APART: Diverse Skill Discovery using All Pairs with Ascending Reward and DropouT

Anonymous Author(s)
Affiliation
Address
email

Abstract

We study diverse skill discovery in reward-free environments [Gregor, Rezende, and Wierstra, 2017], aiming to discover all possible skills in simple grid-world environments where prior methods have struggled to succeed. This problem is formulated as mutual training of skills using an intrinsic reward and a discriminator trained to predict a skill given its trajectory. Our initial solution replaces the standard one-vs-all (softmax) discriminator with a one-vs-one (all pairs) discriminator, and combines it with a novel intrinsic reward function and a dropout regularization technique. The combined approach is named APART: Diverse Skill Discovery using All Pairs with Ascending Reward and DropouT. We demonstrate that APART discovers all the possible skills in grid worlds with remarkably fewer samples than previous works. Motivated by the empirical success of APART, we further investigate an even simpler algorithm that achieves maximum skills by altering VIC, rescaling its intrinsic reward, and tuning the temperature of its softmax discriminator. We believe our findings shed light on the crucial factors underlying success of skill discovery algorithms in reinforcement learning.

1 Introduction

In recent years there is a growing demand for autonomous tools in various challenging domains, including autonomous driving, robot control, and gaming. Deep Reinforcement Learning (Deep RL) has emerged as an effective and efficient solution to tackle these challenges [Schrittwieser et al., 2020, OpenAI et al., 2019, Vinyals et al., 2019]. Deep RL algorithms aim to maximize a given reward function that reflects the system's objectives, yet in many complex real-world scenarios, specifying this reward explicitly is difficult. This difficulty can arise from numerous reasons such as lack of explicit reward, misspecification of the reward, or due to having to weight multiple rewards that signify different, possibly contradictory, goals. For example, in an autonomous driving scenario, determining how should we weigh collisions versus speed, or measuring risk, can be challenging. Other factors such as comfort, traffic rules, and aggressiveness also need to be considered.

This issue has recently sparked a surge of research interest in tackling RL without a clear reward. Previous work has shown that, one way of overcoming reward misspecification, is to utilize a diverse set of policies [Kumar et al., 2020, Zahavy et al., 2022]. These policies can later be used, for example, as a starting point for learning multiple goals, in hierarchical RL, for manual evaluation, or for pure exploration. This approach may also be viewed as a small step towards unsupervised RL, where the goal is to learn informative representations of the environment by interacting with it and reaching simple, reward-free, self driven goals.

One method for obtaining a diverse set of policies is by letting each maximize some intrinsic reward. Unlike the extrinsic reward, the intrinsic reward is not derived from the environment and is chosen specifically to encourage diversity. Our work follows Gregor et al. [2017] in which the intrinsic

Submitted to 16th European Workshop on Reinforcement Learning (EWRL 2023). Do not distribute.

reward is derived from a discriminator (classifier), used to discriminate between the different policies (skills) based on their latest visited states.

Prior work formulates this as a stochastic process in which a skill is drawn from a probability distribution, thereafter played in the environment. The skills and the discriminator are trained jointly as to maximize the mutual information (MI) between two random variables: the skill index (latent variable) and its latest visited state. Fixing the discriminator, an RL algorithm updates the skill associated with the current latent variable, attempting to maximize the MI as a reward function. Fixing the skills, the discriminator is trained in a supervised manner to identify the correct latent variable from only its latest visited state.

MI maximization methods were followed-up in Eysenbach et al. [2019], Achiam et al. [2018] in which the prior distribution over the skills is fixed, and the discriminator is fed with the entire trajectory of a skill rather than just the final state. More recent approaches employ more complex solutions, such as adding an explicit exploration term [Strouse et al., 2021], incorporating additional structure in the sampling and discrimination of skills [Baumli et al., 2021, Hansen et al., 2021] or using a complex multi-staged algorithm [Kamienny et al., 2022, Campos et al., 2020]. However, these methods still fall short in discovering all possible skills even in simple environments such as tabular grid worlds.

In our work, we tackle the issue of incomplete skill discovery by directly improving the classic MI maximization approach, and without adding any additional algorithmic complexity. During our investigation, we identified that the primary cause of the issue is the structure of the rewards used to train the skills. In what follows, we describe our path to reach this conclusion, and as an intermediate result we describe our method APART that does discover all skills. We demonstrate this experimentally.

We begin our investigation by replacing the discriminator's commonly used "One vs All" classification method with "All Pairs" classification, in which $\binom{K}{2}$ binary classifiers are trained to distinguish between each pair of classes [see, e.g., Mohri et al., 2018, Shalev-Shwartz and Ben-David, 2014], where K is the number of classes. This, however, raises the problem of how to construct a reward function for training the skills from the classifier output, which can be inferred in many ways including: using average vote or minimum vote. The former takes all pair comparisons that involve the current skill and calculates an average score, similarly to the one vs all reward. The latter chooses the minimum score, resulting in a reward that, intuitively, attempts to minimize the error compared to the "closest" skill. We test both approaches and opt to use the minimum vote to achieve better skill discrimination, which immediately translates to improved diversity metrics. To further enhance our method, we employ a weighted dropout perturbations over the reward. Since all trajectories begin at the same position, we give less importance to the beginning of each episode, where all skills are packed together and are difficult to differentiate, and focus more on later stages when skills are more separable. This approach can be thought of as interpolating between discriminating based on the last state [Gregor et al., 2017] and discriminating based on the entire trajectory [Eysenbach et al., 2019]. The combination of the All Pairs classifier with minimum vote reward and dropouts constitutes our method, which we name APART.

Finally, we conduct a further investigation to better understand the key components that contribute to the success of APART. This leads us to develop a simpler algorithm that alters the classic VIC algorithm with two modifications: rescaling of the skill intrinsic rewards, and changing the temperature of the softmax function. We then perform an empirical evaluation to demonstrate that this modified algorithm (tuned VIC) achieves performance comparable to APART.

1.1 Additional Related Work

Recent works on diverse skill discovery typically add additional algorithmic mechanisms on top of MI maximization. Strouse et al. [2021] adds an exploration bonus to help the discriminator observe enough training examples and overcome pessimistic exploration. Kamienny et al. [2022] include a randomly diffusing part which adds to local exploration around the last state. Kwon [2020] suggest that the intrinsic reward is subject to bias and employ a Gaussian mixture model. Campos et al. [2020] is a three stage methodology: exploration by training a policy to induce a uniform distribution of latent variables over states, skill discovery by training an encoder-decoder to infer latent skills sampled from this distribution, and skill learning by maximizing MI. Sharma et al. [2019] change the objective to embed learned primitives in continuous spaces. Other MI skill discovery methods

include Chen et al. [2022], Baumli et al. [2021], Hansen et al. [2021]. As previously mentioned, our work aims to learn skills by using MI objective without any additional goals or stages.

2 Preliminaries

2.1 Diverse Skill Discovery

Following Eysenbach et al. [2019], we say that a set of skills is diverse if the skills are **distinguishable** and **separable**. Skills are distinguishable if they reach different states.[1] Skills are separable if there are large distances between them according to some metric (see Appendix C.4 for different choices of metrics).

After being proposed in Gregor et al. [2017], mutual information (MI) maximization became a common method for diverse skill discovery [Eysenbach et al., 2019, Warde-Farley et al., 2019, Hansen et al., 2020, Baumli et al., 2021, Strouse et al., 2021]. To define the MI, we consider a stochastic process in which a latent categorical variable z, which represents a specific skill, is sampled from a prior distribution $p(z)$. Like Eysenbach et al. [2019], we fix the prior distribution p to be uniform over all latent variables z. After z is drawn, the corresponding skill is "played" in the Markov Decision Process and we observe the generated trajectory via a conditional policy $\pi(a|s, z)$. Then, the aforementioned MI is defined between the random trajectory of the skill and the latent variable z.

Notations. $\mathcal{S}$ denotes a visited state. $\mathcal{A}$ denotes the action. $\mathcal{Z} \sim p(z)$ denotes a latent variable on which we condition our policy. A policy conditioned on a fixed $\mathcal{Z}$ is called a skill. Without limitation of generality, $p(z)$ is a uniform distribution and $\mathcal{Z}$ is discrete. $\mathcal{I}(\cdot;\cdot)$ is the Mutual information. $\mathcal{H}[\cdot]$ is Shannon entropy.

We formally define MI maximization as $\mathcal{I}(\mathcal{S};\mathcal{Z})$. Intuitively, maximizing the MI aims for a one-to-one correspondence between the skill and its resulting trajectory. In addition, we would also like to minimize the information between actions and states: $\mathcal{I}(\mathcal{A};\mathcal{Z} \mid \mathcal{S})$, and maximize the entropy $\mathcal{H}(\mathcal{A}|\mathcal{S})$ to assist in exploration. The full objective $\mathcal{F}(\theta)$ as defined in Eysenbach et al. [2019] is in Eq. (10) in Appendix A.3, and results in:

$$\mathcal{F}(\theta) \geq \mathcal{H}[\mathcal{A}|\mathcal{S}, \mathcal{Z}] + \mathbb{E}_{z\sim p(z), s\sim\pi(z)}[\log q_\sigma(z|s) - \log p(z)] \tag{1}$$

Indeed, replacing $p(z|s)$ with a parametric model $q_\phi(z|s)$ (the discriminator) provides a variational lower bound $G(\theta, \phi)$ on our objective $F(\theta)$ [Agakov, 2004], where θ are the policy parameters.

In DIAYN, since an actor critic policy includes entropy over actions (for exploration purposes), the objective can be further simplified as the following, where $r_z(s, a)$ is the reward:

$$r_z(s, a) = \log q_\sigma(z \mid s) - \log p(z). \tag{2}$$

2.2 One-vs-All and All Pairs Classification

One of the goals of this work is to showcase the significant improvement in diversity metrics achieved by switching from a One-vs-All to an All Pairs discriminator. To that end, we first review One-vs-All and All Pairs classification methods and highlight their distinctive features.

Multi-class classification only gained attention in later years after binary classification [e.g., Kearns and Vazirani, 1994, Cortes and Vapnik, 1995] became reasonably well-understood. As a result, the common approach for solving multi-class classification problems involved reducing them to learning multiple binary classifiers [Allwein et al., 2000, for example], and two prevalent reduction schemes emerged: One-vs-All and All Pairs [Mohri et al., 2018, Shalev-Shwartz and Ben-David, 2014].

One-vs-All (OvA) classification is the more widespread approach. It requires learning K different binary classifiers, where classifier i separates class k_i from all other classes. A typical implementation of OvA classification assigns K different score values for each example, then selects the class with maximal score. In deep neural networks this approach usually use a softmax activation function in the final layer. The last layer outputs the K score logits $o_1, \ldots, o_K$, and the network is trained to

[1] Note that skills are still distinguishable if they visit different states using a similar choice of actions.

minimize the categorical cross-entropy loss:

$$J^{\text{ova}} = -\sum_{i=1}^{K} y_i \log \hat{y}_i, \quad \text{where} \;\; \hat{y}_i^{\text{ova}} = \frac{\exp(o_i)}{\sum_{j=1}^{K} \exp(o_j)},$$

and where y_i is the ground truth label.

In contrast to OvA, All Pairs (AP) classification[2] employs a classifier for each pair of classes, resulting in a total of $L = K(K-1)/2$ classifiers. A classifier associated with classes $i \neq j$ predicts whether an example belongs to either class i or class j. Inference can be challenging for AP classifiers, since the L classifiers may produce contradicting outputs, necessitating a combination of their outputs to determine the correct label (for example, using a majority vote). Nevertheless, this is not a concern in our work as will be elaborated in the sequel.

To apply AP classification in a deep network, we adopt the method of Pawara et al. [2020]. We consider each of the L classifiers as a binary classifier whose output is in $\{\pm 1\}$, and the final network layer outputs L outputs for each classifier, denoted $o_1, \ldots, o_L$. The outputs are normalized to $[-1, 1]$ using the hyperbolic tangent function: $\hat{y}_i^{\text{ap}} = \tanh(o_i)$.

Given a labeled example, we assign each classifier i its correct binary label y_i^{ap}. To do so, we introduce a code matrix M_c of size $K \times L$ where each cell j, i contains the binary label for classifier i when the correct label is j. The values in the code matrix are in $\{-1, 0, 1\}$, where 0 indicates that the classifier is indifferent to the result (i.e., "don't care"). An example for the code matrix for $K = 5$ is presented in Appendix A.2. We represent the target label y as a one hot vector. Multiplying it with the code matrix yields a vector of binary labels of size L, as in Eq. (3) We then train the network to minimize the average cross-entropy terms of all the pairs as in Eq. (4):

$$y^{\text{ap}} = M_c^{\top} y. \quad (3) \qquad J^{\text{ap}} = -\frac{1}{L} \sum_{i=1}^{L} \left(\frac{1 + y_i^{\text{ap}}}{2} \log \frac{1 + \hat{y}_i^{\text{ap}}}{2} + \frac{1 - y_i^{\text{ap}}}{2} \log \frac{1 - \hat{y}_i^{\text{ap}}}{2} \right). \quad (4)$$

While OvA classification are typically preferred in supervised learning due to their simplicity, reduced number of parameters, and ease of inference, AP classification has also been successful in various settings and was benchmarked alongside One-vs-All with similar performance. These setting include support vector machines [Zhang et al., 2017], and deep neural networks [Pawara et al., 2020, Galar et al., 2011, Ou and Murphey, 2007]. Moreover, theoretical results Daniely et al. [2012] have shown that AP classifiers are more powerful than OvA classifiers (for linear models) in the sense that they can represent more complex decision rules, but require more examples to generalize. In an RL setting, given a simulator, one could generate as many training examples as needed, so AP classification could potentially improve performance "for free". Somewhat contradictory to the aforementioned theoretical results, Rifkin and Klautau [2004] notice that AP classifiers tend to converge much faster in practice. In what follows we provide further evidence to support this observation.

