Scaling Goal-based Exploration via Pruning Proto-goals

Akhil Bagaria\textsuperscript{1*} and Ray Jiang\textsuperscript{2} and Ramana Kumar\textsuperscript{2} and Tom Schaul\textsuperscript{2}

\textsuperscript{1}Brown University, Providence, RI, USA
\textsuperscript{2}DeepMind, London, UK
akhil_bagaria@brown.edu, tom@deepmind.com

Abstract

One of the gnarliest challenges in reinforcement learning (RL) is exploration that scales to vast domains, where novelty-, or coverage-seeking behaviour falls short. Goal-directed, purposeful behaviours are able to overcome this, but rely on a good goal space. The core challenge in goal discovery is finding the right balance between generality (not hand-crafted) and tractability (useful, not too many). Our approach explicitly seeks the middle ground, enabling the human designer to specify a vast but meaningful proto-goal space, and an autonomous discovery process to refine this to a narrower space of controllable, reachable, novel, and relevant goals. The effectiveness of goal-conditioned exploration with the latter is then demonstrated in three challenging environments.

1 Introduction

Exploration is widely recognised as a core challenge in RL. It is most acutely felt when scaling to vast domains, where classical novelty-seeking methods are insufficient [Taiga \textit{et al.}, 2020] because there are simply too many things to observe, do, and learn about; and the agent’s lifetime is far too short to approach exhaustive coverage [Sutton \textit{et al.}, 2022a].

Abstraction can overcome this issue [Gershman, 2017; Konidaris, 2019]: by learning about goal-directed, purposeful behaviours (and how to combine them), the RL agent can ignore irrelevant details, and effectively traverse the state space. Goal-conditioned RL is one natural formalism of abstraction, and especially appealing when the agent can learn to generalise across goals [Schaul \textit{et al.}, 2015].

The effectiveness of goal-conditioned agents directly depends on the size and quality of the goal space (Section 3). If it is too large, such as treating all encountered states as goals [Andrychowicz \textit{et al.}, 2017], most of the abstraction benefits vanish. On the other extreme, hand-crafting a small number of useful goals [Barreto \textit{et al.}, 2019] limits the generality of the method. The answer to this conundrum is to adaptively expand or refine the goal space based on experience, also known as the discovery problem, allowing for a more autonomous agent that can be both general and scalable.

Taking a step towards this ultimate aim, we propose a framework with two elements. First, a proto-goal space (Section 3), which can be cheaply designed to be meaningful for the domain at hand, e.g., by pointing out the most salient part of an observation using domain knowledge [Chentanez \textit{et al.}, 2004]. What makes defining a proto-goal space much easier than defining a goal space is its leniency: it can remain (combinatorially) large and unrefined, with many uninteresting or useless proto-goals. Second, an adaptive function mapping this space to a compact set of useful goals, called a Proto-goal Evaluator (PGE, Section 4). The PGE may employ multiple criteria of usefulness, such as controllability, novelty, reachability, learning progress, or reward-relevance. Finally we address pragmatic concerns on how to integrate these elements into a large-scale goal-conditioned RL agent (Section 5), and show it can produce a qualitative leap in performance in otherwise intractable exploration domains (Section 6).
2 Background and Related Work

We consider problems modeled as Markov Decision Processes (MDPs) \( \mathcal{M} = (\mathcal{S}, \mathcal{A}, R, T, \gamma) \), where \( \mathcal{S} \) is the state space, \( \mathcal{A} \) is the action space, \( R \) is the reward function, \( T \) is the transition function and \( \gamma \) is the discount factor. The aim of the agent is to learn a policy that maximises the sum of expected rewards [Sutton and Barto, 2018].

