Learning World Graphs to Accelerate Hierarchical Reinforcement Learning

Notations

We summarize the symbolic notations used by the paper in the following table:

$a \in \mathcal{A}$	The finite action space
$s \in \mathcal{S}$	The finite state space
$\mathcal{V}$	The candidate set of states that can be selected by Manager as wide goal
$\mathcal{V}_p$	The set of pivotal states
$\mathcal{V}_{\text{all}}$	The set of all accessible states
$\mathcal{V}_{\text{rand}}$	The set of uniformly sampled states
s_p	a pivotal state
V	variational inference model
z	The binary latent variable in V
τ	A state-action trajectory
π_g	(curiosity-driven) goal-conditioned policy
π^m	Manager policy
π^ω	Manager wide goal policy
g_w	Manager wide goal
s_w	Local $N \times N$ area surrounding g_w
π^n	Manager narrow goal policy
g_n	Manager narrow goal
π^w	Worker policy

Mazes with Learned Pivotal States and Tasks

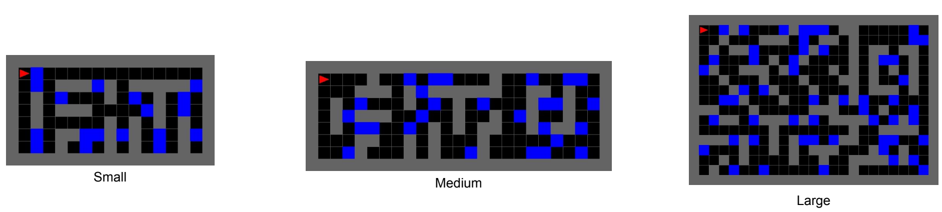
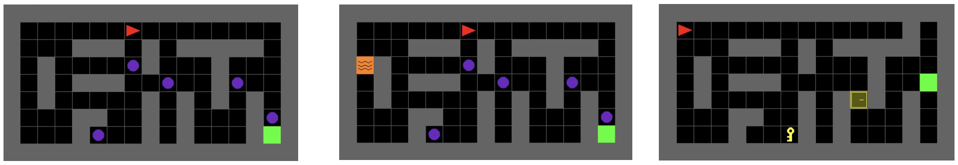


Figure S1: Visualization of 3 mazes in our experiments and along with the learned pivotal states.

Preprint. Under review.



MultiGoal MultiGoal-Stochastic Door-Key

Figure S2: Visualization of tasks in our experiments.

Bird-View Matrix Representation

Object Type	Values
Agent	$(0, 1, 0)$
Wall	$(1, 0, 1)$
Purple Ball	
Exit	$(0, 0, 1)$ in *MultiGoal*; $(0, 0.2, 0.85)$ in *Door-Key*
Door	$(0, 0.7, 0.7)$
Agent+Key	$(0, 0, 0.5)$
Key	$(0, 0, 1)$
Lava	$(-1, 0, 0)$

World Graph Discovery

Evidence Lower Bound Derivation

$$\begin{aligned}
\log p(a|s) &= \log \int p(a|s,z)dz \\
&= \log \int p(a|s,z)p(z|s)\frac{q(z|a,s)}{q(z|a,s)}dz \\
&= \log \int p(a|s,z)\frac{p(z|s)}{q(z|a,s)}q(z|a,s)dz \\
&\geq \mathbb{E}_{q(z|a,s)}[\log p(a|s,z) - \log \frac{q(z|a,s)}{p(z|s)}] \\
&= \mathbb{E}_{q(z|a,s)}[\log p(a|s,z)] + D_{\mathrm{KL}}(q(z|a,s)||p(z|s))
\end{aligned}$$