3 APART: All Pairs with Ascending Reward and DropouT

In this section we introduce the APART algorithm for discovering diverse skills. We now summarize the main elements of APART that differentiate it from prior algorithms. These elements are outlined in Fig. 1, illustrating where each of them comes into play within APART.

(i) An **all pairs discriminator** (Section 2.2).
(ii) New **intrinsic reward functions** that we derived for the AP discriminator.
(iii) **Soft dropout regularization**, designed to improve the sample efficiency when using our new intrinsic reward.

We evaluate APART experimentally in Section 4, where we also ran an ablation study and demonstrate that the combination of modifications is necessary for APART's success. In the following, we provide a detailed explanation for each of the modifications.

Intrinsic reward functions. Since the AP discriminator no longer provides the log probability of each skill, we introduce two reward functions compatible with an AP discriminator. To that end,

[2] All Pairs is also sometimes known as One-vs-One classification or All-vs-All classification.

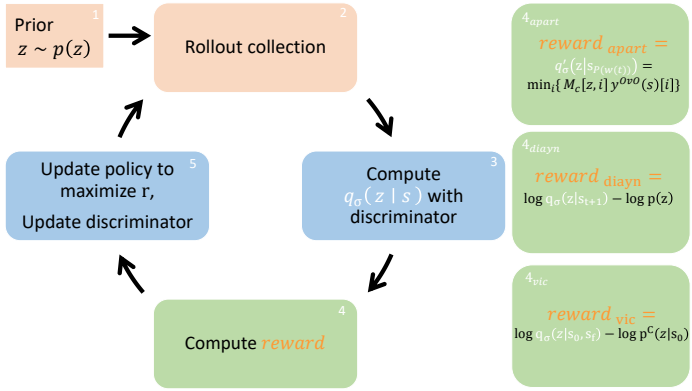


Figure 1: The APART algorithm. Left: General scheme for mutual information maximization algorithms. (1) A latent variable z is sampled from a prior distribution $p(z)$. (2) The agent interacts with the environment according to a policy $\pi(\theta, z)$. (3) Evaluate the discriminator $q_\sigma(z \mid s)$ and train it to minimize the classification loss for predicting the latent variable z. (4) Calculate an intrinsic reward $r_z(s,a) = \log q_\sigma(z \mid s) - \log p(z)$. (5) Train the policy to maximize $r_z(s,a)$. Right: Intrinsic rewards for DIAYN, VIC and APART.

recall that the standard way to classify a new example in an AP classifier is to combine all of the L outputs $\hat{y}$ by multiplying them with the code matrix M_c (Section 2.2). This outputs a vector of scores for each of the K classes, similarly to the output of an OvA classifier. Applying a softmax function over this vector represents a probability distribution over the classes.

We now define our first reward function, called **Average All-Pairs** reward, designed to mimic the intrinsic reward of DIAYN (Eq. (2)) using the output of the AP classifier.

$$q_\sigma(z = y \mid s)_{average} = r_{average} = y_z^{\text{ap}} \cdot \text{softmax}(M_c \hat{y}^{\text{ap}}), \tag{5}$$

where $M_c\hat{y}^{\text{ap}}$ are the predicted all pairs classification values multiplied with the code matrix, y_z^{ap} is a one hot vector indicating the current skill z (Eq. (3)) and $r_{average}$ is the average reward. In this case, $q_\sigma(z \mid s)_{average}$ can be seen as equivalent to $q_\sigma(z \mid s)$ in Eq. (9), since it is approximating the same posterior, and therefore can be substituted to get a variational lower bound on the mutual information. Nevertheless, in our case we don't use the log probabilities for the reward but rather the probabilities themselves [also used in Baumli et al., 2021].

Second, we describe the **Min All-Pairs** reward. This reward corresponds to taking the minimum between all pair comparisons of the current skill and all other skills [Zahavy et al., 2022]. Intuitively, this leverages the structure of the AP classifier to measure distance between skills to try to separate the current skill from its "closest" skill. Note that this method does not maximize the MI explicitly. See Appendix B.4 for an analysis on MI maximization. Our intrinsic reward is defined as:

$$y_z^{\text{ap}}(s) = [y_i^{\text{ap}}]^T \odot \hat{y}_i^{\text{ap}}, \quad r_{apart} = \min_i \{y_z^{\text{ap}}(s)[i]\}. \tag{6}$$

where y_i^{ap} is the relevant row from the code matrix (Eq. (3)) corresponding to the latent variable, and $\hat{y}^{\text{ap}}$ is the hyperbolic tangent applied on the prediction of the discriminator. The $\odot$ notation expresses element-wise multiplication of those two terms resulting in a vector size $1 \times K$. $y_z^{\text{ap}}(s)$ represents pair-wise classification between target latent variable and all other latent variables, in the range $[-1, 1]$. Positive classification scores mean classification has correctly identified the target latent variable, while negative scores mean a misclassification (higher absolute value in the prediction is related to confidence). Therefore, when taking the minimum of this vector, we take the worst pair-wise classification score in comparison to the latent variable.

Soft dropout regularization. One of the important differences between VIC and DIAYN are the time steps in which the RL algorithm receives a nonzero intrinsic reward. In VIC, only the terminal states are rewarded while in DIAYN all states are rewarded.

Our first regularization technique is inspired by both of these ideas and interpolates between them. Concretely, we introduce a **time-ascending mechanism** that gives smaller weights for early time

steps and larger weights for time steps that closer to the end of the episode. This property is desired when all the skills start from the same state as is usually the case since in the early time steps of the episode most skills occupy the same states, whereas in later stages they are more likely to spread out.

We note that this mechanism is a regularization technique and we are still interested in the VIC objective, that is, to find skills that terminate at different states. More explicitly, our time-ascending weights are defined in Eq. (7) where t denotes the time step, T denotes trajectory length, and W is the weight. We note that Eq. (7) presents a specific choice for the weight function, but any other monotonically increasing function can be chosen instead (see Fig. 17 in the appendix for comparison with other functions).

Our second regularization mechanism uses the ascending weights as dropout probabilities to implement a random intrinsic reward. Since the expected value for a random variable X for a Bernoulli distribution doesn't change, the expectation over the intrinsic reward stays the same after the dropout, implying that the optimal policy stays the same $\mathbb{E}_{z\sim p(z), s\sim\pi(z)}[r(s,t)] = r(s)$.

With these two mechanisms combined, our APART intrinsic reward is defined in Eq. (8).

$$W(t) = (t/T)^2, \quad (7) \qquad r^{\text{apart}}(s,t,z) = \begin{cases} r(s,z), & \text{with prob. } W(t). \\ 0, & \text{otherwise.} \end{cases} \quad (8)$$

We provide a pseudo code for the entire APART algorithm in Algorithm 1 in the appendix.

4 Experiments

We evaluate APART through a series of experiments on multiple grid worlds. We first present our setup, then analyze the learned skills with comparison to prior work. Finally, we dive into each algorithm component and analyze its contribution and alternative approaches.

Setup. Our goal to find all skills in simple gridworld environments, something that previous work on diverse skill discovery [Eysenbach et al., 2019, Gregor et al., 2017, and others] do not achieve. For that, we test three different gridworld environments: four rooms, empty, and u-maze (see Appendix C.3). We use a single layer fully connected architecture and one hot input states (Appendix C.1). Although number of discoverable skills is different per each environment (85 in four rooms, 100 in empty and 72 in u-maze), we use a constant number of latents $N_z = 100$. We evaluate our algorithm and compare it to previous work by measuring the effective number of skills (Appendix C.4). A more detailed setup is found in Appendix C.1

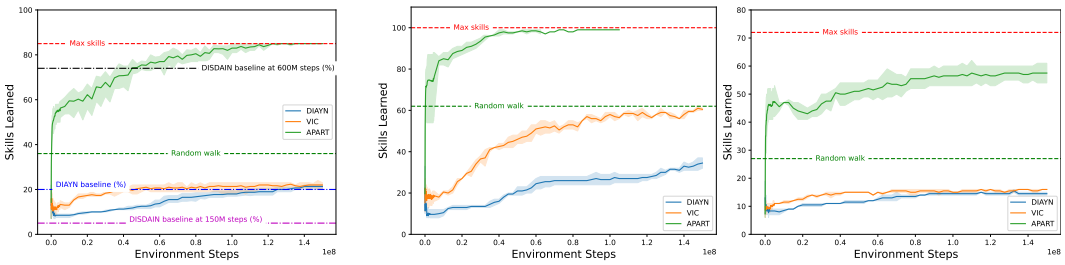
Baselines. We compare our algorithm to two main baselines: DIAYN [Eysenbach et al., 2019] and VIC [Gregor et al., 2017]. Our version of VIC differs from the original version in that we're using a uniform prior distribution instead of learning it. This was shown to be a better choice in Eysenbach et al. [2019], and it narrows the gap between the baselines and our algorithm. The remaining difference, therefore, between DIAYN and VIC is the rewarded states: in VIC, only the final step is rewarded, while in DIAYN all the states are rewarded. We also compare to DISDAIN [Strouse et al., 2021] where relevant.

4.1 Results

We now focus on reviewing the learned skills through a series of experiments and ablation studies.

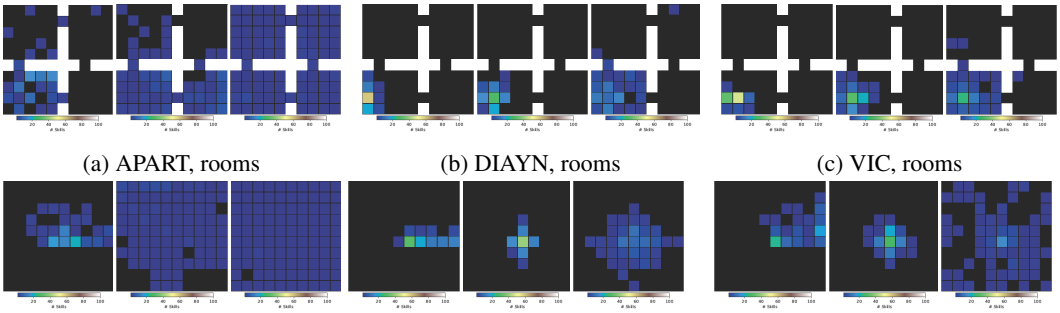
Number of learned skills. Fig. 2 provides a comparison between APART and the baselines in terms of number of learned skills. a: In four rooms, we see that DIAYN and VIC don't exceed 30 skills after convergence of over 100M steps [similar results are reported in Strouse et al., 2021]. DISDAIN[3] does not converge when limiting environment steps to 150M. At 600M steps, convergence achieved by the best seed is reported to be 86% of skills. APART achieves all skills with all seeds in approximately 100M steps with a stable convergence. In Fig. 3 (left side), results are supported with an example rollout before, during and after convergence. While DIAYN and VIC barely leave the first room in the four rooms environment, APART manages to not only visit all rooms but also reaches all available states. b: In empty, DIAYN and VIC converge to less than 60 skills, while APART reaches

[3]DISDAIN reported results are in a percentage from maximum available states (104 available states vs 85 available states in our configuration). DISDAIN uses 20 steps while we use 40



(a) Four Rooms (b) Empty (c) U-maze

Figure 2: **Skills learned in grid worlds:** (a) Four Rooms (b) Empty (c) U-maze. Continuous lines (mean $\pm$ std): green- APART, blue- DIAYN, orange- VIC, **dashed lines**: red- maximum available states in grid (number of unique available states in grid- effectively grid size without walls), green- mean result of a random walk agent (equally distributed actions per step), **dash-dot lines**- results reported on [Strouse et al., 2021]: black- DISDAIN on 600M steps, pink- DISDAIN on 150M steps, blue- DIAYN on 40M steps.



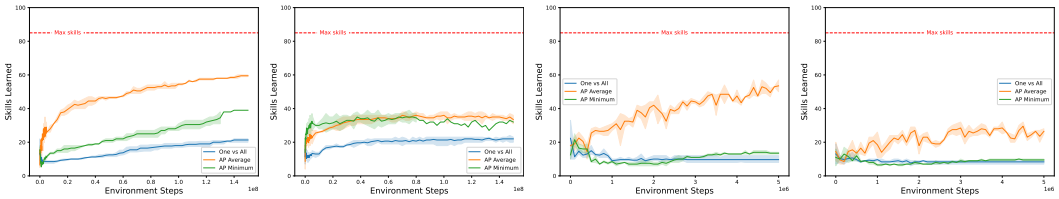
(a) APART, rooms (b) DIAYN, rooms (c) VIC, rooms

(d) APART, empty (e) DIAYN, empty (f) VIC, empty

Figure 3: **States reached during training** with APART (a, d), DIAYN (b, e) and VIC (c, f), on four rooms and empty. Each plot represents a different training point: Left to right: beginning of training, before convergence (5M steps), after convergence (150M steps).

all available states after 50M steps. In Fig. 3 (right side), DIAYN and VIC are centered and don't reach all distant locations while APART converges to reach all available states. c: In u-maze, DIAYN and VIC converge to less than 20 skills, while APART reaches more than 50 available skills (out of the 72 possible states).