Exploration in RL. Many RL systems use dithering strategies for exploration (e.g., \( \epsilon \)-greedy, softmax, action-noise [Lillicrap et al., 2016] and parameter noise [Fortunato et al., 2018; Plappert et al., 2018]). Among those that address deep exploration, the majority of research [Taiga et al., 2020] has focused on count-based exploration [Strehl and Littman, 2008; Bellemare et al., 2016], minimizing model prediction error [Pathak et al., 2017; Burda et al., 2019a; Burda et al., 2019b], or picking actions to reduce uncertainty [Osband et al., 2016; Osband et al., 2018] over the state space. These strategies try to eventually learn about all states [Ecoffet et al., 2021], which might not be a scalable strategy when the world is a lot bigger than the agent [Sutton et al., 2022a]. We build on the relatively under-studied family of exploration methods that maximize the agent’s learning progress [Schmidhuber, 1991; Kaplan and Oudeyer, 2004; Colas et al., 2022].

General Value Functions. Rather than being limited to predicting and maximizing a single reward (as in vanilla RL), General Value Functions (GVFs) [Sutton et al., 2011] predict (and sometimes control [Jaderberg et al., 2017]) “cumulants” that can be constructed out of the agent’s sensorimotor stream. The discounted sum of these cumulants are GVFs and can serve as the basis of representing rich knowledge about the world [Schaul and Ring, 2013; Veeriah et al., 2019].

Goal-conditioned RL. When the space of cumulants is limited to goals, GVFs reduce to goal-conditioned value functions that are often represented using Universal Value Function Approximators (UVFAs) [Schaul et al., 2015]. Hindsight Experience Replay (HER) is a popular way of learning UVFAs in a sample-efficient way [Andrychowicz et al., 2017]. The two most common approaches is to assume that a set of goals is given, or to treats all observations as potential goals [Liu et al., 2022] and try to learn a controller that can reach any state. In large environments, the latter methods often over-explore [Pong et al., 2019; Pitis et al., 2020] or suffer from interference between goals [Schaul et al., 2019].

Discovery of goals and options. Rather than assuming that useful goals are pre-specified by a designer, general-purpose agents must discover their own goals or options [Sutton et al., 1999]. Several heuristics have been proposed for discovery (see Abel [2020] Ch 2.3 for a survey): reward relevance [Bacon et al., 2017; Veeriah et al., 2021], composability [Konidaris and Barto, 2009; Bagaria and Konidaris, 2020], diversity [Eysenbach et al., 2018; Campos et al., 2020], empowerment [Mohamed and Rezende, 2015], coverage [Bagaria et al., 2021a; Machado et al., 2017], etc. These heuristics measure desirability, but they must be paired with plausibility metrics like controllability and reachability to discover meaningful goals in large space. The IMGEP framework [Forestier et al., 2022] also does skill-acquisition based on competence progress, but they assume more structure in the goal space (e.g., Euclidean measure, objects), and use evolution strategies to represent policies instead of RL. STOMP [Sutton et al., 2022b] learns feature attainment options, which are similar to proto-goal achieving policies; but unlike STOMP, we maintain different representations for states and goals. Furthermore, they do not provide a way to prune large feature spaces, nor do they construct new features/goals out of existing ones [Ring, 1994].

3 Goals and Proto-goals

A goal is anything that an agent can pursue and attain through its behaviour. Goals are well formalised with a scalar cumulant \( g : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R} \) and a continuation function \( \gamma_g : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1] \), as proposed in the general value function (GVF) framework [Sutton et al., 2011]. Here, we consider the subclass of attainment goals \( g \), or “endgoals”, which imply a binary reward that is paired with termination. In other words a transition has either \( \langle c_g = 0, \gamma_g > 0 \rangle \) or \( \langle c_g = 1, \gamma_g = 0 \rangle \), i.e., only terminal transitions are rewarding. The corresponding goal-optimal value functions satisfy:

\[
Q^g(s, a) = \mathbb{E}_{s'} \left[ c_g(s, a, s') + \gamma_g(s, a, s') \max_{a'} Q^g(s', a') \right],
\]

with corresponding greedy policy \( \pi^g \) := \arg \max_a Q^g(s, a).