Overall Pivotal State + Goal-Conditioned Policy Learning Algorithm

Initialize network parameters for the recurrent variational inference model V;
Initialize network parameters θ, for π_g;
Initialize $\mathcal{V}_p$ with the initial position of the agent, i.e. $\mathcal{V}_p = \{s_0 = (1,1)\}$;
Initialize a dictionary $P = \{(1,1) : []\}$ to record the action trajectory taken from the origin to each $s_p \in \mathcal{V}_p$;
while *reconstruction error rate e_r has not yet reached plateau* **do**
  Randomly select N $s_p \in \mathcal{V}_p$ with replacement;
  Perform T-step rollout following random walk policy from each selected s_p (use P to reach), i.e. $\tau_{r,n} = \{(s_{0,n} = s_p, a_{0,n}), \cdots (s_{T,n}, a_{T,n})\}, n = 1 \cdots N$;
  Update π_g with a N starting-goal state pairs $(s_{0,n}, g_n)$ while setting each starting state $s_{0,n}$ (use P to reach), each goal state $g_n = s_{T,n}$ from $\tau_{r,n}$;
  Store T-step observed rollout when training π_g, $\tau_{\pi,n} = \{(s^\pi_{0,n} = s_p, a^\pi_{0,n}), \cdots (s^\pi_{T,n}, a^\pi_{T,n})\}$;
  Update V with τ_r's and τ_π's and;
  Update $\mathcal{V}_p$ from the update V based on the prior mean and ;
  Use τ_r's and τ_π's, the updated $\mathcal{V}_p$ and the current P to update P.;
end
Output $\mathcal{V}_p$ and π_θ.;

Algorithm 1: Overall Algorithm for Pivotal State Identification + π_g learning

Algorithm for π_g Learning

Initialize starting position z_0 and goal positions g;
Initialize network parameters θ for $\pi_{g,0}$, here $\pi_{g,t}$ refers to the policy at time rollout time step t;
for iter $= 0, 1, 2, \cdots$ **do**
  Clear gradients $d\theta \leftarrow 0$;
  Simulate under current policy $\pi_{g,t-1}$ until $t_{\max}$ steps are obtained, where, $z_t = f_{\text{LSTM}}(\text{CNN}(s_t), h_{t-1})$, $V_t = f_v(z_t)$, $\pi_t = f_p(z_t)$, $t = 1, \cdots t_{\max}$;
  Feed the rollout trajectory to V and obtain reconstruction error rate e_t for each time step ;
  Reward at time t becomes $r_t = \mathbb{1}_{s_t = g} + e_t$ (whether it reaches the goal + intrinsic reward);
  $R = \begin{cases} 0, \text{ if terminal} \\ V_{t_{\max}+1}, \text{otherwise} \end{cases}$;
  for $t = t_{\max}, \cdots 1$ **do**
    $R \leftarrow r_t + \gamma R$;
    $A_t \leftarrow R - V_t$;
    Accumulate gradients from value loss: $d\theta \leftarrow d\theta + \lambda \frac{\partial A_t^2}{\partial \theta}$;
    $\delta_t \leftarrow r_t + \gamma V_{t+1} - V_t$;
    $\hat{A}_t \leftarrow \gamma\tau \hat{A}_{t-1} + \delta_i$;
    Accumulate policy gradients with entropy regularization: $d\theta \leftarrow d\theta + \nabla \log \pi_t(a_t)\hat{A}_t + \beta \nabla H(\pi_t)$;
  end
end

Algorithm 2: Training of curiosity-driven π_g

Note here we in practice use *generalized advantage estimator* [6].

Pivotal State Learning Progression

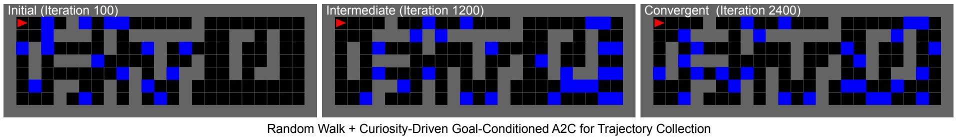


Figure S3: The set of pivotal states (from medium maze) evolves over training by exploring more coverage and more balance of the maze.

Pivotal State Learned via Different Trajectories

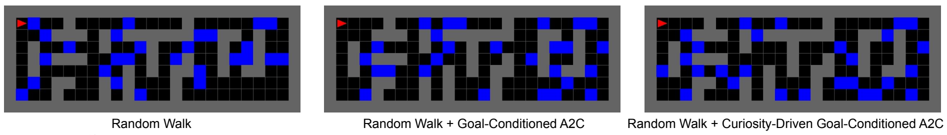


Figure S4: Pivotal states learned via different types of training trajectories (from medium maze).

