Neural Algorithms for Graph Navigation

Anonymous Author(s)
Affiliation
Address
email

Abstract

The application of deep reinforcement learning (RL) to graph learning and meta-learning admits challenges from both topics. We consider the task of one-shot, partially observed graph navigation, acknowledging and addressing the difficulties of partially observed graph environments. In this work, we present a framework for graph meta-learning, and we propose an agent equipped with external memory and local action priors adapted to the underlying graphs. We demonstrate the efficacy of our framework through partially-observed navigation on synthetic graphs, as well as application to partially-observed navigation on 3D meshes, showing substantially improvement in one-shot performance over baseline agents.

1 Introduction

Learning a general exploration strategy is a fundamental problem in AI, which requires the framework of Reinforcement Learning (RL) [29, 24, 12]. In this paper, we focus on *coverage* as a graph exploration problem, such that given a possibly unseen graph environment, our goal is to efficiently visit (i.e., cover) as many new states (i.e., vertices) as possible within a time budget. We formulate the state-coverage problem as a meta-learning task [31], where the goal is to learn a policy from a distribution of input environments that can efficiently explore and generalize to unseen environments. There has been much work on meta reinforcement learning [10, 8], as well as using memory for partially observed tasks [23, 11, 6], but learning on graph environments requires specific considerations.

The enormous success of Graph Neural Networks (GNN) [26, 3, 17, 7] in learning node embeddings can be attributed in part to their explicit permutation invariance. However, we observe that navigation objectives on graphs with indistinguishable nodes cannot necessarily be optimally solved. Including canonical node labels guarantees that the nodes are distinct, but with canonical labels GNNs no longer guarantee permutation invariance [22]. Furthermore, although there exist networks with multiscale dependence on the graph structure [4], partial observability enforces a bound on the GNN spatial width and stymies direct dependence on non-local features. To address these issues, we summarize the main contributions:

- We propose memory-augmented meta-RL for navigation on unseen graphs.
- We introduce an external memory adapted to the graph environment, which enables a recurrent model with greater representational power for the task of meta-RL graph navigation.
- We introduce a local entropy prior for more efficient exploration on graphs during training.
- We perform extensive experiments and an ablation study on synthetic and real-world graph datasets demonstrating the efficacy of our proposed model.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

2 Related Work

Reinforcement Learning over Graphs. Reinforcement learning on graph environments has received more attention recently, particularly in the setting of combinatorial optimization [2, 21]. Recent works in this setting span numerous computational challenging problems including Traveling Salesman [14], MAXCUT [1, 35], and community detection [33] among others. Our setting departs from most combinatorial optimization problems in the assumptions of partial observability and locally determined action space. Most similar to our work is a study of transferable graph exploration [6], although their work mainly concerns learning with an action space independent of the graph structure, and therefore the memory of a regular LSTM model suffices in their environments.

Memory-Augmented Neural Networks. The incorporation of deep learning with an external memory has been well-studied [11]. In reinforcement learning, recurrent parameterizations are well-suited for partially observed MDPs [13]. Our work considers two separate notions of memory, namely spatially adapted memory for reinforcement [23], and memory empowering graph neural networks [15]. Our proposed memory generalizes both notions, defining a time-dependent embedding over arbitrarily structured graphs.

3 Proposed Framework

3.1 Graph Reinforcement Environment

Our setting concerns meta-learning a policy to effectively explore/navigate *unseen graphs*, where the full structure of the graph is not observed and rather needs to be explored by the agent. Formally, we define the POMDP as follows: for a given graph $G = (V, E)$, the state space and action spare are given by $S = A = V$, with deterministic dynamics that only allow for transitions between adjacent vertices. The reward function introduces partial observability into the problem, with positive reward for visiting a node for the first time in an episode, and small negative reward for each timestep. For our experiments we use $+1$ and -0.1, respectively.

This setup characterizes the graph environment for a single graph, while our ultimate goal is to train an agent that can effectively *one-shot* navigate an unseen graph. Formally, for an input distribution of graph environments $\mathcal{G}$, we are interested in the objective $\max_\pi E_{G\sim\mathcal{G},\tau\sim\pi,P_G}\left[\sum_{i=0}^{T} R_G(s_i, a_i, s_{i+1})\right]$ where R_G and P_G denote the reward function and dynamics of G, and we use $\tau \sim \pi, P_G$ to indicate the dependence of the trajectory on the policy and the dynamics of the sampled graph. This objective doesn't assume "rapid learning" as in MAML [10], but only that the learned policy will be run on unseen test graphs without further conditioning.