Proto-goals are sources of goals. Since attainment goals can easily be derived from any binary function, we formally define a proto-goal to be a binary function of a transition \( b_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \{0, 1\} \). We assume that, for a given domain, a set \( B \) of such proto-goals can be queried. Proto-goals differ from goals in two ways. First, less formally, the space of proto-goals can easily be derived from any binary function, we formally define a proto-goal to be a binary function of a transition \( b_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \{0, 1\} \). We assume that, for a given domain, a set \( B \) of such proto-goals can be queried. Proto-goals differ from goals in two ways. First, to fully specify a goal, a proto-goal must be paired with a time-scale constant \( \gamma \in [0, 1] \) (a discount), which defines the horizon over which \( g \) should be achieved. The pair \( (b_i, \gamma) \) then define the goal’s cumulant \( c_g(s, a, s') \) := \( b_i(s, a, s') \) and continuation function \( \gamma_g(s, a, s') := \gamma(1 - b_i(s, a, s')) \). Second, less formally, the space of proto-goals \( B \) is vastly larger than any reasonable set goals \( G \) that could be useful to an RL agent. Hence the need for the Proto-goal evaluator (Section 4) to convert one space into the other.

3.1 Example Proto-goal Spaces

A proto-goal space implicitly defines a large, discrete space of goals. Its design uses some domain knowledge, but, crucially, no direct knowledge of how to reach the solution. The most common form is to use designer knowledge about which aspects of an observation are most salient. For example, many games have on-screen counters that track task-relevant quantities (health, resources, etc.). Other examples include treating inventory as privileged in MINECRAFT, sound effects in console video games, text feedback in domains like NETHACK (see Section 6.3 and Figure 2 in the appendix), or object attributes in robotics. In all of these cases, it is straightforward to build a set of binary functions—for example, in NETHACK, a naive proto-goal space includes one binary function for each possible word that could be present in the text feedback.
3.2 Representation
Each observation from the environment is accompanied by a binary proto-goal vector \( b_t \in \{0, 1\}^{|B|} \), with entries of 1 indicating which proto-goals are achieved in the current state (Figure 1). Initially, the agent decomposes \( b_t \) into 1-hot vectors, focusing on goals that depend on a single dimension. As the agent begins to master 1-hot goals, it combines them using the procedure described in Section 3.3, to expand the goal space and construct multi-hot goals.

When querying the goal-conditioned policy \( \pi(a|s, g) \), we use the same 1-hot or multi-hot binary vector representation for the goal \( g \).

3.3 Goal Recombinations
A neat side-effect of a binary proto-goal space \( B \) is that it can straightforwardly be extended to a combinatorially larger goal space with logical operations. For example, using the logical AND operation, we can create goals that are only attained once multiple separate bits of \( b \) are activated simultaneously.\(^1\) One advantage of this is that it places less burden on the design of the proto-goal space, because \( B \) only needs to contain useful goal components, not the useful goals themselves. This is also a form of continual learning [Ring, 1994], with more complex or harder-to-reach goals continually being constructed out of existing ones. The guiding principle to keep this constructivist process from drowning in too many combinations is to operate in a gradual fashion: we only combine goals that in addition to being plausible and desirable (Section 4), have also been mastered (Section 5.3).

4 Proto-goal Evaluator
The Proto-goal Evaluator (PGE) converts the large set of proto-goals to a smaller, more interesting set of goals \( \mathcal{G} \). It does this in two stages: a binary filtering stage that prunes goals by plausibility, and a weighting stage that creates a distribution over the remaining goals \( \mathbb{P}_G : \mathcal{G} \rightarrow [0, 1] \), based on desirability.

4.1 Plausibility Pruning
Implausible proto-goals are those that most likely cannot be achieved (either ever or given the current data distribution). Having them in the goal space is unlikely to increase the agent’s competence; to the contrary, they can distract and hog capacity. We use the following three criteria to eliminate implausible goals:

**Observed:** we prune any proto-goal \( b_t \) that has never been observed in the agent’s experience, so far.

**Reachable:** we prune proto-goals that are deemed unreachable (e.g., pigs cannot fly, a person cannot be in London and Paris at the same time).