Kumaraswamy Distribution

Important facts about Kumaraswamy Distribution that we leverage in the paper:

- Kuma Distribution highly resembles Beta Distribution in shape, but not coming from the exponential family.
- When $\alpha = 1$ or $\beta = 1$, $\text{Kuma}(\alpha, \beta) = \text{Beta}(\alpha, \beta)$.
- Similar to Beta, Kuma also ranges from bimodal (when $\alpha \approx \beta$) to unimodal ($\alpha/\beta \to 0$ or $\alpha/\beta \to \infty$)
- Kuma, with support $(0, 1)$, has a simple Cumulative Distribution Function (CDF),

$$F_{\text{Kuma}}(x, \alpha, \beta) = (1 - (1 - x^{\alpha}))^{\beta}. \tag{S1}$$

therefore is especially amendable to the reparametrization trick by sampling from uniform distribution $u \sim \mathcal{U}(0, 1)$:

$$z = F^{-1}_{\text{Kuma}}(u; \alpha, \beta) \sim \text{Kuma}(\alpha, \beta). \tag{S2}$$

- $\text{KL}(\text{Kuma}|\text{Beta})$ has a closed form approximation:

$$\text{KL}(\text{Kuma}(a, b)|\text{Beta}(\alpha, \beta)) = \frac{a - \alpha}{a}(-\gamma - \Psi(b) - \frac{1}{b}) \\ + \log(ab) + \log \text{Beta}(\alpha, \beta) - \frac{b-1}{b} + (\beta - 1)b \sum_{m=1}^{\infty} \frac{1}{m + ab} \text{Beta}(\frac{m}{a}, b).$$

where Ψ is the Digamma function, γ euler constant, and it can be approximated via the first few terms of the Taylor series expansion sum. We take the first 5 terms here.

Hard Kumaraswamy Distribution

We follow the steps in [1]. First stretch the support to $(r = 0 - \epsilon_1, l = 1 + \epsilon_2)$, $\epsilon_1, \epsilon_2 > 0$, and the resulting CDF distribution takes the form:

$$F_S(z) = F_{\text{Kuma}}(\frac{z - l}{r - l}; \alpha, \beta).$$

Then, the non-eligible probabilities for 0's and 1's are attained by rectifying all samples below 0 to 0 and above 1 to 1, and other value as it is, that is

$$P(z = 0) = F_{\text{Kuma}}(\frac{-l}{r - l}; \alpha, \beta), P(z = 1) = 1 - F_{\text{Kuma}}(\frac{1 - l}{r - l}; \alpha, \beta).$$

Derivation of $\mathcal{L}_0$ and $\mathcal{L}_T$

$$\mathcal{L}_0 = \left\|\mathbb{E}_{q(\mathbf{Z}|\mathbf{S},\mathbf{A})}\left[\|\mathbf{Z}\|_0\right] - \mu_0\right\|^2, \text{ where}$$

$$\mathbb{E}_{q(\mathbf{Z}|\mathbf{S},\mathbf{A})}\left[\|\mathbf{Z}\|_0\right] = \sum_{t=1}^{T} \mathbb{E}_{q(z_t|\mathbf{S},\mathbf{A})}\left[\mathbb{1}_{z_t \neq 0}\right] = \sum_{t=1}^{T} 1 - p\left(z_t = 0\right) = \sum_{t=1}^{T} 1 - F_{\text{Kuma}}\left(\frac{-l}{r-l}; a_t, b_t\right),$$

$$\mathcal{L}_T = \|\mathbb{E}_{q(\mathbf{Z}|\mathbf{S},\mathbf{A})} \sum_{t=1}^{T-1} \mathbb{1}_{z_t \neq z_{t+1}} - 2\mu_0\|^2, \text{ where}$$

$$\mathbb{E}_{q(\mathbf{Z}|\mathbf{S},\mathbf{A})}[\sum_{t=1}^{T-1} \mathbb{1}_{z_t \neq z_{t+1}}] = \sum_{t=1}^{T-1} p\left(z_t{=}0\right)\left(1{-}p\left(z_{t+1}{=}0\right)\right) + \left(1{-}p\left(z_t{=}0\right)\right)p\left(z_{t+1}{=}0\right).$$

Implementation Details for Latent Model V

Our models are optimized with Adam [4] through gradient descent using mini-batches of size 128, thus spawning 128 asynchronous agents to explore. For Adam optimizer, initial learning rate is 0.0001, $\epsilon = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$; gradients over 40 are clipped to 40 for inference and generation nets. For HardKuma, $l = -0.1$ and $r = 1.1$. The maximum rollout for BiLSTM is 25. The total number of training iterations is 3600. Prior network, inference network and generation network are trained end-to-end [8].