3.2 External Graph Memory

Neural Network Architecture. We first characterize the memoryless neural network that parameterizes our policy π and value function V. For compactness, we assume bias terms are subsumed in the respective weight matrix. Consider a graph environment given by G with $n = |V|$ vertices. By abuse of notation, we will also use s to denote the initial state embedding given by $s \in \mathbb{R}^{n\times k}$ where each node possesses k features (one of which indicates the agent's current location). Then the intermediate embedding is defined through a graph neural network (GNN): $\phi(s) = \text{GNN}\big(A, \text{GNN}(A, s)\big)$.

We point out this embedding is also a matrix of node embeddings, namely $\phi(s) \in \mathbb{R}^{n\times h_1}$, where h_1 is a hidden dimension. The graph neural network architecture is given by a multi-headed graph attention network [30] with skip connections from the input. Given this shared embedding, we further choose weights $W_1 \in \mathbb{R}^{h_2\times h_1}$, $w_2 \in \mathbb{R}^{h_2}$, $W_3 \in \mathbb{R}^{h_3\times h_1}$, $W_4 \in \mathbb{R}^{h_4\times h_3}$, $w_5 \in \mathbb{R}^{h_4}$. Now, for an activation function σ and self-attention pooling operator POOL [19] over node embeddings, we define,

$$\log \pi(\cdot|s) = w_2^T\ \sigma\big(W_1\ \phi_\pi(s)^T\big)\ , \quad V(s) = w_5^T\ \sigma\Big(W_4\ \sigma\big(\text{POOL}(W_3\ \phi_V(s)^T)\big)\Big) \tag{1}$$

where for this model $\phi_\pi = \phi_V = \phi$. Furthermore, for simplicity we explicitly enforce that $\pi(\cdot|s)$ is only supported on the set $N(s)$, as all other actions do not move the agent. With these parameterizations, we train using the PPO algorithm [28].

We now characterize the model using external graph memory. The recurrent model conditions the state embedding ϕ on an external memory embedding $M \in \mathbb{R}^{n \times h_M}$. Therefore ϕ now depends on the entire trajectory: we characterize the time-dependent update as $M_0 = 0$ and

$$
\begin{aligned}
r &= \text{GNN}\big(A, \text{GNN}(A, [M_t|s])\big) \\
\tilde{M}_{t+1} &= \text{GRU}(r, M_t) \\
(M_{t+1})_v &= \begin{cases} (M_t)_v & \text{If } v \notin \{s \cup N(s)\} \\ (\tilde{M}_{t+1})_v & \text{If } v \in \{s \cup N(s)\} \end{cases} \\
\psi(s, M_t) &= [\phi(s)|M_{t+1}]
\end{aligned} \tag{2}
$$

In other words, after reading global information into a matrix r, we apply a Gated Recurrent Unit [5] to locally update our memory embedding, i.e. we only update the rows of M_{t+1} corresponding to vertices in a 1-hop neighborhood of s. For simplicity the GRU cells above share weights. We also mention that for the recurrent model we choose $\phi_\pi = \psi$ but keep $\phi_V = \phi$, as we observed more instability when the value function was allowed to depend on the memory embeddings. The principle of using an external and shaped memory was explored in the work of [23]. However, their method was restricted to 2D and 3D grids, requiring a notion of global coordinates which isn't present for experimental settings, and isn't present for graphs in general.

Entropy Regularization. In order to ensure adequate exploration during training, the PPO objective is typically augmented with an entropy regularizer. We incorporate this term by learning a stochastic policy π and adding $-\lambda E_{\tau \sim \pi, s \sim \tau}[H(\pi(\cdot|s))]$ to our loss, where $H(p) = -\sum_{i=1}^{n} p_i \log p_i$. However, if our graph has bounded maximum degree, then the set of legal actions at any state will be significantly smaller than the set of all actions.