All Pairs vs. One vs All. Here we focus on the improvements achieved by changing our classifier to use the all pairs classification, instead of the standard one vs all method. These improvements are studied using ablations and alternative approaches. In Fig. 4, we see the effect of using the all pairs



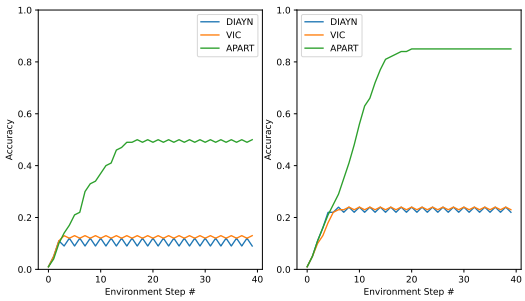
(a) Rooms, All States (b) Rooms, Last States (c) Empty, All States (d) U-maze, All States

Figure 4: **All pairs study.** Effect of all pairs on learning. **All States**: blue- all pairs with minimum reward, orange- all pairs with average reward, green- one vs all. **Last states**: blue- all pairs with minimum reward, orange- one vs all.

classifier. In a, a DIAYN setup is used on rooms (all state contribute to reward), b show a VIC setup

on rooms (only last states contribute to reward), c shows using all states on empty, with 5M steps and d shows using all states on u-maze with 5M steps. Evaluations on 5M steps give us the opportunity to see the trend while saving compute resources along the way. In all graphs, all pairs exceeds the one vs all, whether if the all pairs reward type is the average reward or the minimum reward. When comparing the minimum reward and the average reward setups by themselves, we can see that AP Average achieves the highest scores. However, it does not achieve the maximum environment score as in Fig. 3

Reward type study. The focus of this section is on the improvements we achieve through the addition of ascending rewards and dropouts. An ablative approach is used to study these improvements.



(a) 5M steps (b) 150M steps

Figure 5: Discriminator accuracy per environment step, rooms.

Classifier	Reward	Ascending	Dropout	Result, 5M steps
AP	Min.	-	-	10 ± 1.5
	Min.	+	-	46 ± 1
	Min.	-	+	10 ± 0
	Min.	+	+	$\mathbf{56.25 \pm 3.7}$
	Avrg.	-	-	26 ± 0
	Avrg.	+	+	25 ± 0
OvA	Avrg.	-	-	8.5 ± 0.8
	Avrg.	+	-	12 ± 0
	Avrg.	-	+	9.5 ± 2.5
	Avrg.	+	+	8.5 ± 0.5

Figure 6: Skills Learned by Reward Type, mean $\pm$ std, rooms.

In Fig. 6 we see that while adding the ascending reward and dropout doesn't affect the average reward methods, it does affect learning with the minimum reward. While AP is shown to always be a more successful choice, we can see that when pairing minimum reward AP with ascending reward and dropout we achieve the most learned skills. In order to characterize the behavior during the episode, Fig. 5 presents the discriminator accuracy per environment step t, in 5M steps and 150M steps, where accuracy $= \frac{\text{Correct predictions}}{\text{Total predictions}}$. The three methods achieve lower accuracy in the beginning of the episode since all skills are tightly packed. With all methods, the accuracy ascends up to a certain value and stabilizes around it. It is important to point out that discriminator accuracy has to be paired with a policy, since it is challenging to achieve high accuracy when skills are not spread out. Nonetheless, we see that our method achieves much higher accuracy. While other methods do not exceed accuracy of 25%, APART achieves 85% accuracy, which is the maximum achievable accuracy since this is the number of separable states in rooms environment.

5 Epilogue: Tuning VIC

In this section, we modify the VIC algorithm to match performance of APART. It is well known that Deep RL algorithms are sensitive to the scale of the reward function, and many prior works have tried to address this challenge [Hessel et al., 2021, Kapturowski et al., 2019, Hasselt et al., 2016]. Here, we take a similar approach and tune the scale of the reward of VIC to be similar to the scale of APART's reward. Our simple modifications allow VIC to discover all the skills in our tested grid world environments (Appendix C.3), outperforming multiple prior works.

Our motivation follows by recalling that APART uses an AP classifier whose outputs not represent probabilities. Instead a $tanh$ activation bounds the output in $(-1, 1)$ (Section 2.2). Inspired by this, we first decided to remove the $\log$ function from the reward in VIC, an approach reported to work well in relative VIC [Baumli et al., 2021]. Second, we tuned the temperature of the softmax function to match the scale of APART's reward.

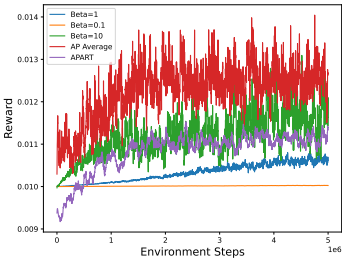
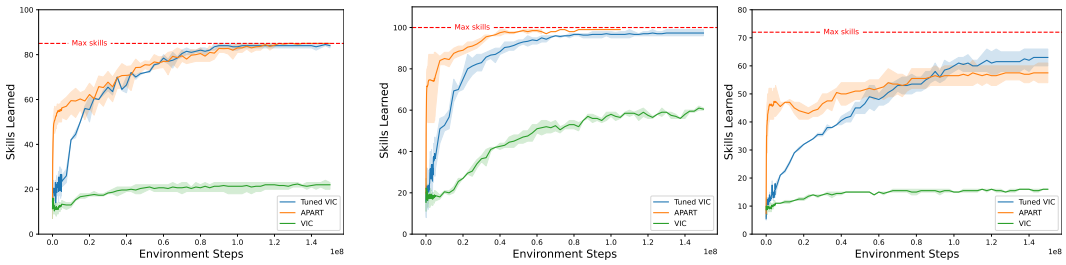


Figure 7: Learned rewards from five different methods: APART and AP Average, Tuned VIC with: $\beta = 10, 1, 0.1$. Rewards are smoothed with $\sigma = 5$.

Fig. 7 compares reward scale for three different values of β, the inverse temperature of the VIC's softmax function. As can be seen, for $\beta = 10$, the scale of the VIC reward matches the scale of the APART reward. In Fig. 20 (Appendix E), we also provide the scale of the vanilla VIC reward, which is observed to be larger than that of APART. We also note that for $\beta = 10$, the probability mass is more concentrated on the outcome with the highest probability, see Appendix B.3 for more details.

Results. We evaluated the performance of the tuned VIC algorithm on the same grid world environments. The results are presented in Fig. 8. The convergence in APART is faster, but the final performance is comparable across all three environments, with a slight advantage for tuned VIC in the U-maze and a slight advantage for APART in the empty environment.



(a) Four Rooms (b) Empty (c) U-maze

Figure 8: **Skills learned in grid worlds:** (a) Four Rooms (b) Empty (c) U-maze. Continuous lines (mean $\pm$ std): orange- APART, blue- Tuned VIC, $\beta = 10$, green- VIC. **dashed red line**: maximum available states in grid (number of unique available states in grid- effectively grid size without walls).

6 Discussion

In this work we introduced APART, a method for skill discovery without an extrinsic reward function.

Most previous works on diverse skill discovery are unable to discover all skills even on simple tabular environments. There are a few cases where tailored solutions such as adding an explicit exploration term [Strouse et al., 2021] or using a complex multi-staged algorithm [Kamienny et al., 2022, Campos et al., 2020] managed to do so. APART, on the other hand, does not change the diversity objective and does not require complicated multi-stage training, yet it can discover all the skills in these environments.

Furthermore, APART demonstrated superior skill discovery, compared to prior work in multiple grid world environments. While grid world environments are arguably a small-scale case study, our gains are significant and outperform multiple commonly used baselines. As such we believe that our insights will be useful for future research. Furthermore, our ablative analysis suggests that each one of the components in APART contributed to its performance, and in particular only utilizing our discriminator achieves a much higher accuracy than those found in prior work.

Inspired from this success, we further studied the baseline VIC algorithm and showed that it can be improved to discover all the skills on grid worlds with two simple steps. These are very encouraging results, considering that the reported results on this benchmark in multiple papers [Eysenbach et al., 2019, Strouse et al., 2021, etc.], were much lower.

Our results in simple grid world environments can be further investigated in more complex environments, such as ATARI [Bellemare et al., 2013] and Mujoco [Todorov et al., 2012]. As a first step towards this, we demonstrate APART's success when using pixel observations (Fig. 19).

Finally, we hope our study on using alternative classification techniques in diverse skill discovery will inspire others to explore more ideas in this direction, such as error correction codes and hierarchical classification. Such investigation in this domain is rather appealing since there is no limit on the amount of samples an RL agent can collect in a simulation. These may spark the definitions of richer notions of diversity.

References

J. Achiam, H. Edwards, D. Amodei, and P. Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

D. B. F. Agakov. The im algorithm: a variational approach to information maximization. *Advances in neural information processing systems*, 16(320):201, 2004.

E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of machine learning research*, 1(Dec):113–141, 2000.

K. Baumli, D. Warde-Farley, S. Hansen, and V. Mnih. Relative variational intrinsic control. In *Proceedings of the AAAI conference on artificial intelligence*, pages 6732–6740, 2021.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

V. Campos, A. Trott, C. Xiong, R. Socher, X. Giró-i Nieto, and J. Torres. Explore, discover and learn: Unsupervised discovery of state-covering skills. In *International Conference on Machine Learning*, pages 1317–1327. PMLR, 2020.

W. Chen, S. Huang, Y. Chiang, T. Chen, and J. Zhu. Dgpo: Discovering multiple strategies with diversity-guided policy optimization. *arXiv e-prints*, pages arXiv–2207, 2022.

C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

A. Daniely, S. Sabato, and S. Shwartz. Multiclass learning approaches: A theoretical comparison with implications. *Advances in Neural Information Processing Systems*, 25, 2012.

B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*. OpenReview.net, 2019.

M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera. An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. *Pattern Recognition*, 44(8):1761–1776, 2011.

K. Gregor, D. J. Rezende, and D. Wierstra. Variational intrinsic control. In *International Conference on Learning Representations*. OpenReview.net, 2017.

S. Hansen, W. Dabney, A. Barreto, D. Warde-Farley, T. V. de Wiele, and V. Mnih. Fast task inference with variational intrinsic successor features. In *ICLR*. OpenReview.net, 2020.

S. Hansen, G. Desjardins, K. Baumli, D. Warde-Farley, N. Heess, S. Osindero, and V. Mnih. Entropic desired dynamics for intrinsic control. *Advances in Neural Information Processing Systems*, 34: 11436–11448, 2021.

H. V. Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver. Learning values across many orders of magnitude. *NIPS*, 2016.

M. Hessel, I. Danihelka, F. Viola, A. Guez, S. Schmitt, L. Sifre, T. Weber, D. Silver, and H. V. Hasselt. Muesli: Combining improvements in policy optimization. *International Conference On Machine Learning*, 2021.

P. Kamienny, J. Tarbouriech, S. Lamprier, A. Lazaric, and L. Denoyer. Direct then diffuse: Incremental unsupervised skill discovery for state covering and goal reaching. In *International Conference on Learning Representations*. OpenReview.net, 2022.

S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2019.

M. J. Kearns and U. Vazirani. *An introduction to computational learning theory*. MIT press, 1994.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

S. Kumar, A. Kumar, S. Levine, and C. Finn. One solution is not all you need: Few-shot extrapolation via structured maxent rl. *Advances in Neural Information Processing Systems*, 33:8198–8210, 2020.

T. Kwon. Variational intrinsic control revisited. In *International Conference on Learning Representations*, 2020.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2018.

OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019.

G. Ou and Y. L. Murphey. Multi-class pattern classification using neural networks. *Pattern Recognition*, 40(1):4–18, 2007.

P. Pawara, E. Okafor, M. Groefsema, S. He, L. R. Schomaker, and M. A. Wiering. One-vs-one classification for deep neural networks. *Pattern Recognition*, 108:107528, 2020.

B. Poole, S. Ozair, A. Van Den Oord, A. Alemi, and G. Tucker. On variational bounds of mutual information. In *International Conference on Machine Learning*, pages 5171–5180. PMLR, 2019.

R. Rifkin and A. Klautau. In defense of one-vs-all classification. *The Journal of Machine Learning Research*, 5:101–141, 2004.

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2019.

D. Strouse, K. Baumli, D. Warde-Farley, V. Mnih, and S. S. Hansen. Learning more skills through optimistic exploration. In *International Conference on Learning Representations*, 2021.

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

D. Warde-Farley, T. V. de Wiele, T. D. Kulkarni, C. Ionescu, S. Hansen, and V. Mnih. Unsupervised control through non-parametric discriminative rewards. In *International Conference on Learning Representations*. OpenReview.net, 2019.

T. Zahavy, Y. Schroecker, F. Behbahani, K. Baumli, S. Flennerhag, S. Hou, and S. Singh. Discovering policies with domino: Diversity optimization maintaining near optimality. *arXiv preprint arXiv:2205.13521*, 2022.

S. Zhang and R. S. Sutton. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

Z.-L. Zhang, X.-G. Luo, S. Garcia, J.-F. Tang, and F. Herrera. Exploring the effectiveness of dynamic ensemble selection in the one-versus-one scheme. *Knowledge-Based Systems*, 125:53–63, 2017.

