Sparse Graphical Memory for Robust Planning

Michael Laskin[*1] **Scott Emmons**[*1] **Ajay Jain**[*1] **Thanard Kurutach**[1] **Pieter Abbeel**[1] **Deepak Pathak**[2]

Abstract

To operate effectively in the real world, artificial agents must act from raw sensory input such as images and achieve diverse goals across long time-horizons. On the one hand, recent strides in deep reinforcement and imitation learning have demonstrated impressive ability to learn goal-conditioned policies from high-dimensional image input, though only for short-horizon tasks. On the other hand, classical graphical methods like A* search are able to solve long-horizon tasks, but assume that the graph structure is abstracted away from raw sensory input and can only be constructed with task-specific priors. We wish to combine the strengths of deep learning and classical planning to solve long-horizon tasks from raw sensory input. To this end, we introduce Sparse Graphical Memory (SGM), a new data structure that stores observations and feasible transitions in a sparse memory. SGM can be combined with goal-conditioned RL or imitative agents to solve long-horizon tasks across a diverse set of domains. We show that SGM significantly outperforms current state of the art methods on long-horizon, sparse-reward visual navigation tasks. Project video and code are available at https://mishalaskin.github.io/sgm/.

1. Introduction

A sensorimotor agent in the real world should act from raw sensory data without relying on hand-engineered state estimation, and achieve multiple goals without having to retrain for each of them. Learning-driven approaches to control, like imitation learning and reinforcement learning, have been quite successful in both training agents to act from raw, high-dimensional input (Mnih et al., 2015) as well as to reach multiple goals by conditioning on them (Andrychowicz et al., 2017; Nair et al., 2018). However, this success has been limited to short horizon scenarios, and scaling these methods to distant goals remains extremely challenging. On the other hand, classical planning algorithms have enjoyed great success in long-horizon tasks with distant goals by reduction to graph search (Hart et al., 1968; LaValle, 1998). For instance, A* was successfully used to control *Shakey the robot* for real-world navigation over five decades ago (Doran & Michie, 1966). Unfortunately, the graph space in which these planning search algorithms operate is abstracted away from raw sensory data via domain-specific priors, and planning over the nodes assumes access to well-defined edges as well as a perfect controller to traverse between nodes. Hence, these planning methods struggle when applied to agents operating directly from high-dimensional, raw-sensory images (Mishkin et al., 2019).

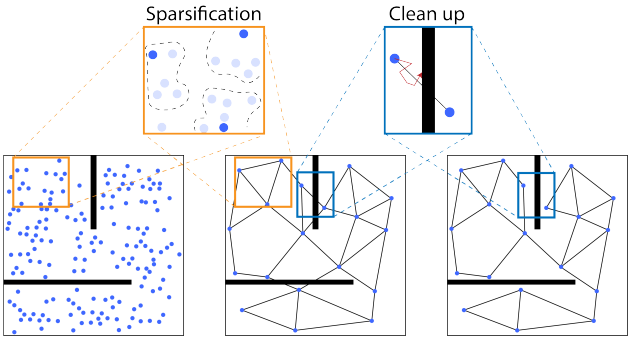
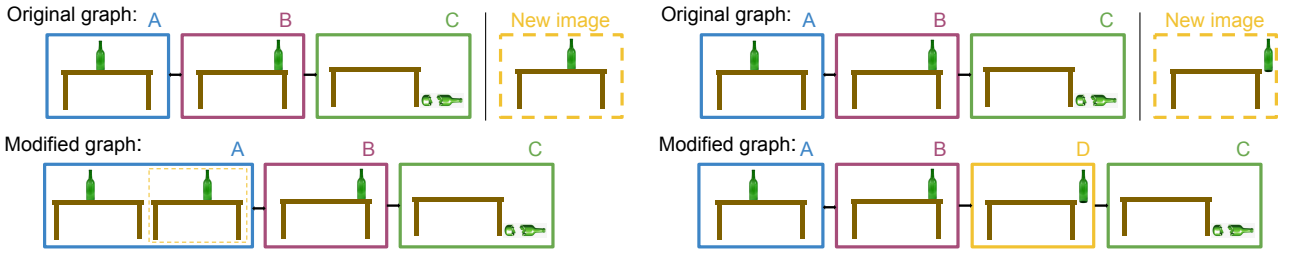


Figure 1. Conceptual visualization of the Sparse Graphical Memory (SGM) building procedure at training time. Sparsification is done while dynamically building the graph. At each time-step, the incoming observation is either merged with existing nodes in the graph or a new node is generated if the observation is visually and temporally far from the other nodes. Once the sparse graph is constructed, incorrect edges are removed with cleanup rollouts.

How can we have best of both worlds, *i.e.*, combine the long-horizon ability of classic graph-based planning with the flexibility of modern, parametric, learning-driven control? One way is to build a graph out of an agent's experience in the environment by constructing a node for every observation and use a learning-based controller (whether RL or imitation) to traverse between those nodes. Some recent work has investigated this direct combination in the context of navigation (Eysenbach et al., 2019; Savinov et al., 2019); however, these graphs grow quadratically in terms of edges and quickly become unscalable beyond small mazes (Eysen-

[*]Equal contribution [1]Berkeley AI Research (BAIR), University of California, Berkeley [2]Facebook AI Research (FAIR). Correspondence to: Michael Laskin <mlaskin@berkeley.edu>.