We address this mismatch by instead minimizing the KL divergence between $\pi(\cdot|s)$ and an appropriate prior distribution Q_s. Of course, for the choice $Q_s(a) \propto \delta_{a \in N(s)}$ where $N(s)$ is the neighborhood of s, this objective is equivalent to maximizing entropy up to a constant term, but we may consider other priors which encourage more efficient exploration of the environment.

For example, in continuous domains, the work in [20] uses an Ornstein-Uhlenbeck process to encourage exploration with a consistent direction over time. As a simple analogue on graphs, we consider the prior $Q_s(a) \propto \delta_{a \in N(s)} + \alpha \delta_{a = \tilde{s}}$, where $\tilde{s}$ is the state immediately previous in the current trajectory and α is a hyperparameter. Choosing negative α induces a prior that discourages backtracking, which will naturally encourage faster mixing.

4 Experiments

Synthetic Datset. Our experiments seek to understand the impact of our contributions and the various choices in the proposed model, namely an external graph memory and a local relative entropy prior. We include node level features indicating the agent's current node, the nodes visited during the episode, and a normalized timestep.

We will nevertheless refer to the model equipped with these features, but without the graph-adapted memory parameterization as "memoryless", as it only possesses an unlearned, binary representation of the past visitations. The modeling choices of memory and entropy prior yield four possible models, all of which we train and test under the same meta-learning conditions. Our implementation of PPO is based on the code given in [18]. We generate synthetic graphs using different random graph families (Watts-Strogatz, Erdos Renyi, Tree graphs), see Appendix A for details of the experimental setup and the synthetic graph generation.

ModelNet Dataset. For real-world applications, the coverage reward function may be too simplistic, since it assigns equal value to visiting each node. In practice, the value of a node may depend on its

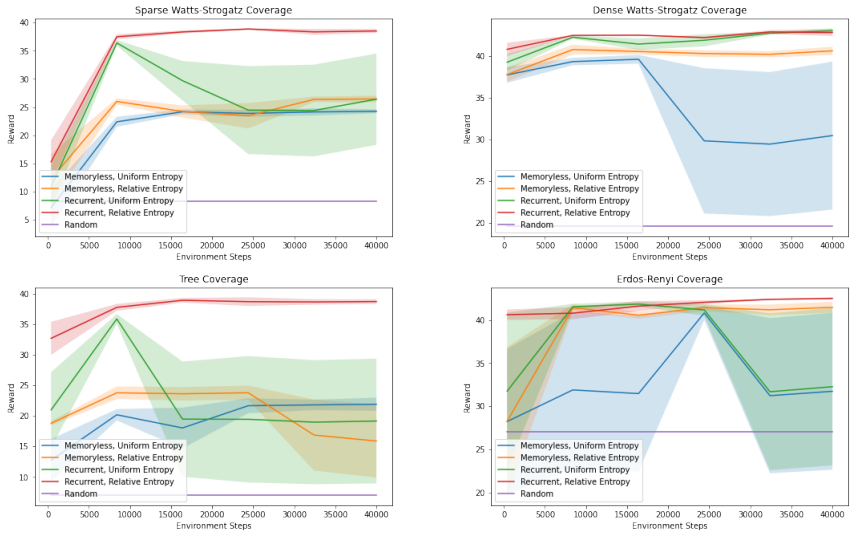
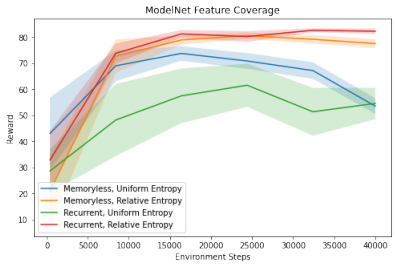
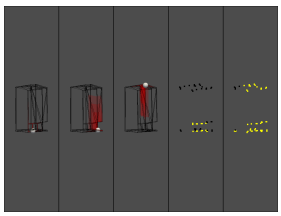


Figure 1: Mean test reward over 100 held-out graphs over the four synthetics graph families. The rewards are calculated over 5 independent runs, with confidence intervals given by normalized standard deviation.

underlying features. To that end, we consider a real-world dataset (ModelNet10) equipped with a reward function (given explicitly in the appendix) that provides an explicit incentive to the agent to prioritize visiting nodes near several so-called “anchors" points rather than all nodes uniformly. For our experiments we sample three anchor points.