A Appendix: Background

A.1 Reinforcement Learning and Markov Decision Processes

A discounted infinite-horizon MDP is a tuple $\mathcal{M} = (S, A, P, \gamma, r)$ where S denotes the state space, A denotes the action space, P is the transition function, $\gamma \in [0, 1)$ is the discount factor and the reward function r. An agent interacts with the MDP as follows. At any state s, the agent chooses an action a, thereafter it receives a reward $r(s)$ and the MDP transitions to the next state, drawn from the distribution $P(\cdot \mid s, a)$.

A policy $\pi(\cdot)$ maps states to actions and defines our agent's behavior. An optimal policy $\pi^\star(\cdot)$ is the policy that maximizes expectation over cumulative discounted reward: $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t)]$, where the expectation is taken over the state transitions, and possibly the randomness of the agent.

The action-value function associated with a policy π, also known as the Q-function, gives us the value (expected discounted future reward) at state s with the immediate reward under action a, thereafter playing according to π. We also denote $Q^\star$ as the action-value function associated with $\pi^\star$. $Q^\star$ has the property that $\pi^\star(s) \in \arg\max_{a \in A} Q^\star(s, a)$ for all states s.

In reinforcement learning, we typically deal with an MDP whose reward or transition functions are unknown, and we are interested in approximating $\pi^\star$ by repeated interactions with the model. One way of finding the optimal policy $\pi^\star$, is to estimate $Q^\star$ then choose $\pi^\star(s) = \arg\max_{a \in A} Q^\star(s, a)$. In deep Q-Networks [DQN; Mnih et al., 2013], a deep neural network (NN) is used to estimate $Q^\star$. The NN receives an observation space, representing the state and outputs an estimate of $Q(s, a)$, from which $Q^\star$ is chosen and acted upon. The observation space can vary and is an implementation choice.

A.2 All Pairs Classification

An example for the code matrix for $K = 5$ is:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & -1 & -1 \end{pmatrix}.$$

A.3 Mutual Information Maximization

The full objective as defined in Eysenbach et al. [2019] is:

$$\begin{aligned} \mathcal{F}(\theta) &\triangleq \mathcal{I}(\mathcal{S}; \mathcal{Z}) + \mathcal{H}(\mathcal{A}|\mathcal{S}) - \mathcal{I}(\mathcal{A}; \mathcal{Z}|\mathcal{S}) \\ &= (\mathcal{H}[\mathcal{Z}] - \mathcal{H}[\mathcal{Z}|\mathcal{S}]) + \mathcal{H}(\mathcal{A}|\mathcal{S}) - (\mathcal{H}(\mathcal{A}|\mathcal{S}) - \mathcal{H}[\mathcal{A}|\mathcal{S}, \mathcal{Z}]) \\ &= \mathcal{H}[\mathcal{A}|\mathcal{S}, \mathcal{Z}] - \mathcal{H}[\mathcal{Z}|\mathcal{S}] + \mathcal{H}[\mathcal{Z}] \\ &= \mathcal{H}[\mathcal{A}|\mathcal{S}, \mathcal{Z}] + \mathbb{E}_{z \sim p(z), s \sim \pi(z)}[\log p(z|s) - \mathbb{E}_{z \sim p(z)}[\log p(z)]] \qquad (9) \\ &\geq \mathcal{H}[\mathcal{A}|\mathcal{S}, \mathcal{Z}] + \mathbb{E}_{z \sim p(z), s \sim \pi(z)}[\log q_\sigma(z|s) - \log p(z)], \qquad (10) \end{aligned}$$

The last inequality is due to Jensen's Inequality.

B Appendix: Algorithm Details

B.1 Algorithm

Algorithm 1: Skill discovery with APART

1: **input**: discriminator parameters σ, RL parameters θ, code matrix $M_c(z)$, replay buffer, batch size B.
2: **define** dropout weights $W(t) = (t/T)^2$ for $t = 1, \ldots, T$.
3: **while** not converged **do**
4: Act (Algorithm 2).
5: sample batch $(s_b, s'_b, t_b, z_b)_{b=1}^{B}$ from replay buffer.
Discriminator predictor
6: **define** $y^{\text{ap}}(s_b) = \tanh(q_\sigma(s_b))$ ($\tanh$ is applied elementwise) for all b.
RL loss
7: **define** for all b, compute $r_b^{\text{apart}} = \begin{cases} \min_i \{M_c[z_b, i]\, y^{\text{ap}}(s_b)\}, & \text{w.p. } W(t_b). \\ 0, & \text{otherwise.} \end{cases}$
8: **apply** RL algorithm to update θ w.r.t. r^{apart}.
Discriminator loss
9: **set** $\hat{y} = M_c^\top z$.
10: **define** $y^{\text{ap}\prime} = \frac{y^{\text{ap}}+1}{2}, \hat{y}' = \frac{\hat{y}+1}{2}$.
11: **define** binary cross entropy loss for the discriminator:
$J^{\text{ap}}(s, z) = -\frac{1}{K}\sum_{i=1}^{K}(y_i^{ap'} \times \log(\hat{y}'_i) + (1 - y_i^{ap'}) \times \log(1 - \hat{y}'_i))$.
12: Update $q_\sigma(s)$ to minimize J^{ap}
13: **end while**

Algorithm 1 describes the main algorithm loop. Referencing Fig. 1, line 4 refers to rollout collection (Algorithm 2), line 6 refers to computing $y^{\text{ap}}(s_b)$ with the discriminator, line 7 calculates the intrinsic reward (Eq. (6)), lines 8, 12 update policies for the RL and discriminator.

B.2 Rollout Collection

Algorithm 2: Rollout collection

1: **input**: trajectory length T, rollout number to collect N, RL parameters θ, replay buffer.
2: **sample** latent variable from prior $z \sim p(\cdot)$.
3: **obtain** RL policy $\pi_{\text{RL}}(\theta, z)$, initial state s_1 from environment.
4: **for** $n = 1, \ldots, N$ **do**
5: **for** $t = 1, \ldots, T$ **do**
6: Sample action a_t from policy $\pi_{\text{RL}}(\theta, z)$ at state s_t.
7: Sample next state s_{t+1} from environment after playing a_t.
8: Store tuple (s_t, s_{t+1}, t, z) in replay buffer.
9: **end for**
10: **end for**

Algorithm 2 describes rollout collection, a classic RL setup: the environment outputs a state which is used to choose an action by the RL algorithm. In our case, a latent variable is also sampled from the prior, which is also used by the RL algorithm to choose the action. The transition tuples are stored in a replay buffer to be later used in training.

B.3 Softmax Temperature Tuning

$$\text{softmax}(x) = \frac{e^{\beta x_1}}{\sum_{i=1}^{n} e^{\beta x_i}}$$

In this formula, x represents the input vector, n is the number of elements in the vector, and β is the inverse of the temperature parameter. The softmax function exponentiates each element in the vector with β, and then divides it by the sum of all exponentiated elements to obtain the probability distribution. The probabilities sum up to 1, creating a valid probability distribution.

The temperature parameter $\beta = 1/T$ controls the smoothness or sharpness of the resulting probability distribution. A lower β value makes the distribution more uniform, spreading the probability mass more evenly across all possible outcomes. Conversely, a higher β value leads to a more peaked distribution, concentrating the probability mass on a few outcomes with the highest values.

If β is very low, approaching zero, the softmax function will tend to assign similar probabilities to all outcomes. This is because extremely high temperatures cause the exponentiated values in the numerator to become more similar, leading to a more uniform distribution.

On the other hand, as β approaches infinity, the softmax function becomes more deterministic. The outcome(s) with the highest input value(s) will dominate the probability distribution, receiving almost all of the probability mass, while the probabilities assigned to other outcomes become nearly negligible.

B.4 Mutual Information Lower Bound

Here, we construct a lower bound on the mutual information with our AP minimum reward.

Similar to other mutual-information algorithms, maximizing the lower bound on the mutual information leads to maximizing the mutual information itself Agakov [2004]. Therefore, we are inspired by MI maximization methods, but do not maximize MI explicitly.

Using Eq. (10) and Eq. (6), we replace $q(z \mid s)$ in Eq. (10) with our own discriminator choice, $q'_\sigma(z \mid s)$:

$$\begin{aligned}
\mathcal{F}(\theta) &\geq \mathcal{H}[\mathcal{A}|\mathcal{S},\mathcal{Z}] + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}[\log q_\sigma(z|s) - \log p(z)] \\
&\sim \mathcal{H}[\mathcal{A}|\mathcal{S},\mathcal{Z}] + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}[\log q'_\sigma(z|s) - \log p(z)] \\
&= \mathcal{H}[\mathcal{A}|\mathcal{S},\mathcal{Z}] + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}[\log e^{\min_i\{y'_z(s)[i]\}} - \log p(z)] \\
&= \mathcal{H}[\mathcal{A}|\mathcal{S},\mathcal{Z}] + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}[\min_i\{y'_z(s)[i]\} - \log p(z)]
\end{aligned}$$

For reward shaping, we replace $q(z \mid s)$ with $q'_\sigma(z \mid s)$. since it gets the semantics of a probability from the training loss, which is the binary cross entropy.

Next, we derive a lower bound on the mutual information using our intrinsic reward. First, we define a softmax on the minimum latent variables as:

$$\mathrm{softmax}(q'_\sigma)(z \mid s) = \frac{\exp(q'(z \mid s))}{\sum_{z'} \exp(q'(z' \mid s))}. \tag{11}$$

We begin from Eq. (9). Our first step is to replace the probability $p(z \mid s)$ with the softmax in Eq. (11).

$$\begin{aligned}
\mathcal{F}(\theta) &\geq \mathcal{H}[\mathcal{A}|\mathcal{S},\mathcal{Z}] + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}[\log(\mathrm{softmax}(q'_\sigma)(z \mid s)) - \mathbb{E}_{z\sim p(z)}[\log p(z)]] && (12)\\
&= \mathcal{H}[\mathcal{A}|\mathcal{S},\mathcal{Z}] + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}\left[q'(z \mid s) - \log\left(\sum_{z'} e^{q'_\sigma(z'|s)}\right) - \mathbb{E}_{z\sim p(z)}[\log p(z)]\right] && (13)\\
&\geq \mathcal{H}[\mathcal{A}|\mathcal{S},\mathcal{Z}] + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}[q'(z \mid s) - \log(Ke) - \mathbb{E}_{z\sim p(z)}[\log p(z)]] && (14)\\
&= \mathcal{H}[\mathcal{A}|\mathcal{S},\mathcal{Z}] + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}[\min_i\{M_c[z,i]\, y_i^{ap}[i]\} - \log(Ke) - \mathbb{E}_{z\sim p(z)}[\log p(z)]]. && (15)
\end{aligned}$$

Eq. (12): comes from Eq. (9), with our own discriminator choice. Eq. (13): breaking apart nominator and denominator from Eq. (11). Eq. (14): since $q'_\sigma(z' \mid s)$ is bound by 1, and we sum over all latent variables, K will be an upper bound, where K is the number of latent variables, similarly to eq. (9) in [Poole et al., 2019]. Eq. (15): using Eq. (6)'s full term

$\log p(z)$ is constant in the uniform distribution case and is no longer needed, since it is independent of the skills and discriminator. its purpose was to normalize the intrinsic reward range to have a zero-mean. In our case, we already achieve a zero mean by using $y_z(s)$ that is in the range $(-1, 1)$, so we can exclude it. We are left with $\mathcal{H}[\mathcal{A} \mid \mathcal{S}, \mathcal{Z}] + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}[r_{apart}]$, which is a lower bound on the mutual information and which we optimize with an RL algorithm.

Table 1: Hyperparameters

Hyperparameter	Value
Torso	1 MLP layer
Number of actors	1
Batch size	640
Skill trajectory length	40
Number of latents	100
Replay buffer size	50,000
Optimizer	ADAM
Learning rate	2e-3
RL algorithm	DQN
ϵ	0.001
Discount γ	0.99

C Appendix: Setup

C.1 Learning

To run experiments, we chose DQN [Mnih et al., 2013] as our RL algorithm. Similar to Strouse et al. [2021], we act with an epsilon greedy policy which approximates the entropy term.

We use a single fully connected layer for the discriminator backbone and the RL backbone. This is similar to using a lookup table in terms of parameter number needed, but the update method of the parameters is gradient based. Our observation space is a one hot vector containing only the location of the agent for the discriminator, and the location with the latent variable information for the RL. Additional implementation choices are explained in D. Additional choices for parameters and their effect on learning can be found in Appendix D.