For each batch of random trajectories, we re-use its starting and ending points to update π_g following Algorithm 2 for 10 iterations as it is much slower for π_g to converge than the latent model. The observed trajectories from the 10th iteration is kept as training examples for the latent model. Thus the total number of training iterations for π_g is 36K.

We initialize λ_i's (see main text) to be $\lambda_1 = 0.01$ (KL-divergence),$\lambda_2 = 0.06$ ($\mathcal{L}_0$), $\lambda_3 = 0.02$ ($\mathcal{L}_T$). After each update of the latent model, we update λ_i's, whose initial learning rate is 0.0005, by maximizing the original objective in a similar way as using Lagrangian Multiplier. At the end of optimization, λ_i's converge to locally optimal values. For example, for medium maze, $\lambda_1 = 0.067$ for the KL-term, $\lambda_2 = 0.070$ for the $\mathcal{L}_0$ and $\lambda_3 = 0.051$ for the $\mathcal{L}_T$ term.

Hierarchical Reinforcement Learning

Algorithm for π^m Learning

For Hierarchical RL models—in our work basing on Feudal Network (FN) [2, 9] —the training of the Worker policy π^w follows the same A2C algorithm as π_g (see Algorithm 2). The training of the Manager policy π^m also follows a similar procedure but as it operates at a lower temporal resolution, its value function regresses against the t_m-step discounted reward where t_m covers all actions and rewards generated from the Worker. We detail the algorithm for π^m learning here:

Exact Entropy for Manager Policy with Wide-Narrow Instruction

Essentially there are $O\left(|\mathcal{V}| \times (N^2)\right)$ possible actions. To calculate the entropy exactly, all of them has to be summed, making it easily computationally intractable:

$$\mathcal{H}{=}\sum_{w \in \mathcal{V}} \sum_{w_n \in s_w} \pi^n(w_n|s_w, s_t)\pi^\omega(w|s_t) \log \nabla\pi^n(w_n|s_w, s_t)\pi^\omega(w|s_t).$$

Taking Actions with Traversal and Optional Finetuning of π_g

Below is the procedure of the Manager and the Worker in sending/receiving orders using either traversal paths among $\mathcal{V}$:

1. the Manager gives a wide-narrow subgoal pair (g_w, g_n).

Initialize network parameters θ for π^m, here $\pi^{m,t}$ refers to the policy at time rollout time step t;
Given a map of $\mathcal{V}$, $s_{\mathcal{V}}$;
for iter $= 0, 1, 2, \cdots$ **do**
  Clear gradients $d\theta \leftarrow 0$;
  Reset the set of time steps where $\pi^{m,t}$ omits a new subgoal $S_m = \{\}$ and $t_m = 0.$;
  while $t <= t_{\max}$ *or episode not terminated* **do**
    Simulate under current policy $\pi^{m,t-1}, \pi^{w,t-1}$;
    if *the Worker has met the previous subgoal or exceeded the horizon* c **then**
      Sample a new subgoal $g_{m,t}$ from $\pi^{m,t}$;
      $z_{m,t} = f_{\text{LSTM}}(\text{CNN}(s_{m,t}, s_{\mathcal{V}}), h_{m,t_m}),\ V_{m,t} = f_v(z_{m,t}),\ \pi_t = f_p(z_{m,t})$;
    end
    $S_m = S_m \cup \{t_m\}$ and $t_m = t$;
  end
  $R = \begin{cases} 0, & \text{if terminal} \\ V_{t_{\max}+1}, & \text{otherwise} \end{cases}$;
  for $t = t_{\max}, \cdots 1$ **do**
    $R \leftarrow r_t + \gamma R$;
    if $t \in S_m$ **then**
      $A_{m,t} \leftarrow R - V_{m,t}$;
      Accumulate gradients from value loss: $d\theta \leftarrow d\theta + \lambda \frac{\partial A_{m,t}^2}{\partial \theta}$;
      Accumulate policy gradients with entropy regularization:
       $d\theta \leftarrow d\theta + \nabla \log \pi_{m,t}(g_{m,t}) A_{m,t} + \beta \nabla H(\pi_{m,t})$;
    end
  end
end