(a) Node merging example

(b) Node creation example

Figure 2. Two-way consistency. In (a), we have the original directed graph containing 3 nodes – A, B and C where A connects to B, B connects to A, and B connects to C. Given a new image in the dashed yellow box, with which cluster should it be merged? The answer is A because both the perceptual and acting distances to get to and from the node A are small. In (b), we have the same original graph and another new image where the bottle is about to fall off the table edge. With which node should this be merged? At first glimpse, B seems visually similar to the new image and therefore should have a small perceptual distance. However, if we carefully consider the action needed to transition from the new image to B, we find that it is almost impossible to do so. Because the action distance in one direction, i.e., from the new image to B, is large, a new node D should be created.

bach et al., 2019). This strategy either leads to extremely brittle planning trajectories because recovering from errors in such large graphs is infeasible, or else relies on human demonstrations for bootstrapping (Savinov et al., 2019).

In this work, we propose to address this synergistic challenge in combining classical and modern paradigms by dynamically sparsifying the graph as the agent collects more experience in the environment to build what we call *Sparse Graphical Memory* (SGM). In fact, building a sparse memory of key events has long been argued by neuroscientists to be fundamental to animal cognition. The idea of building cognitive topological maps was first demonstrated in rats by seminal work of Tolman (1948). The key aspect that makes building and reasoning over these maps feasible in the ever-changing, dynamic real world is the sparse structure enforced by landmark-based embedding (Foo et al., 2005; Gillner & Mallot, 1998; Wang & Spelke, 2002). Yet, in artificial agents, automatic discovery of sparse landmark nodes remains a key challenge.

One way to discover a sparse graph structure is to dynamically merge similar nodes. But how does one obtain a similarity measure? This is a subtle but central piece of the puzzle. Observations that look similar in the observation space may be far apart in the action space, and vice-versa. Consider the conceptual example in Figure 2(b), where the graph already contains 3 nodes $\{A, B, C\}$. The new node D is visually similar to B, but merging with B would imply that the bottle can be saved from breaking. Therefore, a merely visual representation of the scene cannot serve as a viable metric. We propose to use an asymmetric distance function between nodes and employ *two-way consistency* as the similarity measure for merging nodes dynamically. The basic idea is that two nodes are similar if they both can be reached in similar steps from all their neighbors as well as if all their neighbors can be reached from both of them with similar effort. For our conceptual example, it is not possible to go back from the falling-bottle to the standing-bottle, and hence the two-way consistency does not align for scene B and the new observation. Despite similar visual appearance, they will not be merged. For two-way consistency, we discuss two alternatives: temporal distance learning in a self-supervised fashion and goal-conditioned Q-values learned via RL.

We evaluate the success of our method, SGM, in a variety of navigation environments. First, we observe in Table 1 that SGM has a significantly higher success rate than previous methods, on average increasing the success rate by 40% across the environments tested. As our ablation experiments demonstrate, SGM's success is due in large part to its sparse structure that enables efficient correction of distance metric errors. In addition, we see that the performance gains of SGM hold across a range of environment difficulties from a simple point maze to complex visual environments like ViZDoom and SafetyGym. Finally, compared to prior methods, planning with our proposed sparse memory can lead to nearly an order of magnitude increase in speed.

2. Related Work

Planning is a classic problem in artificial intelligence. In the context of robotics, RRTs (LaValle, 1998) use sampling to construct a tree for path planning in configuration space, and SLAM jointly localizes the agent and learns a map of the environment for navigation (Bailey & Durrant-Whyte, 2006; Durrant-Whyte & Bailey, 2006). Given an abstract, graphical representation of an environment, Dijkstras Algorithm (Dijkstra, 1959) generalizes breadth-first search to efficiently find shortest paths in weighted graphs, and the use of a heuristic function to estimate distances, as done in A* (Hart et al., 1968), can improve computational efficiency.

Beyond graph-based planning, there are various parametric approaches to planning. Perhaps the most popular planning framework is model predictive control (MPC) (Garcia et al., 1989). In MPC, a dynamics model, either learned or known, is used to search for paths over future time steps. To search for paths, planners solve an optimization problem that aims to minimize cost or, equivalently, maximize reward. Many such optimization methods exist, including forward shooting, cross-entropy, collocation, and policy methods (Hargraves & Paris, 1987; Rubinstein, 1999). The resulting agent can either be in open-loop and just follow its initial plan, or in closed-loop and replan at each step.

Aside from MPC, a variety of reinforcement learning algorithms, such as policy optimization and Q-learning, learn a policy without an explicit dynamics model (Lillicrap et al., 2016; Mnih et al., 2013; Schulman et al., 2015; 2017). In addition to learning a single policy for a fixed goal, some methods aim to learn hierarchical policies to decompose complex tasks (Kaelbling, 1993; Pong et al., 2018; Schaul et al., 2015), and other methods aim to learn goal-conditioned policies able to reaching arbitrary goals. Parametric in nature, these model-free approaches are highly flexible, but, as does MPC with a learned dynamics model, they struggle to plan over long time horizons due to accumulation of error.