(a) Mean test reward over 100 held-out graphs sampled from the ModelNet dataset.



(b) Visualization of a ModelNet mesh and performance of the two most extreme models, memoryless and regular entropy (fourth column) vs. graph-adapted memory and the local relative entropy prior (fifth column).

We train on 100 graphs from ModelNet, decorated with features using a process given in the Appendix, and test on 100 unseen graphs from the same distribution, with the episode length capped at 100 steps. The results are plotted in Figure 2a. In Figure 2b we visualize the feature-aware reward: anchor points are white, densities are red, unvisited nodes are black, and visited nodes are yellow.

5 Conclusion

In this work, we empirically demonstrated the merit of graph-adapted memory and non-uniform entropy priors for the task of meta-graph navigation. Our work offers a first step in understanding the impact of RL and meta-RL in the context of graph environments, under the implicit constraints induced by using GNNs for parameterization.

For future work, the question of how to scale RL based on graph trajectories to substantially larger graphs remains challenging, as backpropagating through memory over the entire trajectory becomes increasingly difficult with increased length. Additionally, there may exist more diverse choices of local relative prior beyond controlling the degree of backtracking, which may improve performance further.

References

[1] Thomas D Barrett, William R Clements, Jakob N Foerster, and Alex I Lvovsky. Exploratory combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1909.04063*, 2019.

[2] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[3] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[4] Zhengdao Chen, Lisha Li, and Joan Bruna. Supervised community detection with line graph neural networks. In *International Conference on Learning Representations*, 2019.

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[6] Hanjun Dai, Yujia Li, Chenglong Wang, Rishabh Singh, Po-Sen Huang, and Pushmeet Kohli. Learning transferable graph exploration. In *Advances in Neural Information Processing Systems*, pages 2518–2529, 2019.

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[8] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017.

[9] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1): 17–60, 1960.

[10] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[11] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[12] Arthur Guez, Theophane Weber, Ioannis Antonoglou, Karen Simonyan, Oriol Vinyals, Daan Wierstra, Remi Munos, and David Silver. Learning to search with mctsnets. In *International Conference on Machine Learning*, pages 1822–1831, 2018.

[13] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

[14] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[15] Amir Hosein Khasahmadi, Kaveh Hassani, Parsa Moradi, Leo Lee, and Quaid Morris. Memory-based graph networks. *arXiv preprint arXiv:2002.09518*, 2020.

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*, 2014.

[17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. 2017.

[18] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018.

[19] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pages 3734–3743, 2019.

[20] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[21] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *arXiv preprint arXiv:2003.03600*, 2020.

[22] Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. *arXiv preprint arXiv:1811.01900*, 2018.

[23] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *International Conference on Learning Representations*, 2018.

[24] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[25] Heinz Prüfer. Neuer beweis eines satzes über permutationen. *Arch. Math. Phys*, 27(1918):742–744, 1918.

[26] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[27] William J Schroeder, Jonathan A Zarge, and William E Lorensen. Decimation of triangle meshes. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70, 1992.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[31] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

[32] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684): 440–442, 1998.

[33] Bryan Wilder, Eric Ewing, Bistra Dilkina, and Milind Tambe. End to end learning and optimization on graphs. In *Advances in Neural Information Processing Systems*, pages 4672–4683, 2019.

[34] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[35] Weichi Yao, Afonso S Bandeira, and Soledad Villar. Experimental performance of graph neural networks on random instances of max-cut. In *Wavelets and Sparsity XVIII*, volume 11138, page 111380S. International Society for Optics and Photonics, 2019.