C.2 Hyperparameters

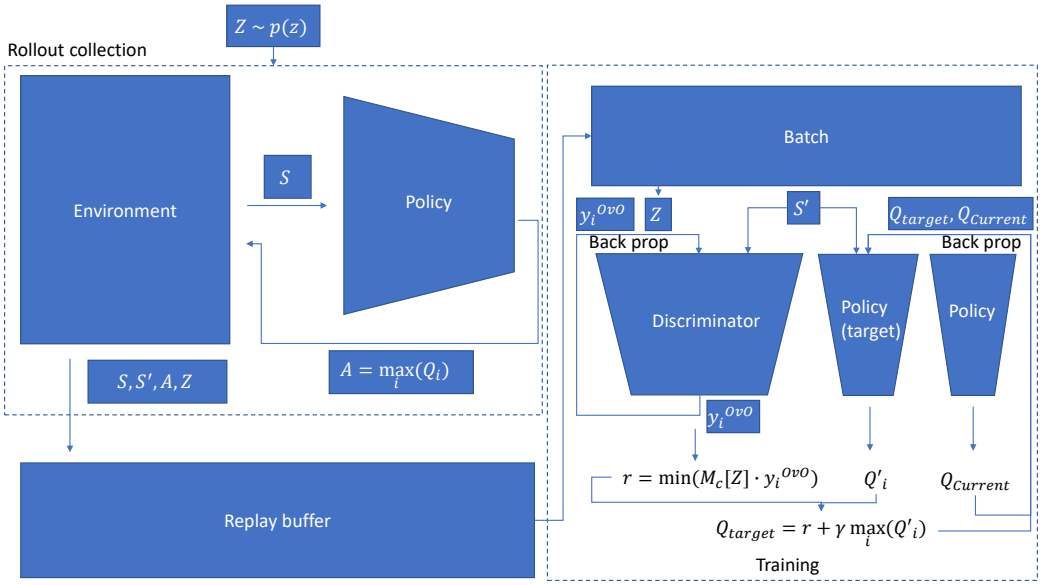


Figure 9: Detailed algorithm implementation

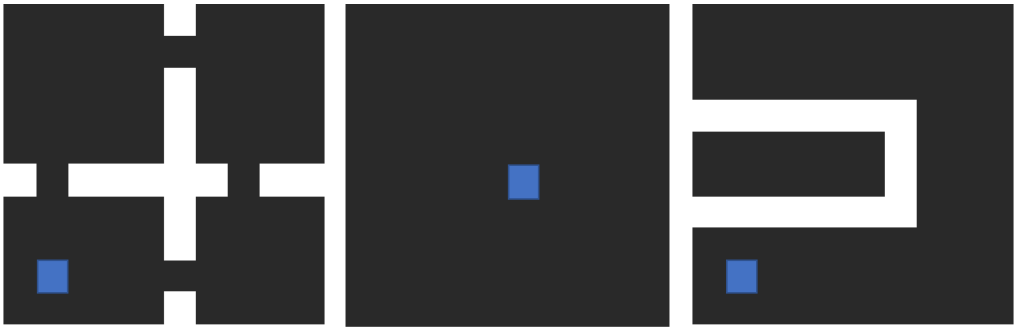
C.3 Grid World Environments

We use three different grid world environments, each with five actions to choose from: {Stay, Left, Right, Up, Down}. Although 20 steps are enough to explore almost all states, we show in Fig. 14 that learning improves with more environment steps. An agent has more time to explore and is rewarded longer in the final locations. However, longer episodes are also associated with longer training durations, so we choose an environment step length of 40 steps for $T = 40$.

Four rooms. 10x10 grid separated into four rooms. Each room is connected to the two closest rooms, but the passage is only 1 pixel, forcing many skills to occupy the same state. There are 85 states, the agent's initial position is one step left and one step down from the bottom left corner [Sutton et al., 1999, Strouse et al., 2021, Gregor et al., 2017, Fig. 10a].

Empty grid. An empty 10x10 grid. There are 100 states, the agent's initial position is in the middle of the grid (Fig. 10b).

U-maze. 10x10 grid, where the whole trajectory is a narrow corridor (3 pixels wide). To reach all states, multiple skills need to advance through the narrow corridor. There are 72 states, the agent's initial position is one step left and one step down from the bottom left corner [Kamienny et al., 2022, Fig. 10c].



(a) Four Rooms (b) Empty (c) U-maze

Figure 10: **Grid worlds examples, including initial agent location:** (a) Four Rooms (b) Empty (c) U-maze.

C.4 Metrics

We measure each algorithm by counting the effective number of skills, i.e. number of occupied last states.

$$n_{\text{effective}} \triangleq unique(\cup_{i=1}^{K} S_f(i)) \tag{16}$$

Where $S_f(i)$ is the last state of skill i, and we count over all latent variables the number of unique last states. This measure is interpretable as a transformation of mutual information (see [Gregor et al., 2017, Eysenbach et al., 2019, Strouse et al., 2021]) therefore the effective number of skills is an effective and important measure.

We measure metrics with two configurations: 5M environment steps and 150M environment steps. On the 5M benchmark, we test for sampling efficiency. In our setting, most of the convergence is in the first million of steps and this is a good indicator to the rest of the training. Furthermore, running a training for hundreds of millions of steps will take the average user weeks and running for 5M steps will usually only take less than one day on a regular computer. On the 150M benchmark, we test for full skill acquisition, meaning whether $n_{\text{effective}} = n_{\text{available}}$, where $n_{\text{available}}$ represents all of the available states (effectively, grid size without walls)

D Appendix: Implementation Choices

In this section, we review implementation choices and their effect on learning. This is done by fixing all parameters except for one, and choosing 3-4 options for the tested parameter. If the value is

numerical, desirably, we'll choose at least one option smaller than our chosen hyperparameter and one larger. If the value is categorical, desirably we'll want to show a few different choices and explain their similar and different characteristics. All graphs in this section are on Rooms grid environment. For example, we will discuss what batch size is best out of four options, learning rates, number of environment steps, etc.

Number of NN layer weights. Our setup uses only one fully connected layer as our backbone. Therefore, number of parameters used (weights) is the number of inputs times the number of outputs (plus bias terms). The number of network outputs is $L = K(K-1)/2$ for AP but only K for OvA, resulting in a significantly larger number of weights for AP. We therefore want to test the effect of the amount of weights between the two methods. For this, we added another fully connected layer to the OvA method with 2300 weights (to get a similar number of parameters between methods). We calculate number of weights as follows: rooms environment has 85 available states and we use 100 latent variables. Therefore, for OvA, we have 85 inputs and 100 outputs. The regular setup will result in about 85x100=8500 parameters, while after adding a hidden layer we will have 85x2300+2300x100=425,500 parameters. AP has same number of inputs (85) but uses $K(K-1)/2$ outputs and therefore uses about 85x100x99/2=420,750 weights. Furthermore, adding another layer means adding a non-linearity which is helpful for learning. In Fig. 11 we show that adding a hidden layer of size 2300 does not lead to a significant change in learning, indicating that it is not the number of weights constraining the learning for OvA.

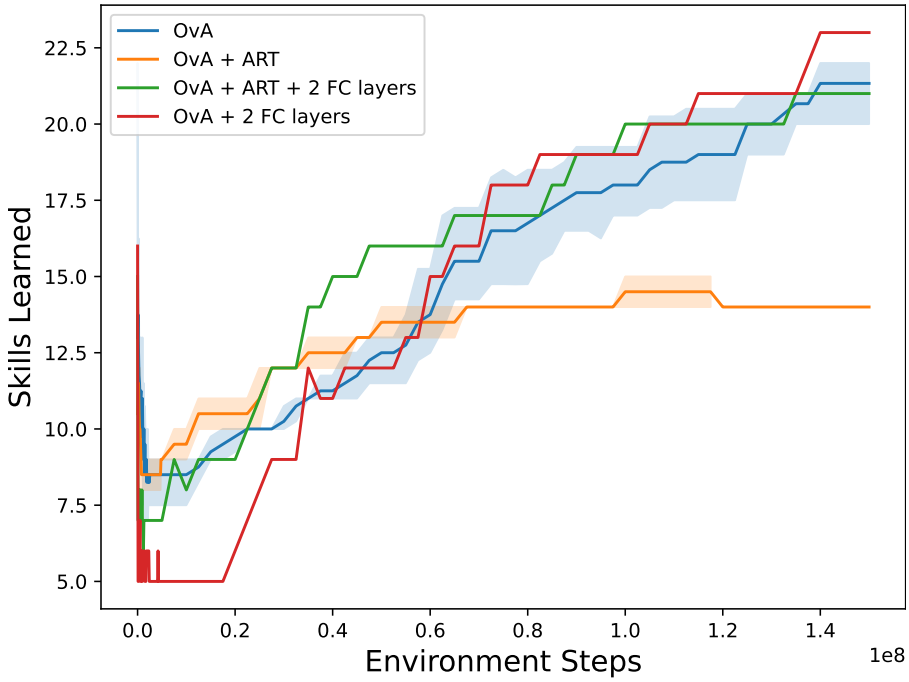


Figure 11: One vs All with two FC layers vs one. ART stands for ascending reward and dropout.

Batch size. The number of samples taken from the replay buffer in each iteration of APART (Line 5 in Algorithm 1) has an effect on both smoothness of the training and compute resources. While the batch size is bounded by the size of the GPU memory (if using a GPU), a larger batch size also increases training time and can slow down convergence. A smaller batch size, on the other hand, can lead to unstable learning. We test different values of this hyperparameter in multiples of environment steps. Although we sample randomly from the replay buffer, not necessarily taking entire rollouts, we still want to keep an explainable multiple of the environment steps for the batch size, meaning we test our batch sizes as multiples of number of chosen environment steps (40). In Fig. 12, we see the effect of different choices, including 16 (choice from [Strouse et al., 2021]), 320 (8 times rollout length), 640 (16 times rollout length, this is our chosen parameters, in order to be able to run multiple runs simultaneously) and 1280 (32 times rollout length) on the number of learned skills.

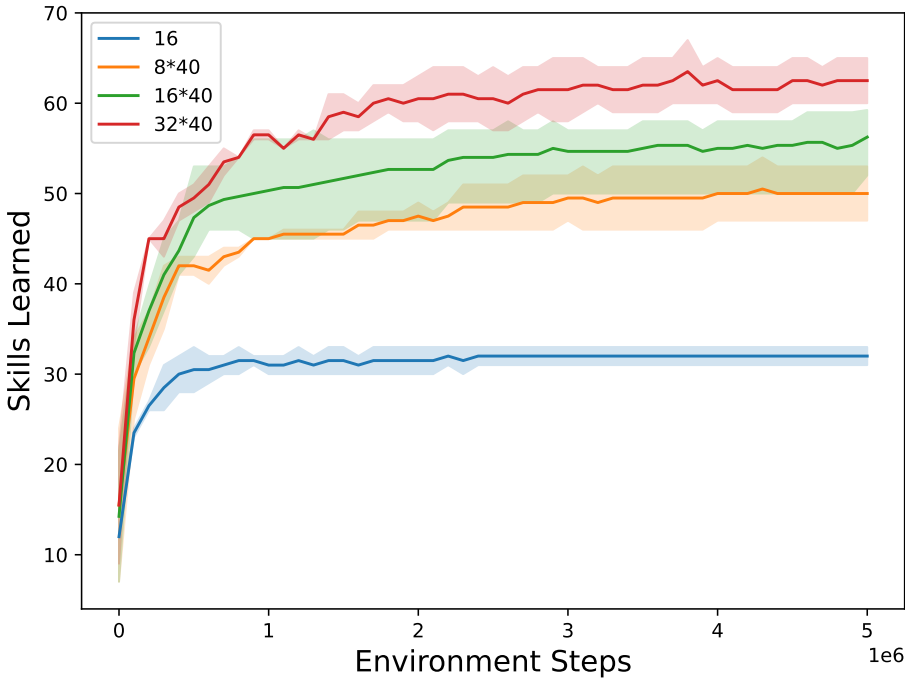


Figure 12: different batch sizes

Learning rate. Our algorithm trains two NN's simultaneously. One is trained using the DQN algorithm and the other implements the discriminator. The learning rate of these two NNs are tightly coupled since the reward for the RL is derived from the discriminator outputs, and the state from the RL algorithm is sampled from the policy chosen by the DQN. If the discriminator were to learn too slowly, the reward will be far from the true estimation. If the RL algorithm were to learn too slowly, its policy won't be meaningful for the discriminator to learn from. In Fig. 13 we can see the effect of different learning rates on learning, our chosen learning rate is $2\mathrm{E}{-3}$ for discriminator and $2\mathrm{E}{-3}$ for the RL algorithm.

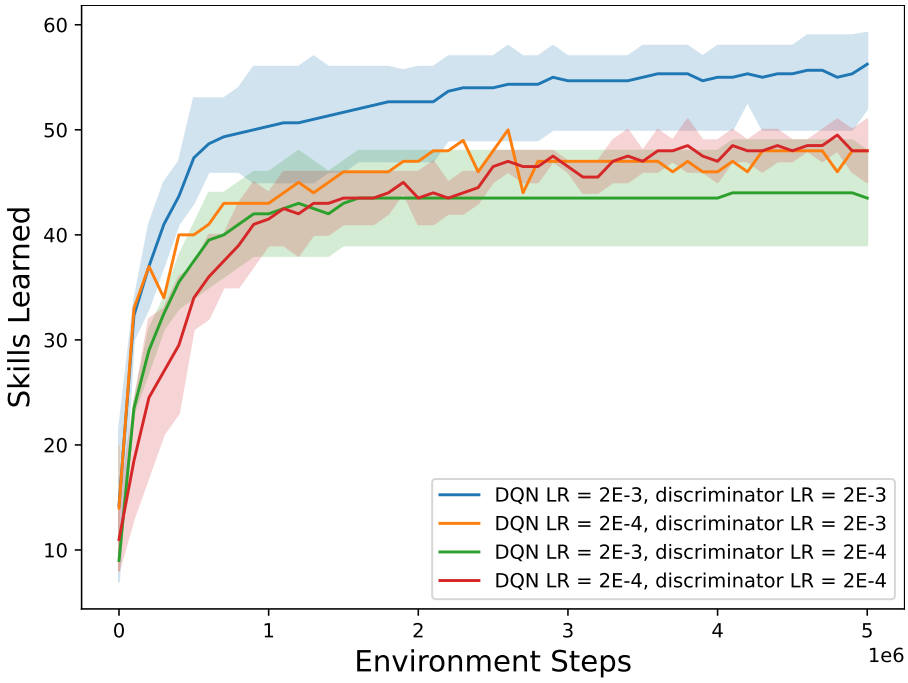


Figure 13: different learning rates

Environment step length. Our environments don't have a "natural" end to them (collisions for example), thus the only termination point is determined by the environment step length. Although 20 environment steps are enough to reach all possible states, additional allowed steps mean more possible paths to the same state. Moreover, staying on the last state for a longer period of time enables more collected transitions in which the state and skill are coupled (the same latent variable leads to the same state), therefore strengthening the link between them. On the other hand, we do need to terminate the episode at some point in order to keep collecting more episodes. In Fig. 14, we can see a few different choices for environment step lengths. Strouse et al. [2021] use 20 environment steps, yet we observe that this is not an optimal choice. Our choice of 40 environment steps is also not an optimal choice, and in hindsight we might have chosen 50 environment steps for our experiments, but because of limited time and resources we used 40.