Recent work combines these graph-based and parametric planning approaches by using past observations for graph nodes and a learned distance metric for graph edges. Variations of this approach include Search on the Replay Buffer (Eysenbach et al., 2019), which assumes access to uniform sampling of the environment for graph nodes; Semi-Parametric Topological Memory (Savinov et al., 2019), which assumes a demonstration to bootstrap the graph; and Mapping State Space Using Landmarks for Universal Goal Reaching (Huang et al., 2019), which subsamples the policys past training observations to choose graph nodes. Hallucinative Topological Memory (HTM) (Liu et al., 2020) uses a contrastive energy model to construct more accurate edges, and Shang et al. (2019) use dynamic programming for planning with a learned graph. The defining feature of our work is a two-way consistency check to induce sparsity, as previous work either stores the entire replay buffer in a graph, limiting scalability as the graph grows quadratically in the number of nodes, or it subsamples the replay buffer without considering graph structure.

3. Preliminaries

We consider long-horizon, goal-conditioned tasks. At test time, an agent is provided with its starting observation o_{start} and a goal observation o_{goal} corresponding to a goal state, and seeks to reach the goal state via a sequential decision making process. Many partially observable and visual tasks can be defined by a goal observation, such as an image of a goal location for navigation.

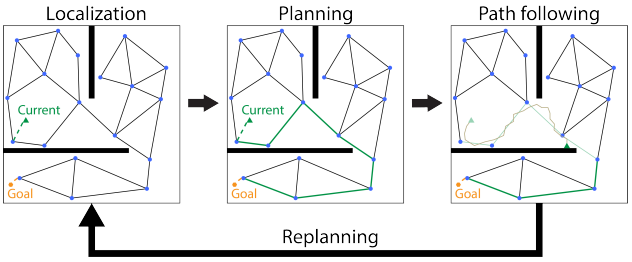


Figure 3. Execution using SGM. In localization, we find the closest node using discrepancies in the asymmetric distance function. In planning, we use Dijkstra's algorithm to find the shortest path (for simplicity we omit the direction of edges here). In path following, divergence from the waypoints or transition failure may happen. The agent then needs correct the memory, relocalize and replan.

We assume access to a short-horizon parametric controller $\pi(o_{\text{start}}, o_{\text{goal}})$ that is capable of accomplishing the task when the starting and goal states are nearby, *i.e.* the optimal action sequence is short. This controller can be learned via reinforcement learning with goal relabeling or self-supervised learning. However, such controllers are generally unable to reach distant goals (Table 2).

To reach distant goals, we propose a semi-parametric agent that models feasible transitions with a nonparametric graph used to guide a parametric low-level controller. To accomplish long-horizon tasks, the semi-parametric agent plans a sequence of waypoint observations $P = \{o_{w_1}, \dots o_{w_p}, o_{\text{goal}}\}$ taken from prior experience in the environment, and uses the controller to reach each waypoint sequentially. The experience used for selecting waypoints is collected simultaneously with controller acquisition and stored in a buffer, *e.g.* for experience replay. To be useful for graph-based planning, the experience is then encoded as a *graphical memory* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, where nodes $\mathcal{V}$ are observations or embeddings, edges $\mathcal{E}$ connect nearby observations, and real-valued weights $\mathcal{W}$ measure the pairwise, asymmetric distance $d(o_u, o_v)$ between observations. The distance function is asymmetric as we do not assume transitions are reversible. In practice, ground truth pairwise distances are not available, so $d(\cdot, \cdot)$ is parameterized by a neural network and learned jointly with the policy.

4. Sparse Graphical Memory for Planning

Given a replay buffer, in Section 4.2, we provide a procedure for constructing *sparse graphical memory* that has limited redundancy between observations through *perceptual* and *two-way acting consistency* checks. This node sparsification allows our graphical memory to scale to large environments. Furthermore, sparsification drastically reduces the number of errors in the graph that we need to remove during *dynamic graph cleanup*, discussed in Section 4.4.

Algorithm 1 `BuildSparseGraph`

Input: replay buffer $\mathcal{B}$, distance function d
Output: sparse graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$
Initialize empty vertex set $\mathcal{V} = \emptyset$
for $\hat{o} \in \mathcal{B}$, each observation in the replay buffer, **do**
  if the observation is novel according to perceptual and two-way consistency, i.e., $\nexists o \in \mathcal{V} : C_p(o, \hat{o}) < \tau_p, C_{\leftarrow}(o, \hat{o}) < \tau_a, C_{\rightarrow}(o, \hat{o}) < \tau_a$ **then**
    add the observation $\hat{o}$ to the graph $\mathcal{G}$:
    $\mathcal{V} = \mathcal{V} \cup \{\hat{o}\}$
    $\mathcal{E} = \mathcal{E} \cup \{(o, \hat{o}) : o \in \mathcal{V}, d(o, \hat{o}) < \text{MAXDIST}\}$
    $\mathcal{E} = \mathcal{E} \cup \{(\hat{o}, o) : o \in \mathcal{V}, d(\hat{o}, o) < \text{MAXDIST}\}$
  end if
end for
assign weights $\mathcal{W}(o_i, o_j) = d(o_i, o_j) \;\; \forall (o_i, o_j) \in \mathcal{E}$
$\mathcal{E}_{\text{filtered}} =$ filter $\mathcal{E}$ to k-NN via Equation 5
return $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$

4.1. Graph Construction

The graphical memory is built either via a single pass through a replay buffer of experience or online during experience collection, according to Algorithm 1. In particular, an observation is only recorded if it is novel. Once an observation is added to the graph, we create incoming and outgoing edges when the distance function meets a MAXDIST threshold, and set the edge weight to the distance.

4.2. Sparsity via Two-way Consistency

A new observation $\hat{o}$ is only added to the memory if it fails a *perceptual similarity* or *two-way acting consistency* check with each observation already in the memory, in which case we consider it novel and useful to retain for planning.