**Controllable:** similarly, we prune goals that are outside of the agent’s control (e.g., sunny weather is reachable, but not controllable).

\(^1\)Note that we combine goals, but not their corresponding value-functions [Barreto et al., 2019; Tasse et al., 2022]; we let the UVFA \( Q_{d}(s, a, g) \) handle generalization to newly created goals and leave combination in value-function space to future work.

For the first criterion, we simply track global counts \( N(g) \) for how often we have observed the proto-goal \( b_t \) that corresponds to \( g \) being reached. Estimating reachability and controllability is a bit more involved. We do this by computing a pair of proxy value functions: each goal \( g \) is associated with two types of reward functions (or cumulants)—one with “seek” semantics and the other with “avoid” semantics:

\[
R_{\text{seek}}(s, g) = 1 \text{ if } g \text{ is achieved in } s \text{ else } 0
\]

\[
R_{\text{avoid}}(s, g) = -1 \text{ if } g \text{ is achieved in } s \text{ else } 0.
\]

These seek/avoid cumulants in turn induce seek/avoid policies, and value functions \( V_{\text{seek}}, V_{\text{avoid}} \) that correspond to these policies. Estimates of these values are learned from transitions stored in the replay buffer \( B \).

A proto-goal \( g \) is globally reachable if it can be achieved from some state:

\[
\max_{s \in B} V_{\text{seek}}(s, g) > \tau_1,
\]

where \( \tau_1 > 0 \) is a threshold representing the (discounted) probability below which a goal is deemed to be unreachable.

A proto-goal \( g \) is judged as uncontrollable if a policy seeking it is equally likely to encounter it as a policy avoiding it:

\[
\mathbb{E}_s \left[V_{\text{seek}}(s, g) \right] - \mathbb{E}_s \left[-V_{\text{avoid}}(s, g) \right] < \tau_2,
\]

up to threshold \( \tau_2 \). The set of plausible goals \( \mathcal{G} \) is the subset of those proto-goals induced by \( B \) that satisfy both Eq. 1 and 2.

**Scalably Estimating Many Seek/Avoid Values with LSPI**
As a first line of defense in the process of trimming a vast proto-goal space, the reachability and controllability estimation (and hence the computation of the proxy values \( V_{\text{seek}}, V_{\text{avoid}} \)) must be very cheap per goal considered. On the other hand, their accuracy requirement is low: they are not used for control, and it suffices to eliminate some fraction of implausible goals. Consequently, we have adopted four radical simplifications that reduce the compute requirements of estimating proxy values, to far less than is used in the main deep RL agent training. First, we reduce the value estimation to a linear function approximation problem, by invoking two iterations of least-squares policy iteration (LSPI, [Lagoudakis and Parr, 2003; Ghavamzadeh et al., 2010]), one for the “seek” and one for the “avoid” policy. As input features for LSPI we use random projections of the observations into \( \mathbb{R}^{|\varnothing|} \), which has the added benefit of making this approach scalable independently of the observation size. Third, the estimation is done on a batch of transitions that are only a small subset of the data available in the agent’s replay buffer \( B \) [Lin, 1993].\(^2\) Finally, we accept some latency by recomputing proxy values asynchronously, and only a few times (\( \approx 10 \)) per minute. Section 6.2 shows that such a light-weight approach is indeed effective at identifying controllable goals.