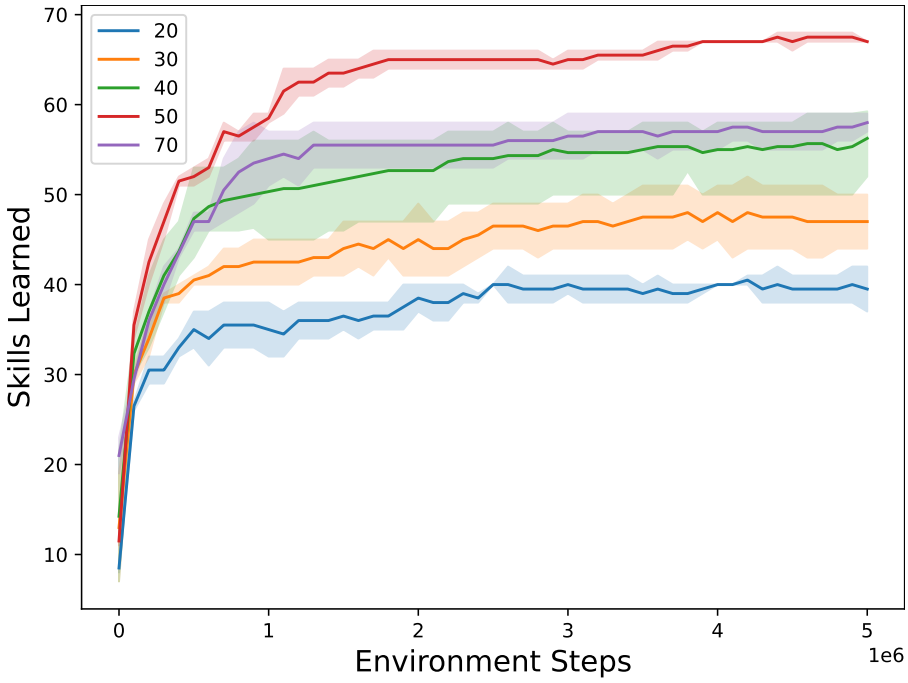


Figure 14: different env steps

Replay buffer size. The replay buffer holds transitions (s, s', a, z) from rollouts, and is sampled randomly during training. When collecting new transitions, the oldest transitions are replaced by the new sampled rollouts. The larger the buffer, the larger the transition history is and the more offline the learning is, meaning the learning is further away from the current policy, since older samples that use older policies are used. On the other hand, a small buffer can result in instability during training since the effect of each collected rollout will be larger and won't be smoothed out by the effect of the older transitions [Zhang and Sutton, 2017]. In Fig. 15, we can see the effect of the buffer size in our settings. We can see that the gap between a 10K buffer and a 50K buffer is small whereas the gap between 50K and 100K is significant. Our choice for this hyperparameter is 50K as it shows the best empirical results.

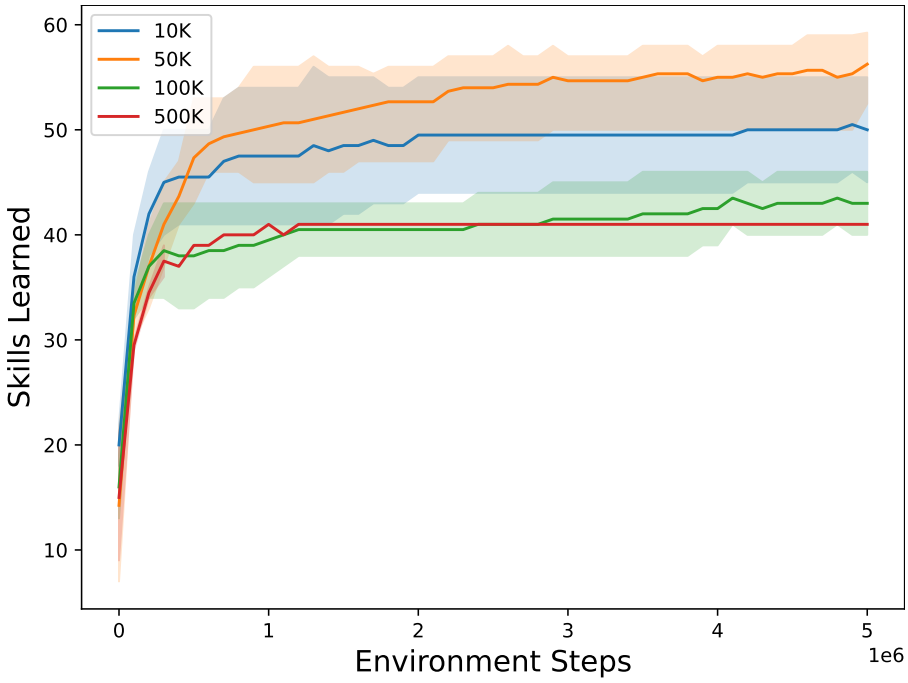


Figure 15: different buffer sizes

Number of latents. We experiment with different number of latents and see their effect on learning. For each rollout, we select a prior out of a distribution size, which is the number of latents. For example, if we use 100 latents, our prior is chosen as a number between 0 and 99. Since our metric for diversity is the number of unique last states Eq. (16), and there is a constant and known number of last states per environment, we would like to pick number of latents which is close to the number of possible states. In Fig. 16, we explore different choices, including our choice which is 100 latents, but also a much smaller choice of 10, a choice of 50 and a larger choice of 200.

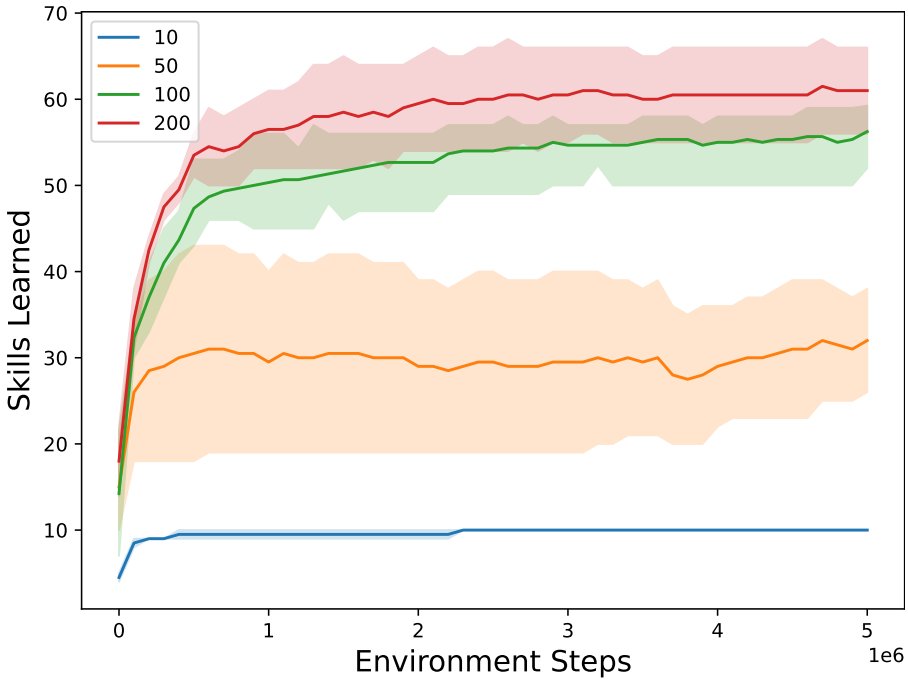


Figure 16: Number of learned skills vs different number of latent variables

Ascending reward function. The ascending function Eq. (7) in our algorithm is used to add weights to emphasize the reward as the step number increases. We want the final steps to have a larger weight than the initial steps, since in the initial steps the skills are tightly packed and harder to discriminate (since all agents have the same initial position). Therefore, it is important to choose an ascending function. In Fig. 17, we pick a few options (for the positive x values corresponding to step number): $W(t) = t, W(t) = t^2, W(t) = t^4, W(t) = e^{5t-5}$. We show that the exact choice of an ascending function for the weights is not as important, since their performance is similar.

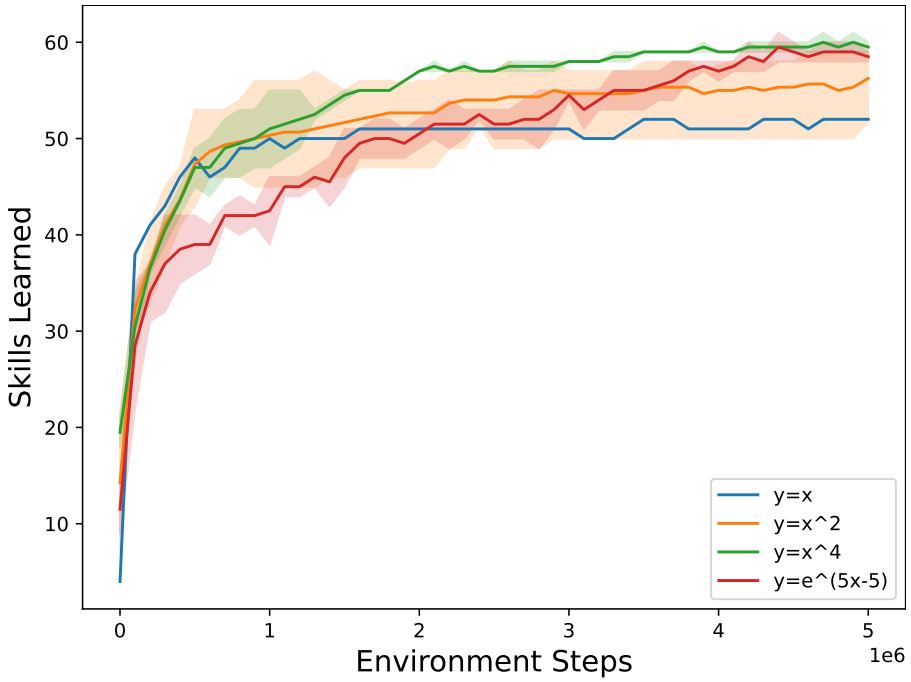


Figure 17: **Different weight functions over 5 million environment steps in rooms.** Blue: linear function, orange: power of 4, green: $\exp(5t-5)$, red: squared function (APART).

Using "don't cares" in AP discriminator. When using AP classification, Section 2.2, we use pair comparisons between each two classes i, j, resulting in $K(K-1)/2$ comparisons. The "don't cares" are the pair comparisons from the AP classification that are indifferent to the result. For example, if the true class is 0 and the pair comparison is between 1 and 2, the prediction can ignore the classifier that classifies between 1 and 2. One option (used in Pawara et al. [2020]) is to use this information in the discriminator's loss, with a value of 0 (or 0.5 when scaling outputs to the range (0, 1)). In Fig. 18 we revisited this choice and compare it with only backpropagating on meaningful pair comparisons (masking out all "don't care" comparisons). Meaning, discriminator loss is only given $K-1$ meaningful comparisons, instead of $K(K-1)/2$ comparisons which hold some meaningful comparisons but also "don't cares". We found that in our setting, it is better to mask out the "don't cares". Another advantage of masking don't cares is backpropagating over only $K-1$ values instead of $L = K(K-1)/2$ values.