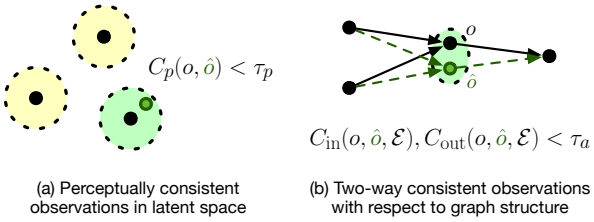
We say that $\hat{o}$ is perceptually consistent with a previously recorded observation $o \in \mathcal{V}$ if

$$C_p(o, \hat{o}) = ||\phi(o) - \phi(\hat{o})||_2 < \tau_p, \tag{1}$$

where C_p measures the visual similarity of observations that the agent receives through the l_2 distance between embeddings of each observation. In state-based tasks, the identity function is used for the embedding $\phi(\cdot)$. However, nearby states can have pixel-space observations that are significantly different, such as when an agent rotates (Savinov et al., 2019). To mitigate this problem, for high-dimensional image observations, $\phi(\cdot)$ is a learned embedding network such as a VAE (Kingma & Welling, 2014) or a subnetwork of the distance function.

Perceptual consistency only verifies that observations are similar visually, and is used as a fast, pairwise, and symmetric test in latent space, as illustrated in Figure 4a.

To merge nodes, we want a measure of *acting* consistency



(a) Perceptually consistent observations in latent space

(b) Two-way consistent observations with respect to graph structure

Figure 4. SGM uses perceptual and two-way distance consistency checks to find redundant pairs of observations in the replay buffer, recording only novel observations.

according to the capabilities of the controller. While perceptual consistency is symmetric, equivalency between nodes should account for the asymmetric, irreversibility of environments and controllers. For example, agents have momentum. To this end, we propose that two states should be aggregated only if they share the same possible incoming and outgoing states, as illustrated in Figure 4b. Our proposed two-way distance consistency is related to bisimilarity for MDP state aggregation (Ferns et al., 2004; Givan et al., 2003), but it is a weaker, approximate notion that uses only asymmetric distances between observations $d(\cdot, \cdot)$ checked locally in the graphical memory, not requiring the unknown MDP. In addition, bisimulation is a very strict criterion (Li et al., 2006) that would not result in meaningful sparsity in the memory.

We compare the new observation locally to predecessors and successors of the recalled node,

$$C_{\leftarrow}(o, \hat{o}, \mathcal{E}) = \max_{u:(u,o)\in\mathcal{E}} |d(u, o) - d(u, \hat{o})| < \tau_a, \text{ and} \tag{2}$$

$$C_{\rightarrow}(o, \hat{o}, \mathcal{E}) = \max_{v:(o,v)\in\mathcal{E}} |d(o, v) - d(\hat{o}, v)| < \tau_a. \tag{3}$$

Taking a conservative approach, we check all three measures of consistency, (1-3). If there is any recorded observation o such that the candidate new observation $\hat{o}$ passes the perceptual consistency and distance consistency checks according to the corresponding thresholds which are hyperparameters, *i.e.*, if $\exists o \in \mathcal{V}$ with $C_p(o, \hat{o}) < \tau_p$, $C_{\leftarrow}(o, \hat{o}) < \tau_a$ and $C_{\rightarrow}(o, \hat{o}) < \tau_a$, then the candidate observation o is redundant and excluded from the memory.

4.3. Semi-Parametric Control

At test time, the graph may not contain the start and end observations. We construct weighted edges to their nearest neighbors in the graph by querying the distance function, ensuring that there is at least one path from the start to the goal. Then, we select waypoints $P = \{o_{w_1}, \ldots, o_{w_p}, o_{\text{goal}}\}$ according to the shortest path from the start to the goal in the weighted graph using Dijkstra's algorithm.

To follow the waypoints, we need to localize the agent in the graph and determine when waypoint $o_{w_{i+1}}$ is reached. SoRB queries the distance function $d(o_{\text{current}}, o_{w_{i+1}})$ and applies a threshold to determine when to switch to the

next waypoint. In the presence of untraversable edges, however, thresholding the distance is insufficient. If edge $(o_{w_i}, o_{w_{i+1}})$ is untraversable, it appeared in the plan precisely because $d(o_{w_i}, o_{w_{i+1}}) \leq \text{MAXSTEPS}$, possibly due to visual similarity. Assuming the agent reached the previous waypoints, the current observation will be similar to o_{w_i} *i.e.* $d(o_{\text{current}}, o_{w_i})$ is small, and we expect $d(o_{\text{current}}, o_{w_{i+1}})$ to be small as well, leading to incorrect localization.

For a stricter waypoint check, we take a similar approach to successor consistency $C_{\rightarrow}$, but we make the check maximally strict by considering distance discrepancies on all of $\mathcal{V}$. Specifically, we measure distance from our current observation o_{current} to our waypoint $o_{w_{i+1}}$ by the maximum discrepancy in distance functions, considering a waypoint as reached when

$$\max_{u \in \mathcal{V}} |d(o_{\text{current}}, u) - d(o_{w_{i+1}}, u)| < \text{ACTINGCUTOFF}. \tag{4}$$

As the maximization is taken over all $u \in \mathcal{V}$ including $o_{w_{i+1}}$, the discrepancy is at least $d(o_{\text{current}}, o_{w_{i+1}})$ assuming that $d(o_{w_{i+1}}, o_{w_{i+1}}) = 0$, and is zero when $o_{\text{current}} = o_{w_{i+1}}$. While test-time localization is a bottleneck to the performance speed of our method, precomputing pairwise distances in the graph, batching distance function evaluation for o_{current} at test time, and evaluating the L_∞ norm of the vector of distance differences improves speed at runtime.