A Appendix

A.1 Experiment Discussion

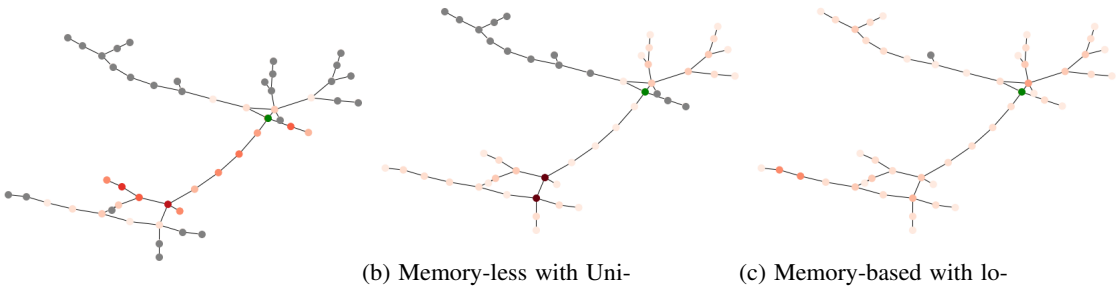
Discussion. For the synthetic graphs, the results divide according to the sparsity of the underlying graphs. For the sparse Watts-Strogatz and tree graphs, the model with both proposed methods perform best, with a marked difference between the models with graph-adapted memory versus the memoryless models. Notably, the introduction of the relative entropy prior that biases against backtracking stabilizes the performance across different random initializations. As corroborated by the median reward in Table 2, the recurrent model with regular entropy is capable of performing similarly to the recurrent model with relative entropy. But the random trajectories encouraged by a uniform action prior can result in instability and poor performance, dramatically increasing the normalized standard deviation of the former model.

For the dense Watts-Strogatz model and the Erdos-Renyi model, we observe that the choice of entropy prior has significantly less impact on the performance. This is unsurprising, as increasing the graph density implies higher average degree, and the distributional distance between the uniform action prior and the action prior biased against backtracking decreases as the support of the action distribution increases. However, the inclusion of graph-adapted memory still benefits the model significantly, indicating that even in the denser regime, saving memory of the past trajectory is crucial for efficient exploration.

Discussion. On ModelNet, even after downsampling the mesh graphs retain high density and low diameter. Nevertheless, we see different behavior than on the dense synthetic families as a result of the feature-dependent reward function. The choice of non-backtracking prior still improves performance, but the distinction between memoryless and recurrent model becomes smaller. We attribute this change to the nature of the reward function; once the agent enters the support of a truncated Gaussian, it may greedily choose actions that most increase the density for that feature to locate the anchor point, and therefore the region of highest reward.

In Figure 2b, we compare the visited nodes under the memoryless, uniform entropy prior model against our proposed model. Indicated by the yellow highlighted points, the former (fourth column) locates and effectively covers the two lower anchor points but struggles to sufficiently explore and cover the third before the time budget is exhausted. Meanwhile, the latter (fifth column) effectively covers all the anchors, as well as most of the graph vertices, which will possess a small positive reward if inside the support of one of the truncated Gaussians. The agent neglects to visit nodes in the top left which have zero density and therefore zero positive reward.

A.2 Experimental Details



(a) Random Policy

(b) Memory-less with Uniform Entropy

(c) Memory-based with local entropy prior

Figure 3: Visualization of vertex coverage after a 100-step episode on a sparse Watts-Strogatz graph, using the policy models, (a) uniformly random actions, (b) memory-less and regular entropy, and (c) graph-adapted memory with local entropy prior. The green color indicates the starting node, the gray indicates unvisited nodes, and the visited nodes are colored red proportional to their total number of visits (darker red color $\rightarrow$ more visits).

Synthetic Graphs Experimental Setup. To test our proposed framework, we consider several random graph families from which we sample training and test sets. Specifically, we consider 50 vertex graphs drawn from the following families:

- Sparse Watts-Strogatz graphs (Sparse WS) [32] graphs conditioned on connectedness.
- Dense Watts-Strogatz graphs (Dense WS) [32] graphs conditioned on connectedness.
- Uniformly random tree graphs [25].
- Erdos-Renyi graphs conditioned on connectedness [9].

The Watts-Strogatz graph distribution is characterized by two parameters [32], the average node degree k and rewiring probability p, where the former primarily controls graph density and diameter. For the sparse family we choose $k = 2, p = 1$ and for the dense family we choose $k = 4, p = 0.1$. For the Erdos-Renyi graph we sample each edge with a uniform probability $p = 0.1$. Average statistics of the four graph families, indicating their level of density, are given in Table 1. Further characterization of this synthetic experiment, and other details of the experimental setup are given in the supplementary materials.

Table 1: Statistics of the four synthetic graph families, averaged over 500 graphs of 50 vertices in each graph.