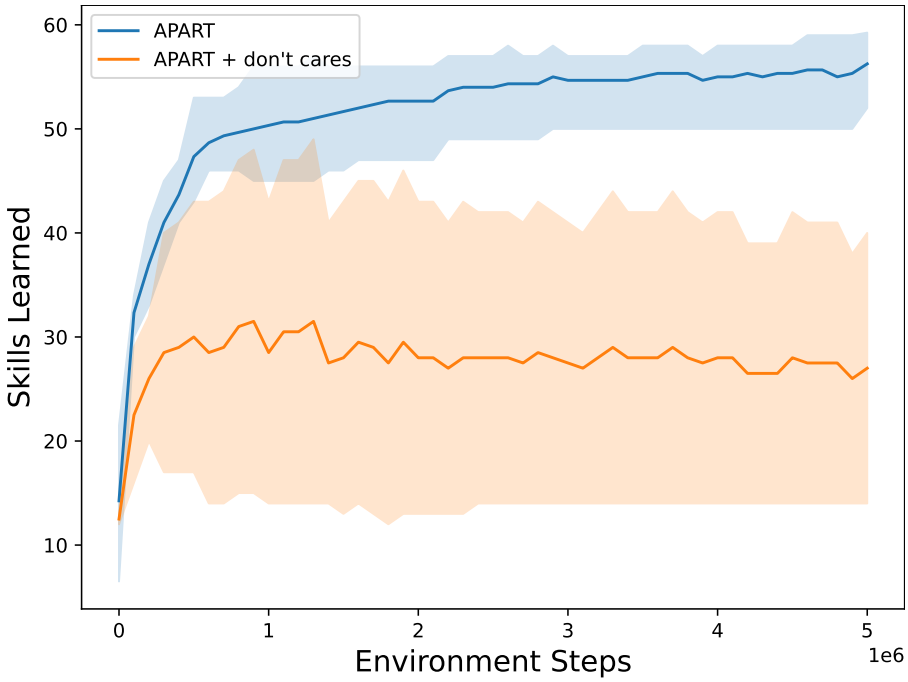


Figure 18: Adding "don't cares" to discriminator

Using different observation spaces. As explained in Appendix A.1, the observation space is an implementation choice. We initially chose to use a one-hot vector observation Appendix C.1 but other options are also available. In order to further investigate our method, and in preparation to run on more complex environments (Atari [Bellemare et al., 2013] for example), we want to see its performance with a pixel-based observation space, meaning the observation space is an image containing all grid information, as a human player would observe when playing the game. To accommodate a pixel-based observation space, we also change our backbones to CNNs [Krizhevsky et al., 2017], as is the regular choice for image-based observation spaces, and adjust our learning rates accordingly. We do these changes for both the discriminator and the RL. The backbone used has 3 CNN layers with feature sizes of (32, 64, 128), followed by an FC layer. We also use padding of 2, considering we don't want to lose any of our input pixels, and stride of 1. We don't use any pooling for the same reason. The learning rates we use are $1\text{E}{-5}$ for the RL algorithm and $3\text{E}{-4}$ for the discriminator. The rest of the parameters remain unchanged. We can see in Fig. 19 that we are able to achieve good learning with this observation space as well, paving the way to experimenting on more complex environments.

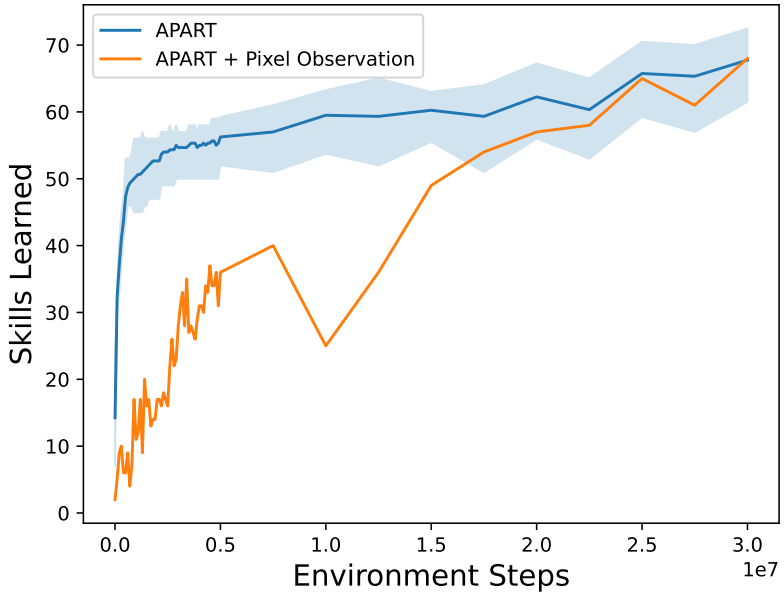


Figure 19: Pixel observation space

E Appendix: Additional Results

Reward Comparison. Reward comparison of all methods found in Fig. 7, with the addition of VIC. It is evident that VIC stands out due to its significantly larger scale and distinctiveness compared to other methods.

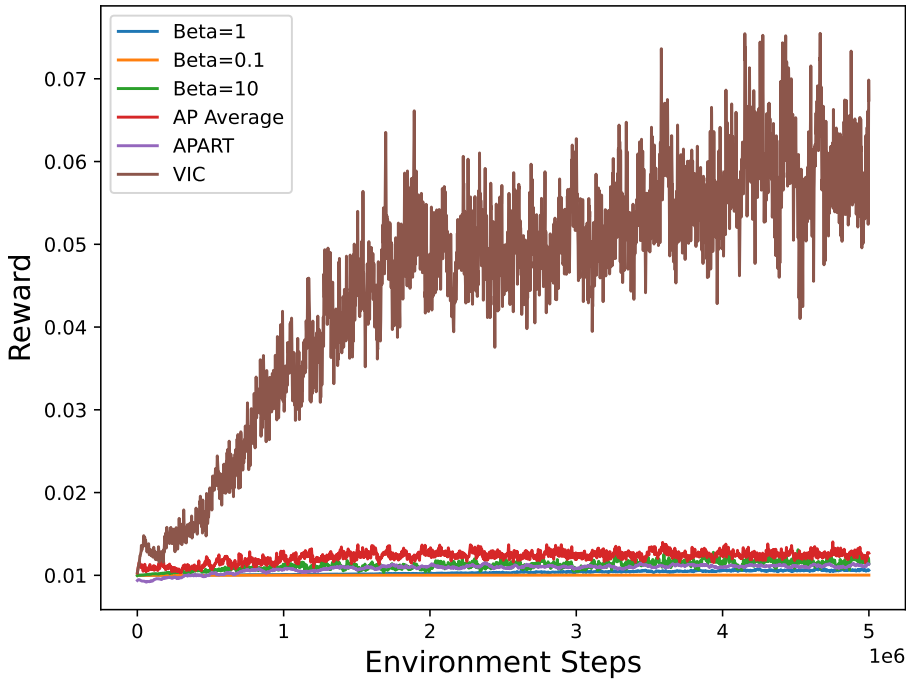
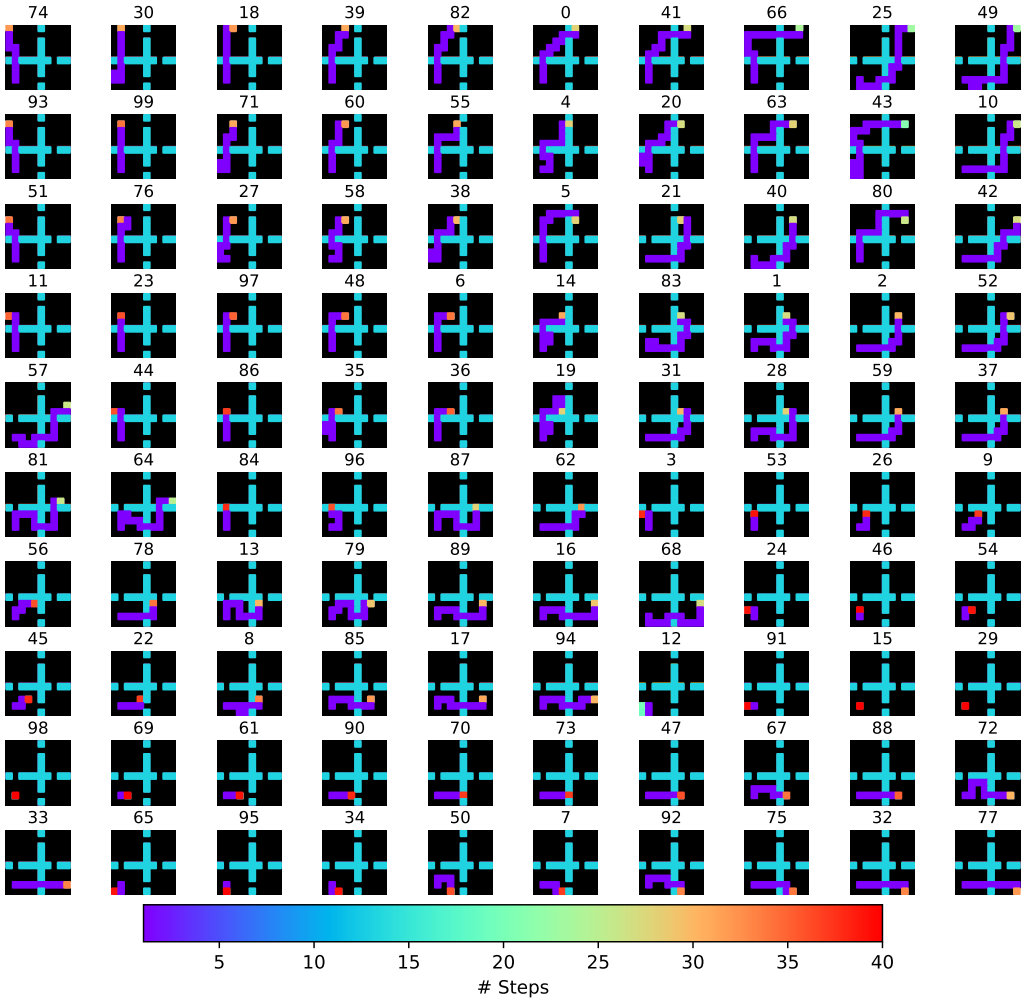
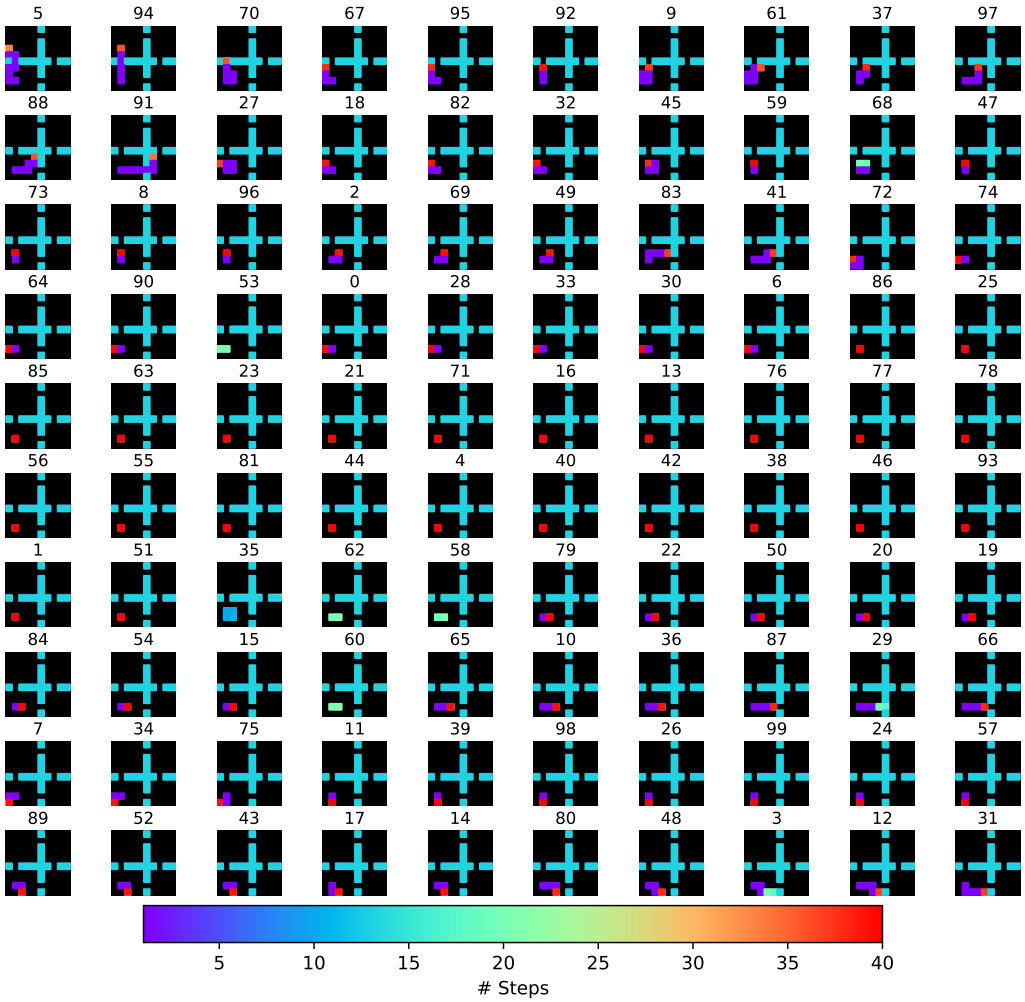


Figure 20: Comparison of the learned rewards from six different methods: VIC, APART and AP Average, Tuned VIC with: $\beta = 10$, $\beta = 1$ and $\beta = 0.1$. Rewards are smoothed with $\sigma = 5$.