4.4. Graphical Memory Cleanup

Once we have constructed our node-sparse graphical memory $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, a key remaining challenge is having an accurate set of feasible transitions $\mathcal{E}$. The distance function $d(\cdot, \cdot)$ that determines $\mathcal{E}$ is learned and does not perfectly characterize the capabilities of the short-horizon controller. Even one untraversable edge, however, can be exploited by our planner as a so-called "wormhole."

We propose two methods to refine the edge set: k-nearest filtration and walk-through dynamic graph cleanup. The first is a simple, inexpensive procedure that we experimentally found removes many of the initial faulty edges, and the second is a more expensive second pass that aims to remove any faulty edges that still remain, allowing the graph to be corrected in a self-supervised manner.

We minimize errors in the memory by filtering edges, limiting nodes to their k nearest successors. In k-nearest edge filtration, we retain only the edges in $\mathcal{E}$ that are among the k outgoing edges of smallest weight for some node. Letting $\mathcal{E}^{i,j}_{\text{closer}}$ specify the set of edges outgoing from node o_i with distance less than $d(o_i, o_j)$ for $(o_i, o_j) \in \mathcal{E}$, *i.e.*,

$$\mathcal{E}^{i,j}_{\text{closer}} = \{(o_i, o_u) \in \mathcal{E} : d(o_i, o_u) < d(o_i, o_j)\},$$

our k-nearest filtration procedure yields

$$\mathcal{E}_{\text{filtered}} = \left\{(o_i, o_j) \in \mathcal{E} : \left|\mathcal{E}^{i,j}_{\text{closer}}\right| < k\right\}. \tag{5}$$

After filtration, the worst-case number of untraversable edges grows only *linearly* in the sparsified node count, not quadratically.

However, untraversable edges will remain after filtration—edges are created due to inaccurate $d(\cdot, \cdot)$, and filtration relies on the same distance function. In our experiments with visual observations, we find that untraversable edges meeting the distance threshold often connect distant yet visually similar locations, such as head-on views of a wall, the perceptual aliasing problem.

To correct the memory, ground-truth traversibility information is needed. Thus, we correct the graphical memory through environment walkthroughs. In graph cleanup, we reset the environment, sample a goal $o_{\text{goal}} \in \mathcal{V}$ from the agent's memory, plan a path $o_{w_1}, \ldots, o_{w_p}$ to the goal, and follow the low-level controller in the environment to traverse the path as described in Section 4.3. During execution, we deem an edge $(o_{w_i}, o_{w_{i+1}})$ to be infeasible if the agent (a) previously reached waypoint o_{w_i} and (b) does not reach an observation consistent with endpoint $o_{w_{i+1}}$ after a fixed number of actions, ATTEMPTCUTOFF. We then mark $(o_{w_i}, s_{w_{i+1}}) \in \mathcal{E}_{\text{filtered}}$ as failed, adding it to the initially empty set $\mathcal{E}_{\text{failed}}$, and replan according to the updated edges

$$\mathcal{E}_{\text{cleaned}} = \mathcal{E}_{\text{filtered}} \setminus \mathcal{E}_{\text{failed}}. \tag{6}$$

Once the goal waypoint o_{goal} is reached, the cleanup procedure is repeated until a time-limit is reached or $\mathcal{E}_{\text{cleaned}}$ reaches a steady state.

5. Experimental Setup

We evaluate SGM under two high-level learning frameworks: reinforcement learning (RL), and self-supervised learning (SSL). As a general data structure, SGM can be paired with any learned image features, asymmetric distance metric, or low-level controller. However, some learning methods are better suited to particular environments. Below, we describe our training procedure in detail.

Environments We benchmark against the two available environments used by the SoRB and SPTM baselines, and an additional visual navigation environment. These range in complexity and are shown in Figure 5. With RL, we run our experiments on PointEnv, a maze environment used for experiments in SoRB (Eysenbach et al., 2019) with (x, y) coordinate states. We increase the difficulty of this environment by thinning the walls in the maze, which exposes errors in the distance metric since two nearby coordinates may be on either side of a maze wall. SoRB also ran visual

experiments on the SUNCG houses data set (Song et al., 2017), but these environments are no longer public.

To evaluate SGM in image-based environments, we use the ViZDoom navigation environment and pretrained networks from SPTM. In addition, we evaluate navigation in the OpenAI Safety Gym (Ray et al., 2019). In both environments, the graph is constructed over *visual first-person view observations* in a large space with obstacles, reused textures, and walls. Such observations pose a real challenge for learning distance metrics, since they are both high-dimensional and perceptually aliased: there are many visually similar images that are temporally far apart.

Distance metric We explore two learning methods to acquire the asymmetric, temporal distance metric used for graph sparsification and localization. Within the RL framework, PointEnv gives sparse reward $r = 0$ if the goal is reached and $r = -1$ at all other steps. Thus, the undiscounted return is the number of steps taken to reach the goal. Distances can be approximated using the undiscounted $\gamma = 1$ goal-conditioned action-value function according to $d(o_i, o_j) = -\max_a Q(o_i, o_j, a)$. We further increase the robustness of the distance function by using an ensemble of critics and distributional Q-values (Bellemare et al., 2017) as proposed in SoRB.

