi-level hierarchies with hindsight. *Proc. ICLR*.

Nau, D.; Patra, S.; Roberts, M.; Bansod, Y.; and Li, R. 2021. GTPyhop: A hierarchical goal+task planner implemented in Python. In *ICAPS Wksp. on Hierarchical Planning (HPlan)*.

Nau, D. 2013a. Game applications of HTN planning with state variables. In *ICAPS Workshop on Planning in Games*. Keynote talk.

Nau, D. S. 2013b. Pyhop, version 1.2.2: A simple HTN planning system written in python. Software release. https://bitbucket.org/dananau/pyhop/src/master/.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. In *Proc. NeurIPS*.

Ram, A., and Leake, D., eds. 1995. *Goal-Driven Learning*. Bradford Press.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. In *Proc. ICML*, 1312–1320.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms.

Shivashankar, V.; Kuter, U.; Nau, D. S.; and Alford, R. 2012. A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proc. AAMAS*, 981–988.

Shivashankar, V.; Alford, R.; Kuter, U.; and Nau, D. S. 2013. The GoDeL Planning System: A more perfect union of domain-independent and hierarchical planning. In *Proc. IJCAI*, 2380–2386.

Shivashankar, V.; Alford, R.; Roberts, M.; and Aha, D. 2016. Cost-optimal algorithms for planning with procedural control. In *Proc. ECAI*, 1702–1703.

Shivashankar, V. 2015. *Hierarchical Goal Networks: Formalisms and Algorithms for Planning and Acting*. Ph.D. Dissertation, Computer Science Dept., University of Maryland.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *AIJ* 112(1):181–211.

Willems, L. 2021. Rl starter files. https://github.com/lcswilliems/rl-starter-files.