4.2 Desirability Weighting
The second task of the PGE is to enable sampling the most desirable goals from the reduced set of plausible goals \( \mathcal{G} \) produced via pruning. A lot has been discussed in the literature about what makes goals desirable [Gregor et al., 2016;
Bacon et al., 2017; Konidaris and Barto, 2009; Eysenbach et al., 2018; Bellemare et al., 2016; Machado et al., 2017}; for simplicity, we stick to the two most commonly used metrics: novelty and reward-relevance. We use a simple count-based novelty metric [Auer, 2002]:

$$\text{novel}(g) := \frac{1}{\sqrt{N(g)}}$$  \hspace{1cm} (3)

where $N(g)$ is the number of times goal $g$ has been achieved across the agent’s lifetime. The desirability score (or “utility”) of a goal $g$ is then simply $u(g) := R(g) + \text{novel}(g)$, where $R(g)$ is the average extrinsic reward achieved on transitions where $g$ was achieved. Desirability scores for each goal are turned into a proportional sampling probability:

$$\mathbb{P}_{G}(g) := \frac{u(g)}{\sum_{g’ \in G} u(g’)}.$$  \hspace{1cm} (4)

In practice, when queried, the PGE does not output the full distribution, but a (small) discrete set of $K$ plausible and desirable goals, by sampling from $\mathbb{P}_{G}$ with replacement ($K = 100$ in all our experiments).

5 Integrated RL Agent

This section details how to integrate proto-goal spaces and PGE components into a goal-conditioned RL agent. As is typical in the goal-conditioned RL literature, we use a Universal Value Function Approximator (UVFA) neural network to parameterize the goal-conditioned action-value function $Q_{\theta}(s, a, g)$, which is eventually used to pick primitive actions. At the high level, we note that the PGE is used at three separate entry points, namely in determining how to act, what to learn about (in hindsight), and which goals to recombine. What is shared across all three use-cases is the plausibility filtering of the goal space (implausible goals are never useful). However, the three use-cases have subtly different needs, and hence differ in the goal sampling probabilities.

5.1 Which Goal to Pursue in the Current State

For effective exploration, an agent should pursue goals that maximize its expected learning progress, i.e., it should pick a goal that will increase its competence the most [Lopes et al., 2012]. As proxies for learning progress, we adopt two commonly used heuristics, namely novelty [Auer, 2002] (Eq. 3) and reachability [Konidaris and Barto, 2009; Bagaria et al., 2021b]. The issue with exclusively pursuing novelty is that this could lead the agent to prioritise the most difficult goals, which it cannot reach with its current policy yet, and hence induce behaviour that is unlikely to increase its competence. Thus, we combine novelty with a local reachability metric, for which we can reuse the goal-conditioned value $V_{\theta}(s_{t}, g)$, which can be interpreted as the (discounted) probability that the agent can achieve goal $g$ from its current state $s_{t}$, under the current policy $\pi_{g}$. To avoid computing reachability for each goal in $G$, which can be computationally expensive, we instead sample $M$ goals based on novelty and pick the closest:

$$g_{t} = \arg \max_{g \in \{g_{1}, ..., g_{M}\} \sim \text{novel}} \left[ V_{\theta}(s_{t}, g) \right].$$

Stratified Sampling over Heterogeneous Timescales

The attainment count for a goal $N(g)$ can be low because it is rarely reachable, or because it naturally takes a long time to reach. To account for this heterogeneity in goal space, we first estimate each goal’s natural timescale and then use stratified sampling to preserve diversity and encourage apples-to-apples desirability comparisons. To estimate the characteristic timescale (or horizon) $h$ for each goal, we average the “seek” value-function over the state-space: $h(g) := \mathbb{E}_{s \sim G} \left[ V_{\text{seek}}(s, g) \right]$. Once each goal has a timescale estimate, we divide the goals in the goal space into different buckets (quintiles). Then, we uniformly sample a bucket of goals; since the goals in the bucket have similar timescales ($\approx h$), we use novelty and reachability to sample a specific goal from that bucket to pursue (see Algorithm 2 in the appendix for details).

Learning about Extrinsic Reward

The evaluator always picks actions to maximize the extrinsic task reward. If the actors never did during training, then the action-value function would have unreliable estimates of the task return (called the tandem effect [Ostrovski et al., 2021]). So, periodically, the actors pick the task reward function and select actions based on that. Since the task reward function may not correspond to a specific goal, we represent the task reward function as a special conditioning—a $\mathbf{0}$ tensor serves as the goal input to $Q_{\theta}(s, a, g = 0)$.