Distance metrics can also be learned entirely offline via SSL. One advantage of this is decoupling the distance metric from the controller, allowing flexibility in controller design. According to Savinov et al. (2019), we collect rollouts by acting randomly in the environment, sample pairs of observations o_i, o_j from the same or different rollouts, and learn a Siamese binary classifier that predicts whether o_j occurs within the next k steps following o_i on a rollout. SPTM uses the predicted probability of negative as a distance. We interpret this as the probability an edge is untraversable, *i.e.* $p((o_i, o_j) \notin \mathcal{E})$. In our experiments in ViZDoom, we use $d(o_i, o_j) = -\log p((o_i, o_j) \in \mathcal{E})$ for the natural probabilistic interpretation (which we derive in the supplement) that a shortest path with such weights minimizes the probability of there being any error along the path.

The accuracy of the distance function can be further improved with a contrastive objective as proposed by Liu et al. (2020) to learn an encoder $d_\theta(o) : \mathcal{O} \to \mathcal{Z}$ that temporally clusters observations. In SafetyGym, we extract observation triplets $(o_a, o_+, \{o_-\})$ from the random rollouts which contain an anchor o_a, a temporally close o_+, and multiple temporally far negatives $\{o_-\}$. We then construct edges and plan using the exponentiated score $\exp\left(-\alpha d_\theta(o_i, o_j)\right)$ as a probability measure, where α is a temperature parameter.

Perceptual Features Perceptual consistency check features can be shared with the distance function, as in ViZDoom, where we use the pretrained ResNet-18 backbone of the Siamese architecture as-is. In our SafetyGym experiments, we use a β-VAE (Higgins et al., 2017; Kingma & Welling, 2014) trained to reconstruct observations to extract visual features. We found that the β-VAE was sufficiently expressive to extract effective features for SGM's perceptual consistency check. Our RL experiments use the identity.

Figure 5. The three environments used for testing SGM. PointEnv is a small maze with coordinate observations. We increase its difficulty by thinning the walls. ViZDoom is a large environment, which can take up to 5 minutes and 5k steps to traverse entirely. ViZDoom actions are discrete and observations are first-person camera views. SafetyGym is another large environment with first-person view obsercations, and supports continuous actions.

Low-level Controller For the RL experiments, we use actor-critic methods to train a low-level controller (the actor) and corresponding distance metric (the critic) simultaneously. In particular, we use distributional RL and D3PG, a variant of deep deterministic policy gradient (Barth-Maron et al., 2018; Lillicrap et al., 2016). For experiments on SafetyGym, we use a proprioceptive state-based controller for both SGM and the dense baseline. For the SSL experiments in ViZDoom, we use the trained, behavior cloned visual controller from SPTM. The controller is trained to predict actions from a dataset of random rollouts, where goals are given by achieved observations.

6. Results

We investigate the role of sparse graphical memory in self-supervised learning and reinforcement learning setups across three different environments—namely, PointEnv, ViZDoom, and SafetyGym—with the following questions.

6.1. How does sparsity influence success rate?

We hypothesize that sparsity should improve robustness of plans relative to dense methods due to the removal incorrect edges in the graph. We test this hypothesis across all three environments. For PointEnv, we construct a dense graph

TECHNIQUE	SUCCESS RATE	CLEANUP STEPS	OBSERVATION	ENV
SORB	28.0 ± 6.3 %	400K	PROPRIO	POINTENV
SORB + SGM	**100.0 ± .1** %	400K	PROPRIO	POINTENV
SPTM	39.3 ± 4.0%	-	VISUAL	VIZDOOM
SPTM + SGM	**60.1 ± 4.0%**	114K	VISUAL	VIZDOOM
CONSPTM	68.2 ± 4.1 %	1M	VISUAL	SAFETYGYM
CONSPTM + SGM	**96.6 ± 1.5%**	1M	VISUAL	SAFETYGYM

Table 1. SGM boosts performance across all existing state-of-the-art semi-parametric graphical methods.

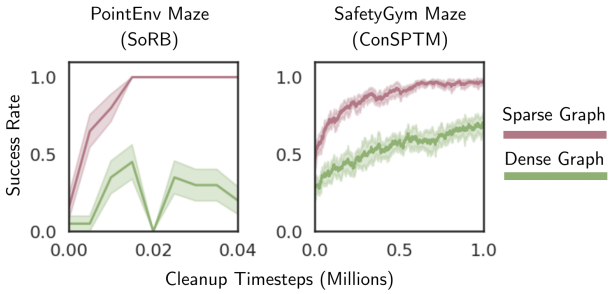


Figure 6. Success rate as a function of cleanup steps in PointEnv (FourRooms maze) and Safety Gym. SGM is rapidly corrected while SoRB, because of errors in its dense graph, is infeasible to clean. SPTM can be cleaned, but only slowly.