Graph Family	**Edge Density**	**Average Diameter**
Sparse WS Graph	0.0408	17.44
Dense WS Graph	0.0816	8.15
Tree Graph	0.0400	18.87
Erdos-Renyi Graph	0.1011	5.13

Table 2: Final median test reward on four synthetic graph families, for all choices using memory & local prior.

Graph Family	**Memory?**	**Local Prior?**	**Reward**
Sparse WS Graph	✗	✗	24.48
	✗	✓	25.70
	✓	✗	36.46
	✓	✓	**38.41**
Dense WS Graph	✗	✗	40.80
	✗	✓	40.72
	✓	✗	**42.75**
	✓	✓	42.61
Tree Graph	✗	✗	22.61
	✗	✓	22.20
	✓	✗	36.20
	✓	✓	**38.44**
Erdos-Renyi Graph	✗	✗	41.39
	✗	✓	41.99
	✓	✗	42.41
	✓	✓	**42.58**

For each graph family, we sample 500 training graphs and 100 testing graphs, using the coverage reward function as stated above, with the number of steps per episode capped at 100. The results of the synthetic experiment over the four chosen graph families are given in Figure 1. We plot how the test mean reward evolves during training, and test median reward for the graph families are given in Table 2.

ModelNet Graph Experimental Setup. The ModelNet10 dataset [34] consists of meshes in 3D space from 10 categories, equipped a simplex structure of vertices, edges, and faces. In order to

render meshes as graph environments, we apply decimation [27] to downsample while preserving overall shape, until the underlying graph has between 50 and 100 vertices.

In order to place meaningful features on our induced graphs, we consider the following process on each graph:

- We sample several distinct “anchor" points from among the vertices.
- We define a truncated, isotropic Gaussian in coordinate space after normalizing the pairwise distances, centered on each anchor point.
- For each vertex, we take as features the density under the respective Gaussian distributions.
- We further restrict the initial distribution to be uniform over all vertices except the “anchor" points.

Assume this process induces a feature vector x_v for each $v \in V$. Then we can define the feature-aware reward function as,

$$R(s_n, a_n, s_{n+1}|\tau) = \begin{cases} -0.1 & s_{n+1} \in \tau \\ -0.1 + \mathbf{1}^T x_{s_{n+1}} & \text{otherwise} \end{cases} \tag{3}$$

The ModelNet meshes are explicitly processed as follows:

- Discard meshes with over 1000 points
- Apply a decimation algorithm [27] to reduce to approximately 100 vertices, discarding meshes with vertex counts outside the range $[50, 100]$
- Extract the graph induced by the largest connected component of the decimated mesh
- On three random, distinct vertices, define a standard Gaussian distribution on normalized 3D coordinates centered at each sampled point
- Define features on all vertices, given by their densities under the three Gaussians, clipping densities below 10^{-3} to 0

Architecture and Hyperparameters. We characterize the critic, policy, and memory networks. Given the parameterization in Section 3, we choose the attention networks in the GAT operator and the attention POOL operator to be two layer networks, with 4 attention heads for the former. All activations are chosen as ReLUs, and all hidden sizes are 32, excluding $h_1 = 64$ and $h_M = 3$.

For training, we optimize with the Adam optimizer [16], using an initial learning rate of $7 * 10^{-4}$ for the synthetic experiments and $2 * 10^{-4}$ for the ModelNet experiments. The PPO clipping parameter is 0.2, and gradients are scaled to bound their norms by 0.5. The batch size is 400, with 4 gradient updates on each batch. Finally, α is chosen as -2.0 for the relative entropy prior model.

A.3 Additional Experiments

Comparison to Single Vector Memory Embedding. In order to compare against a simpler choice of recurrent model, we also consider a reinforcement learning agent with a memory embedding given as a single vector. The policy and value function are parameterized as before, but the recurrent update utilizes a single memory embedding m_t as follows:

$$\begin{aligned} r &= \text{GNN}\big(A, \text{GNN}(A, [\mathbf{1}m_t^T|s])\big) \\ m_{t+1} &= \sum_{v \in N(s)} \text{GRU}(r_v, m_t) \\ \psi(s, m_t) &= [\phi(s)|m_{t+1}] \end{aligned} \tag{4}$$

In other words, the single memory embedding is appended to every node embedding before applying the graph convolutions, and the new memory vector is pooled from its neighbors after an update

through a GRU. We demonstrate that this simpler implementation of memory is outperformed by the graph-adapted memory in Figure 4.