5.2 Which Goals to Learn about in Hindsight

Once the agent picks a goal $g$ to pursue, it samples a trajectory $\tau$ by rolling out the goal-conditioned policy $\pi_{g}(\cdot; g)$. Given all the goals achieved in $\tau$, $\mathcal{G}_{\tau}^{A}$, the agent needs to decide which goals $G \subset \mathcal{G}_{\tau}^{A}$ to learn about in hindsight.

We always learn about the on-policy goal $g$, and the task reward (which corresponds to the conditioning $g = 0$). Among, the set of achieved goals $G \subset \mathcal{G}_{\tau}^{A}$, the agent samples a fixed set of $M_{\text{her}}$ goals and learns about them using hindsight experience replay [Andrychowicz et al., 2017] (we use $M_{\text{her}} = 15$). Similar to the previous section, we want to sample those $M_{\text{her}}$ goals that maximize expected learning progress. We found that using a count-based novelty score as a proxy for learning progress (sample proportionally to novel($g$)) worked well for this purpose, and outperformed the strategies of (a) learning about all the achieved goals and (b) picking the $M_{\text{her}}$ goals uniformly at random from $\mathcal{G}_{\tau}^{A}$.

5.3 Mastery-based Goal Recombination

We use one simple form of goal recombination in the agent: for any pair of goals that it has mastered, it adds their combination (logical AND) as proto-goal candidate to be evaluated by the PGE. A goal is considered mastered when its success rate is above a pre-specified threshold $\kappa$ ($= 0.6$ in all our experiments). For example, if the agent has mastered the goal of getting the key, and another goal of reaching the door, it will combine them to create a new proto-goal which is attained when the agent has reached the door with the key. Implementation details about creating and managing combination proto-goals can be found in the appendix (Algorithm 5).
5.4 Distributed RL Agent Implementation

For all non-toy experiments, we integrate our method with an off-the-shelf distributed RL agent, namely R2D2 [Kapitwoski et al., 2018]. It is a distributed system of 128 actors asynchronously interacting with 128 environments. The learner is Q-learning-based, using a goal-conditioned action-value function $Q_{θ}(s, a, g)$ parameterized as a UVFA. Experience is stored in a replay buffer $B$, including the binary proto-goal annotation vector $b$. More details about the agent, as well as pseudo-code, can be found in Section C of the appendix.

6 Experiments

Our empirical results are designed to establish proto-goal RL as an effective way to do exploration, first in a classic tabular set-up (TAXI, Section 6.1), and then at scale in two large-scale domains (NETHack and BABA IS YOU, Sections 6.3 and 6.4) whose combinatorial proto-goal spaces, left unpruned, would be too large for vanilla goal-conditioned RL. Alongside, ablations and probe experiments show the effectiveness of our controllability and desirability metrics, and provide qualitative insights into the discovered goal spaces.

6.1 Tabular Experiment: Exploration in TAXI

We build intuition about proto-goal exploration on the TAXI domain, a tabular MDP classically used to test hierarchical RL algorithms [Dietterich, 1998]. In this problem, the agent controls a taxi in a $5 \times 5$ grid; the taxi must first navigate to the passenger, pick them up, take them to their destination (one of 4) and then drop them off. The default reward function is shaped [Randlov and Alstrøm, 1998], but to make it a harder exploration problem, we propose the SPARSETAXI variant, with two changes: (a) no shaping rewards for picking up or dropping off the passenger and (b) the episode terminates when the passenger is dropped off. In other words, the only (sparse) positive reward occurs when the passenger is dropped off at their correct destination.

As proto-goal space, we use a factoring of the state space, namely one $b_i$ for each entity (taxi, passenger, destination) in each grid location ($|B| = 34$). Figure 2 shows the progression of how the PGE gradually refines a goal space from those throughout training. The set of reachable states expands gradually to mimic a curriculum; at first, goals correspond to navigating the taxi to different locations, later they include goals for dropping off the passenger at different depots. Also noteworthy is that proto-goals corresponding to the destination are absent from the goal space, because they are classified as uncontrollable.