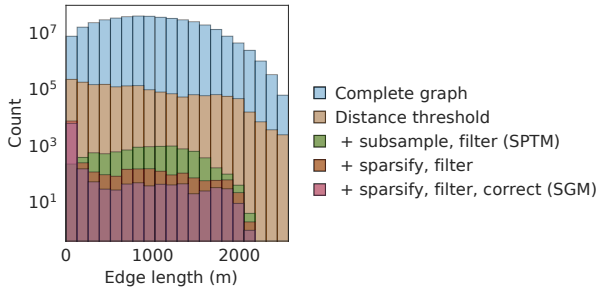


Figure 7. Overlayed histograms of edge lengths for graphical memories in ViZDoom, where long edges are incorrect. Graph quality significantly improves with SGM sparsification and cleanup techniques that reduce the frequency of unrealistically long edges.

of 1k nodes with SoRB. For ViZDoom, we use the SPTM procedure to construct the dense graph with $2k$ nodes sampled from random exploration. For SafetyGym, we use the contrastive variant of SPTM (ConSPTM) as a distance metric to construct the graph. In all three cases, we benchmark performance against the sparse complement of these graphs constructed with SGM.

We show success rates for reaching randomly sampled goals in Table 1. SGM improves performance across all three environments and learned distance metrics. This shows that SGM is a general method that can be used to augment any dense semi-parametric graphical memory regardless of the exact distance metric used.

6.2. How quickly can graph errors be corrected?

Although sparsification is the largest contributor to the removal of faulty edges, a small number of incorrect connections in the graph can still result in "wormhole" connections and lead to a faulty plan. For this reason, cleanup rollouts are also crucial to generating robust plans. We investigate the time that it takes to clean up a graph and hypothesize that sparse graphs can be cleaned faster than dense ones.

To do so, we run clean up rollouts on both dense and sparse graphs on two environments. As before, we employ the SoRB framework for PointEnv and ConSPTM for SafetyGym. Success rate curves shown in Figure 6 show that sparse graphs converge on optimal plans much faster than dense ones with the same number of cleanup steps. The reason for quick convergence is that there are less edges to traverse and therefore less errors to clean up in a sparse graph than its dense complement. Moreover, since SGM can yield arbitrarily sparse graphs, cleanup is a simple but general method for robustness to faulty edges.

6.3. How does performance scale with task difficulty?

We study how (a) task success rates and (b) solution efficiency scale as task difficulty increases in the ViZDoom visual maze navigation environment. We define three difficulty levels: easy goals within 200 m of the agent starting state, medium goals from 200-400 m, and hard goals from 400-600 m. We allow 100, 200, and 300 environment steps, respectively. The same starts and goals are used across baselines, and as in our previous experiments, the goal is defined from a first-person view observation.

In Table 2, we show that random action and self-supervised controller baselines have poor success rate when goal distance increases, $\leq 18.5\%$. The SPTM baseline is also unable to scale to difficult goals, with similarly low success rate. In contrast, SGM achieves the highest success rates, nearly doubling success rate of the next-best for medium goals. Further, in Figure 8, SGM uses the least number of steps in these rollouts, finding efficient plans.

Technique	Easy $\leq$ 200 m	Medium $\leq$ 400 m	Hard $\leq$ 600 m
Random actions	58.0%	21.5%	12.0%
Visual controller	75.0%	34.5%	18.5%
SPTM, subsampled observations	70.0%	34.0%	14.0%
SPTM + SGM + 54K cleanup steps	88.0%	52.0%	**26.0%**
SPTM + SGM + 114K cleanup steps	**92.0%**	**64.0%**	**26.0%**

Table 2. Success rate versus goal difficulty in ViZDoom visual maze navigation

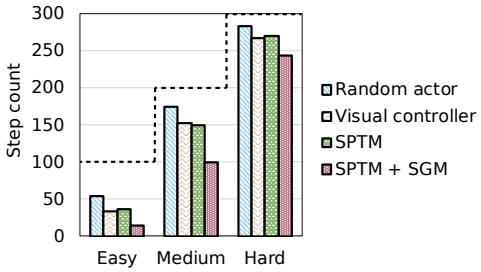


Figure 8. Average path length in ViZDoom

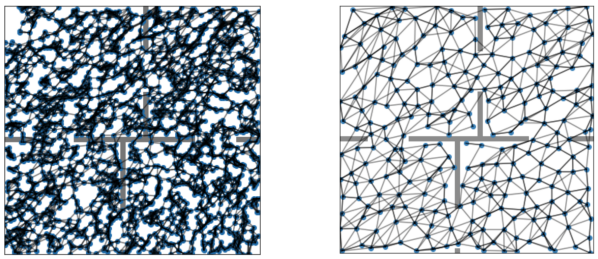


Figure 9. Qualitative comparison of PointEnv graphical memory before and after our proposed sparsification technique.

It is worth noting that in the original SPTM implementation, the test time graph was bootstrapped with a human-provided walkthrough of the maze, which provided ground truth distance information. In our experiments, we do not provide any walkthroughs, allow much shorter rollouts (100-300 vs 5000 steps), and do not restrict goals to a few designated objects. These difficulties cause many errors in the initial graphical memory, but sparsification and some cleanup significantly improve success and improve plan efficiency. We hypothesize that fewer steps are needed due to fewer waypoints that allow more deviation by the agent.

6.4. How many errors are in the graphical memory?

We proposed sparsification, nearest-neighbor and edge cleanup mechanisms to limit the errors in a graphical memory. In this section, we ablate these techniques and study the quality of the resulting graph. It is computationally infeasible to count errors by attempting to traverse each edge in a large baseline graph. However, edges that connect distant observations in a graph are largely untraversable. For example, in Section 6.3, we showed that the visual controller has difficulty reaching distant goals. In this experiment, we compute the frequency of edges of each length using the ground-truth distance between endpoints.