Algorithm 3: Training of π^m for FN HRL models

2. Agent takes action based on the Worker policy π^w that is conditioned on (g_w, g_n) and reaches s'. If $s' \in \mathcal{V}$, g_w has not yet met, and there exists a valid path basing on the edge paths from the world graph $s' \to g_w$, agent then follows this path to reach g_w.
3. When agent reaches g_w for the first time during the current horizon, either through traversal or the Worker's policy, the Worker receives reward +1.
4. If agent reaches g_n, the Worker receives reward +1 and terminates this horizon.
5. the Worker receives reward -0.01 for every action agent takes outside of traversal.
6. Either g_n is reached or the maximum time step for this horizon is met, the Manager renews its subgoal pair.

Alternatively, if there are significant task-specific changes in the environment, such as new blockages or stochasticity, we instead prefer to guide traversal between s' and g_w with π_g, the goal-conditioned policy learned in world graph discovery stage (step 2) and then add a final step to finetune π_g under the task-specific environment using s' as the starting state and g_w the goal state.

Implementation Details for HRL

We inherit most hyperparameters from the training of π_g, as the Manager and the Worker both share similar architecture as π_g. The hyperparameters of π_g in turn follow those from [7]. Because these tasks are more difficult than goal-orientation, we increase the maximal number of training iterations from 36K to 100K and the rollout steps for each iteration from 25 to 60. Hyperparameters specific to HRL are the horizon $c = 20$ and the size of s_w, $N = 5$ for small and medium, $N = 7$ for large. We follow a rigorous evaluation protocol acknowledging the variability in Deep RL [3]: each experiment is repeated with 3 seeds [10, 5], 10 additional validation seeds are used to pick the best model which is then tested on 100 testing seeds. Mean and variance of testing results are summarized in the main text.

The RL models are optimized with Adam [4] through gradient descent using mini-batches of size 32, thus spawning 32 asynchronous agents. The hyperparameters are mostly consulted from the settings in [7]. The set of hyparameters are fine-tuned on π_g first, if we observe any need of adjustment, tune the corresponding hyperparameters with π_g on small maze, as it has quick turn-around rate for trial and error. For other environments and models, hyperparameters are mostly directly inherited from π_g and the model specific hyperparameters are estimated to a reasonable value basing on the environment and other related hyperparameters.

For Adam optimizer, initial learning rate is 0.0005, $\epsilon = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$; gradients over 20 are clipped to 20. Discount rate γ for return is 0.95 for both the Manager and the Worker. The objective for value estimation is weighted with $\lambda = 5$. The entropy term is weighted with $\beta = 0.01$. The rewards are designed to range between -1 and 1 so no clipping is needed. The maximum rollout for each LSTM during training is 60; the horizon $c = 20$ is a third of the rollout size. The maximum training iteration is $36K$ for π_g and $100K$ for the rest, and training is stopped early if model performance reaches a plateau. The size of Manager's local attention is $N = 5$ for small and medium mazes, $N = 7$ for large, which is roughly estimated based on maze size and the size of $\mathcal{V}_p$. The size of $\mathcal{V}_p$ is set to be 20% of the whole state space.

References

[1] J. Basting and et al. Interpretable neural predictions with differentiable binary variables. 2019.

[2] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

[3] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[4] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.

[5] G. Ostrovski, M. G. Bellemare, A. v. d. Oord, and R. Munos. Count-based exploration with neural density models. *ICML*, 2017.

[6] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[7] w. Shang, h. van Hoof, and m. Welling. Stochastic activation actor-critic methods. 2018.

[8] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems*, pages 3483–3491, 2015.

[9] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3540–3549. JMLR. org, 2017.

[10] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *NIPS*, 2017.