In terms of performance, our proposed method of goal-discovery also leads to more sample-efficient exploration in SPARSETAXI (Figure 3). Compared to a vanilla Q-learning agent with $\epsilon$-greedy exploration, our goal-conditioned Q-learning agent learns to reliably and quickly solve the task. More details can be found in Appendix A.

6.2 Verifying the Controllability Measure

Our method of measuring controllability using the discrepancy between “seek” and “avoid” values (Section 4.1) is novel, hence we conduct a set of sanity-checks to verify that it can capture controllability in all its guises. Three toy experiments probe three separate types of controllability:
Static controllability: Proto-goals whose attainment status does not change. The passenger destination in TAXI is a good example of this kind of uncontrollability—while the destination changes between episodes, there is no single transition in the MDP in which it changes.

Time-based controllability: Some problems have a timer that increments, but is not controllable by the agent. We check whether our controllability metric classifies such time-based proto-goals as plausible, using $4 \times 4$ grid-world with a timer that increments from 1–100 (which is the max number of steps in an episode).

Distractor controllability: More generally, parts of the observation that change independently of the agent’s actions are distractors for the purpose of controllability. For this test, we use a visual gridworld, where one image channel corresponds to the controllable player, and the two other channels have pixels that light up uniformly at random [Gregor et al., 2016] (often referred to as a “noisy TV” [Schmidhuber, 2010; Burda et al., 2019a]).

For each of these toy setups, we compare our controllability predictions (Eq. 2) to ground-truth labels, and find it to correctly classify which proto-goals are controllable (see Figure 1 and Section B in the appendix for details). The prediction quality depends on the amount of data used to estimate the seek/avoid values.

6.3 Natural Language Proto-goals: MINIHack

The first large-scale domain we investigate is MINIHack [Samvelyan et al., 2021], a set of puzzles based on the game NETHACK [Küttler et al., 2020], which is a grand challenge in RL. In addition to image-based observations, the game also provides natural language messages. This space of language prompts serves as our proto-goal space—while this space is very large (many 1000s of possible sentences), it contains a few useful and interesting goals that denote salient events for the task. Figure 2 illustrates how word-based proto-goal vectors are created in MINIHack.

We use two variants of the RIVER task as exemplary sparse-reward exploration challenges. We choose them because Samvelyan et al. [2021]’s survey noted that while novelty-seeking algorithms [Burda et al., 2019b; Raileanu and Rocktäschel, 2020] could solve the easiest version of RIVER, they were unable to solve more difficult variations.

In all of these, the agent must make a bridge out of boulders and then cross it to reach its goal. In the NARROWIVER variant, the agent needs to place one boulder to create a bridge, and the difficulty depends on the number of monsters who try to kill the player. Figure 4 (left) shows that while increasing the number of monsters degrades performance, our proto-goal agent outperforms the baseline R2D2 agent on each task setting. In the WIDELAVARIVER variant, the river is wider, requiring 2 boulders for a bridge, and includes deadly lava that also dissolves boulders. Figure 4 (right) shows that our proto-goal agent comfortably outperforms its baseline.

Discovered goal space. Words corresponding to important events in the game find their way into the goal-space. For instance, the word “water” appears in the message prompt when the boulder is pushed into the water and when the player falls into the river and sinks. Later, combination goals like “boulder” AND “water” also appear in the goal-space and require the agent to drop the boulder into the water.
6.4 Doubly Combinatorial Puzzles: BABA IS YOU

The game BABA IS YOU [Teikari, 2019] has fascinating emergent complexity. At first sight, the player avatar is a sheep (“Baba”) that can manipulate objects in a 2D space. However, some objects are “word-blocks” that can be arranged into sentences, at which point those sentences become new rules that affect the dynamics of the game (e.g., change the win condition, make walls movable, or let the player control objects other than the sheep with an “X-is-you”-style rule). The natural reward is to reach the (sparse) win condition of the puzzle after 100s of deliberate interactions.