The comparison to the single memory embedding baseline in Figure 4 on the sparser graph families indicates the specific need for structurally determined memory. The performance of the single vector memory model is comparable to the memoryless model, suggesting that in the absence of meaningful features to distinguish vertices, the past trajectory cannot be effectively captured by a single vector.

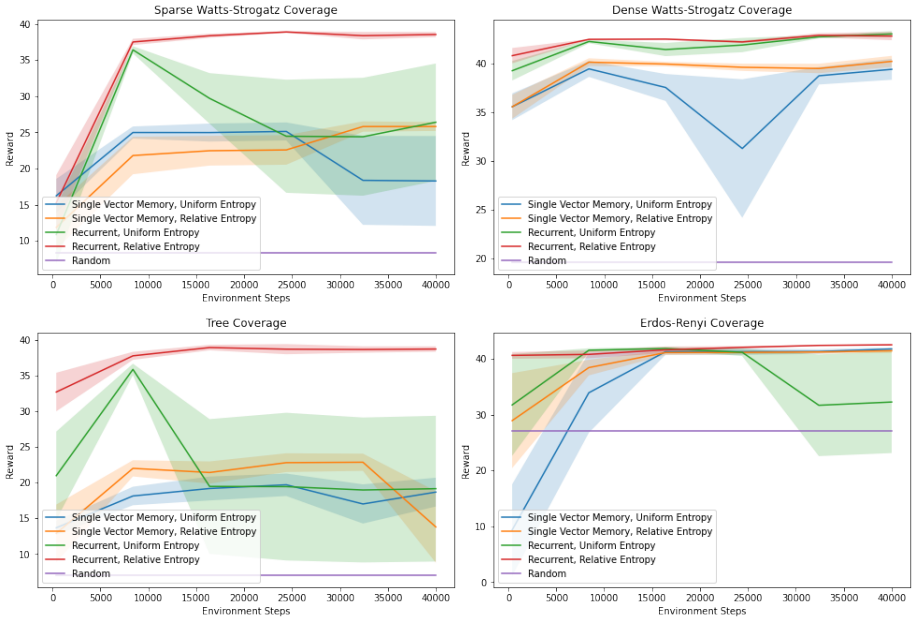


Figure 4: Mean test reward over 100 held-out graphs on synthetic graph families over 5 independent runs, comparing the proposed graph-augmented memory against a single (vector) memory embedding. The recurrent plot is shared with Figure 1.

Additional Visualizations. We include another visualization of the ModelNet mesh visitations, further separating the base model from our proposed model as shown in Figure 5. We also include a visualization of the performance of several considered models in the synthetic experiment setting in Figure 3. We characterize not just the visitation of each node but the total number of visits, noting that repeatedly visiting a node suggests dithering behavior of the agent. We observe that dithering is indeed a component of the random and memoryless models, leading to suboptimal navigation, while our proposed model effectively covers nearly the entire graph.

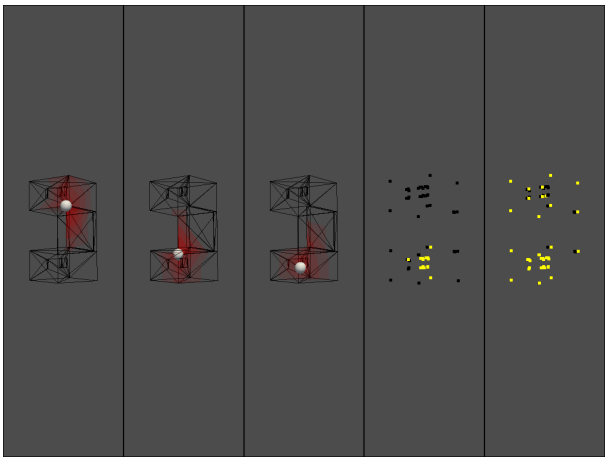


Figure 5: Visualization of a ModelNet mesh from the “desk" category, and the performance of the two most extreme models, memoryless and regular entropy (fourth column) vs. graph-adapted memory and the local relative entropy prior (fifth column).