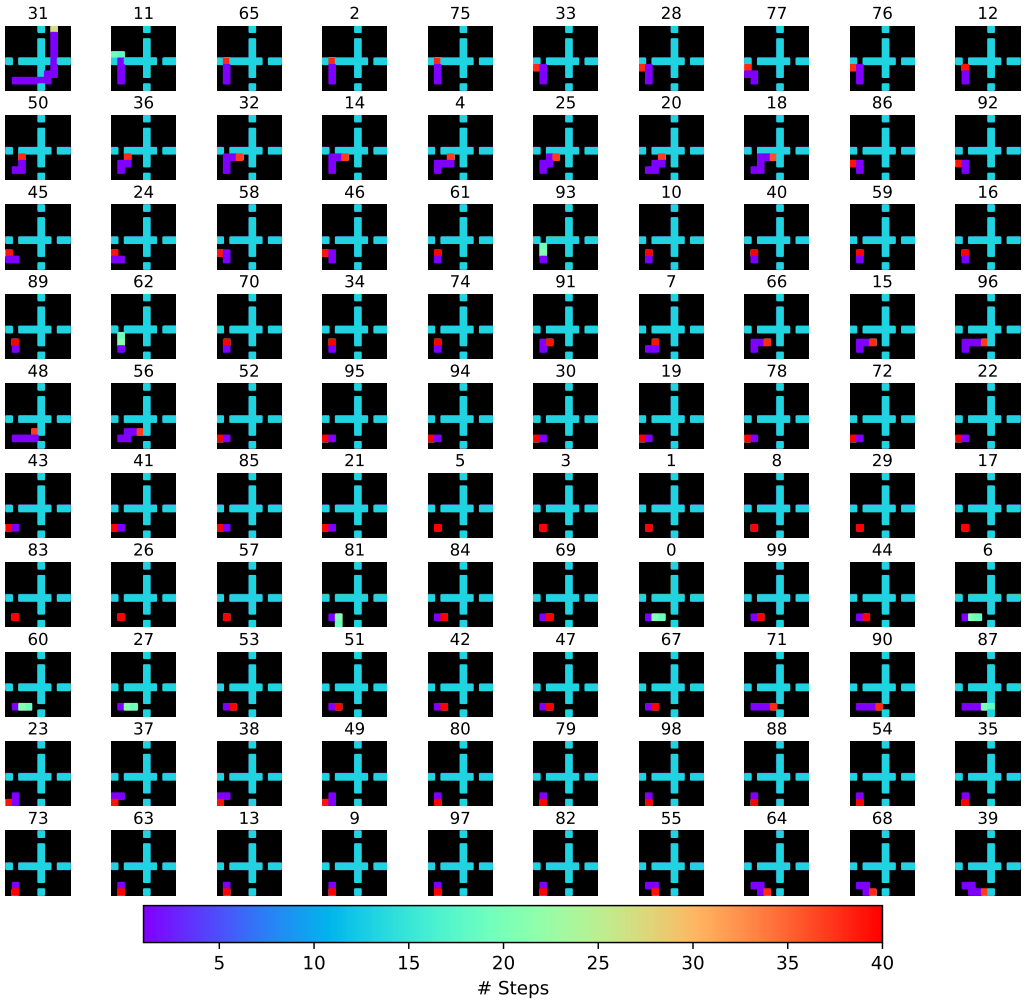
Learned Skills Up until now, we only talked about number of learned skills without providing additional information on the skills learned. In Appendix E, we show all learned skills during the training process, on different environments and for different methods. Skills are sorted by last state visited and are numbered by their latent ID. It is interesting to note that not all states are reached in the fastest way. Another observation is that since our latent number is larger than the possible states number, we will always have a few overlapping skills. We can also see the skills evolving through time in different algorithms.



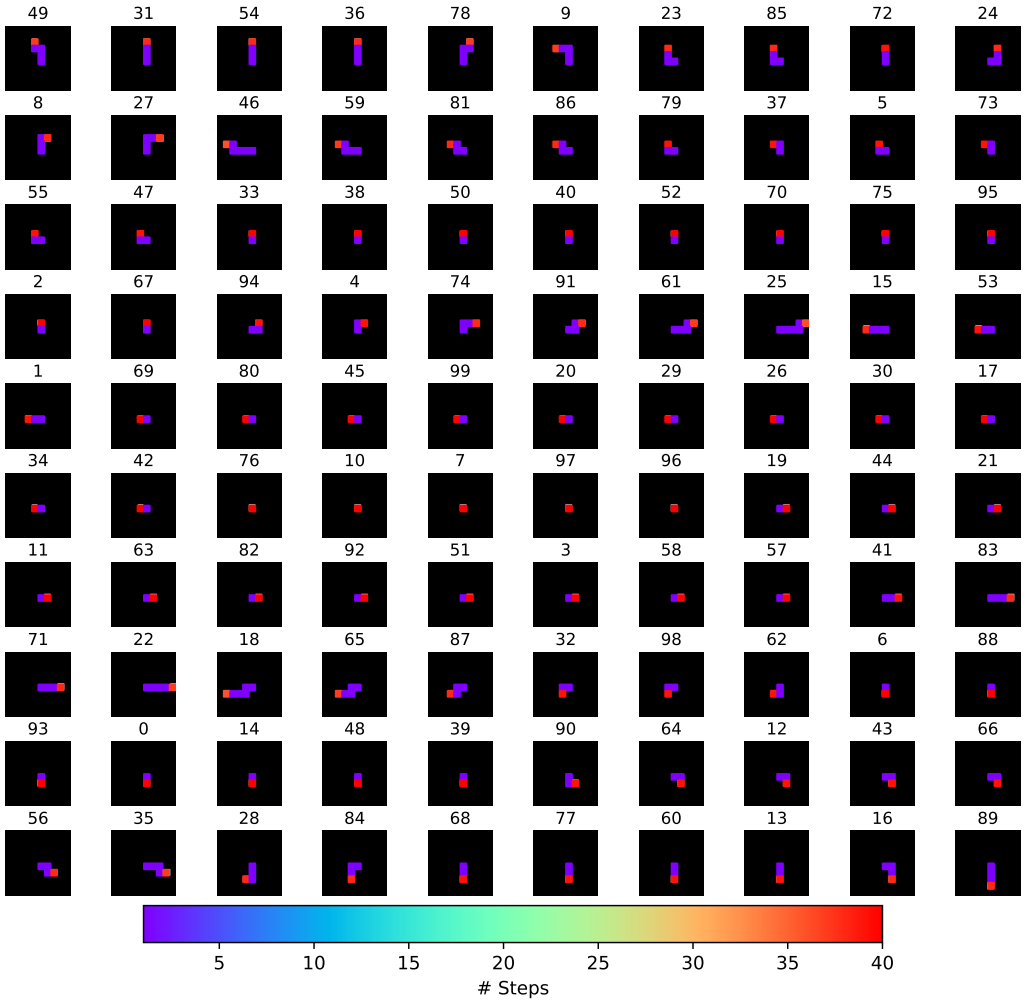
(a) Rooms, APART


5 94 70 67 95 92 9 61 37 97
88 91 27 18 82 32 45 59 68 47
73 8 96 2 69 49 83 41 72 74
64 90 53 0 28 33 30 6 86 25
85 63 23 21 71 16 13 76 77 78
56 55 81 44 4 40 42 38 46 93
1 51 35 62 58 79 22 50 20 19
84 54 15 60 65 10 36 87 29 66
7 34 75 11 39 98 26 99 24 57
89 52 43 17 14 80 48 3 12 31
5 10 15 20 25 30 35 40
Steps

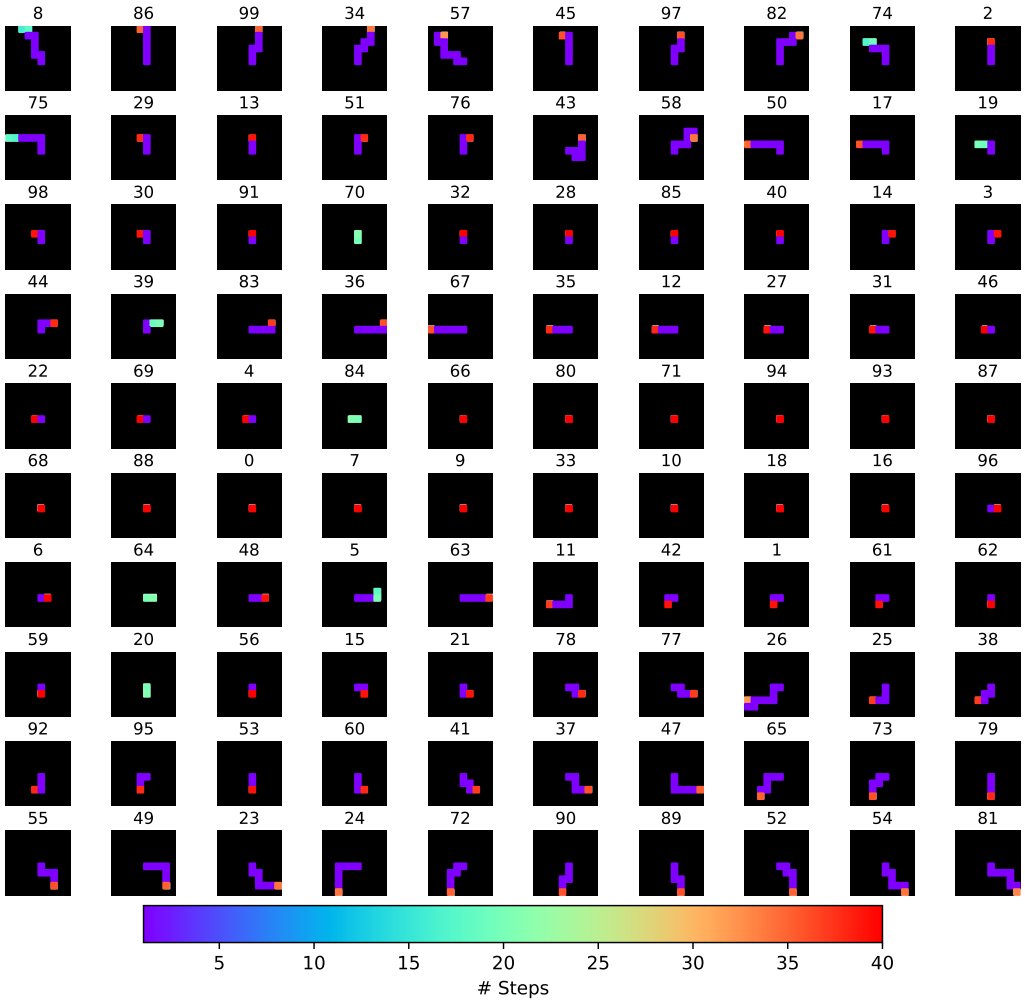
(b) Rooms, VIC



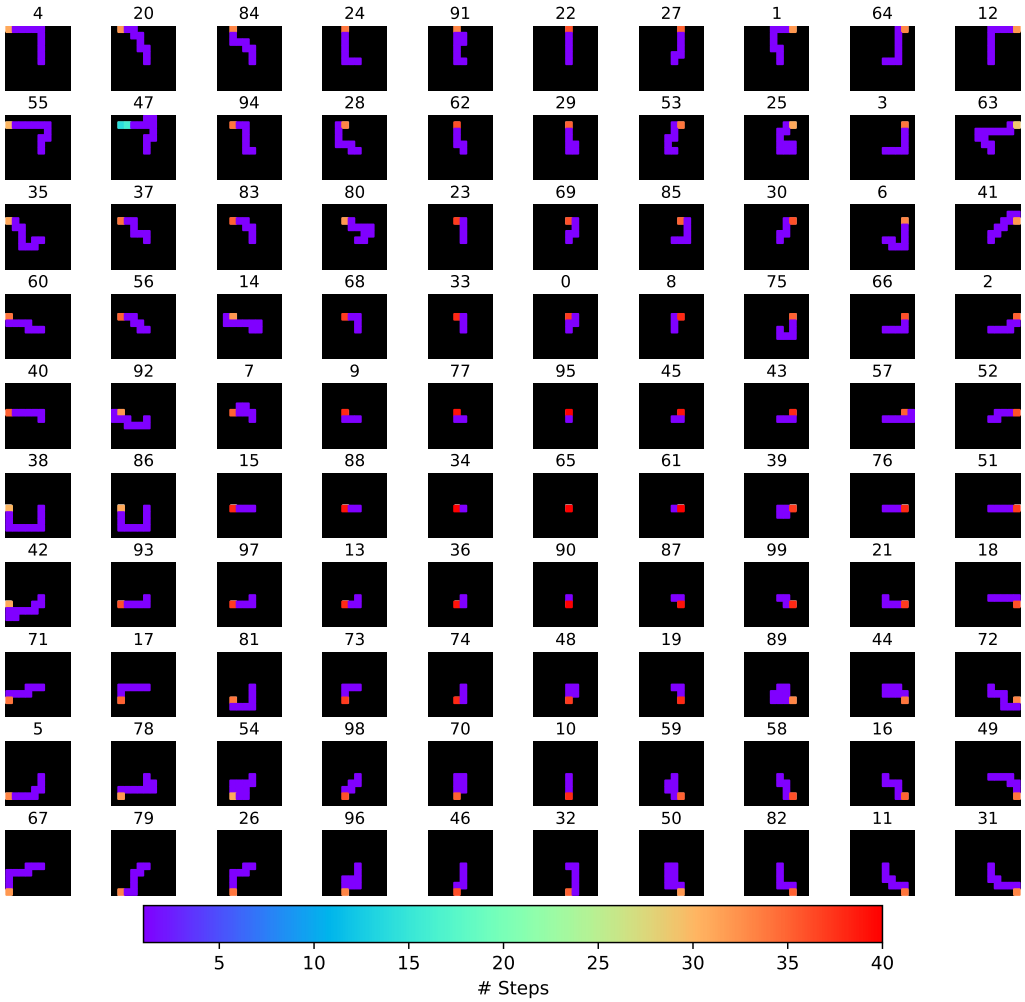
(c) Rooms, DIAYN



(d) Empty, DIAYN


8 86 99 34 57 45 97 82 74 2
75 29 13 51 76 43 58 50 17 19
98 30 91 70 32 28 85 40 14 3
44 39 83 36 67 35 12 27 31 46
22 69 4 84 66 80 71 94 93 87
68 88 0 7 9 33 10 18 16 96
6 64 48 5 63 11 42 1 61 62
59 20 56 15 21 78 77 26 25 38
92 95 53 60 41 37 47 65 73 79
55 49 23 24 72 90 89 52 54 81
5 10 15 20 25 30 35 40
Steps

(e) Empty, VIC



(f) Empty, APART

Figure 21: **All skills after convergence of algorithm**, skills are ordered by last state.