When testing various RL agents on BABA IS YOU, we observed a common failure mode: the exploration process does not place enough emphasis on manipulating object and text blocks (see also Appendix D.2). So, we created a simple level (U-MAZE shown in Figure 5) that is designed to focus on the crucial aspect of rule manipulation. This puzzle requires the agent to learn how to push a word block in place (from center to bottom left), which adds a new win-condition, and then touch the correct block (on the bottom right). Exploration here is challenging because the agent has to master both navigation and block-manipulation before it can get any reward. In addition, the game’s combinatorially large state space is a natural challenge to any novelty-based exploration scheme.

As in TAXI, we use a simple factored proto-goal space, with one binary element for every object (specific word blocks, wall, sheep) being present at any grid-position. Plausible 1-hot goals could target reaching a specific position of the sheep or movable blocks. Most combinations (2-hot proto-goals) are implausible, such as asking the sheep to be in two locations at once, but some could be useful, e.g., targeting particular positions for both “Baba” and a word-block.

Given the exploration challenges in this domain (R2D2 never sees any reward, even on smaller variants of the puzzle), we use the stronger, state-of-the-art Agent57 agent as baseline here, which adds deep exploration on top of R2D2—it constructs an intrinsic reward using novelty and episodic memory [Badia et al., 2020]. Figure 5 (top right) shows that our R2D2 with proto-goals (but no intrinsic rewards) outperforms Agent57. Note that with careful tuning, Agent57 does eventually get off the ground on this task, but never within the 200M frame budget considered here (see Appendix C.1 for details). On the other hand, Agent57 has the advantage that it does not require engineering a proto-goal space.

Discovered goal space. At first, the goal-space is dominated by navigation goals; once these are mastered, goals that move the word-blocks begin to dominate. Then the agent masters moving to a particular location and moving a word-block to some other location. Eventually, this kind of exploration leads to the agent solving the problem and experiencing
Ablations. Figure 5 (bottom left) analyzes how often the agent should act according to the extrinsic reward instead of picking a goal from the discovered goal-space. When that probability is 0, the agent never reaches the goal during evaluation: acting according to the task reward function 10% of the time during training performed the best in this setting. In a second ablation, Figure 5 (bottom middle) shows the importance of using desirability metrics on top of plausibility when mapping the proto-goal space to the goal-space. Finally, Figure 5 (bottom right) shows the impact of the number of goals sampled for computing local reachability during goal-selection (Section 5.1). Appendix E details other variants tried, how hyperparameters were tuned, etc.

7 Conclusion and Future Work

We presented a novel approach to using goal-conditioned RL for tackling hard exploration problems. The central contribution is a method that efficiently reduces vast but meaningful proto-goal spaces to a smaller sets of useful goals, using plausibility and desirability criteria based on controllability, reachability, novelty and reward-relevance. Directions for future work include generalising our method to model-based RL to plan with jumpy goal-based models, more fine-grained control on when to switch goals [Pislarski et al., 2022], making the proto-goal space itself learnable, as well as meta-learning the ideal trade-offs between the various desirability criteria.

Acknowledgements

We thank Ray Jiang and Ramana Kumar for building the BABA IS YOU environment, Alex Vitvitsky for his patient help with the R2D2 code, Steven Kapturowski and Patrick Pislarski (who spotted a double comma in a reference!) for feedback on an earlier draft. We also thank John Quan, Vivek Veevria, Amol Mandhane, Dan Horgan, Jake Bruce and Charles Blundell for their guidance. Finally, we thank Chentian Jiang, David Abel, Robert Lange, Yash Chandak and Chris Lu for their support and camaraderie.

References


[Pıslar et al., 2022] Miruna Pıslar, David Szepesvari, Georg Ostrovski, Diana Borsa, and Tom Schaul. When should