Figure 7 shows that the graphs of all baselines are dominated by long, mostly untraversable edges. Even after subsampling the replay buffer, SPTM is dominated by long, incorrect edges. In contrast, our proposed sparsification of the graph with perceptual and two-way acting consistency retains short edges while minimizing long edges. Graph cleanup for 114K steps removes many remaining errors.

METHOD	TIME TO TAKE ACTION (S)
SORB	0.550 ± 0.220
SGM (OURS)	$\mathbf{0.077 \pm 0.004}$

Table 3. The average and stdev wall-clock time for taking an action with SGM (our method) and SoRB (previous state-of-the-art).

6.5. How efficient is graphical planning?

In real-world settings, runtime can make or break an algorithm. To test the efficiency of SGM, we evaluate against SoRB the total time it takes to act, including localization of new observations, forming a plan with Dijkstra's algorithm, and using the low-level controller. Table 3 shows that SGM is over seven times faster to act than SoRB. In our experiments, we found that querying the distance function to localize new observations, which has to be done at every time step, was the main performance bottleneck. Because SGM is sparse, this localization check is more efficient, making SGM significantly faster than the previous state-of-the-art.

7. Conclusion

In this work, we proposed a new data structure: an efficient, sparse graphical memory that allows an agent to consolidate many environment observations, model its capability to traverse between states, and correct errors. In a range of difficult visual and state-based navigation environments, we demonstrate significantly higher success rates, shorter rollouts, and faster execution over dense graph baselines and learned controllers. We hope that this direction of combining classic search-based planning with modern learning techniques will enable efficient approaches to long-horizon sensorimotor control tasks. In particular, we see scaling sparse graphical memory to challenging manipulation tasks as a key outstanding challenge for the future work.

8. Acknowledgments

This work was supported in part by Berkeley Deep Drive (BDD), ONR PECASE N000141612723, Open Philanthropy Foundation, NSF CISE Expeditions Award CCF-1730628, DARPA through the LwLL program, the DOE CSGF under grant number DE-SC0020347, the NSF GRFP under grant number DGE-1752814, and Komatsu.

Appendix

A. Environments & Hyperparameters

PointEnv: PointEnv is maze environment introduced in (Eysenbach et al., 2019) where the observation space is proprioceptive. We run all SoRB experiments in this environment. The episodic return is undiscounted $\gamma = 1$, and the reward is an indicator function: $r = 0$ if the agent reaches its goal and $r = -1$ otherwise. The distance to goals can thus be approximated as $d = |Q(s, a)|$. We approximate a distributional Q function, which serves as a critic, with a neural network that first processes the observation with a 256-unit, fully-connected layer, that then merges this processed observation with the action, and that then passes the observation-action combination with another 256-unit, fully-connected layer. For an actor, we use a fully-connected network that has two layers of 256 units each. Throughout, we use ReLU activations and train with an Adam optimizer (Kingma & Ba, 2015) with a step size of 0.0003. To evaluate distances, we use an ensemble of three such distributional Q functions, and we pessimistically aggregate across the ensemble.

As hyperparameters for SGM, we use MAXDIST $= 10$ as the threshold for drawing edges, $\tau_p = 0.05$ as the perceptual consistency threshold, $\tau_a = 5$ as the acting consistency threshold, $k = 5$ during k-nearest filtration, ACTINGCUTOFF $= 1$ for localization, and MAXSTEPS $=$ 30 during cleanup. As hyperparameters for SoRB, we use MAXDIST $= 6$ as the threshold for drawing edges, $k = 5$ during k-nearest filtration, and MAXSTEPS $= 18$ during cleanup. Following (Eysenbach et al., 2019), we localize to waypoints with SoRB by querying the distance function from our current observation to our waypoint, considering a goal as reached if the distance to it is below MAXDIST $= 6$.

ViZDoom: For our ViZDoom visual maze navigation experiments, we use the large training maze environment of (Savinov et al., 2019). The distance metric is a binary classifer trained with a Siamese network using a ResNet-18 architecture. The convolutional encoder embeds observations into a 512 dimensional latent vector. Two observations are then concatenated and passed through a 4 layer dense network with ReLU activations, 512 hidden units, and a binary cross entropy objective where $y = 1$ if the two observations are temporally close and $y = 0$ otherwise. An

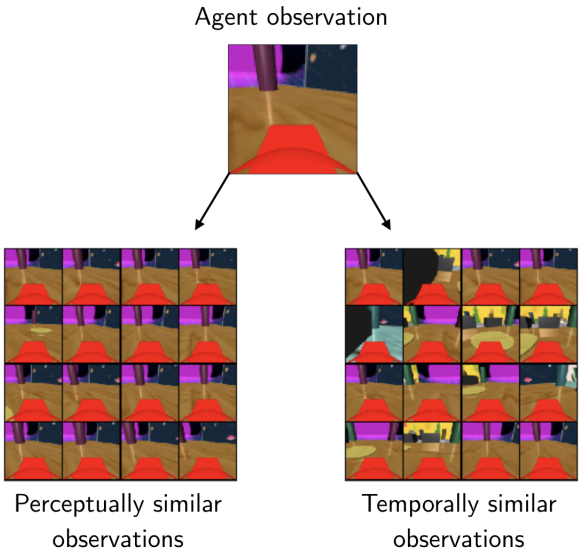


Figure 10. A visual example of perceptual and acting consistency checks for SafetyGym using a β-VAE for visual features and the contrastive objective for temporal features. The temporally nearby observations are, unsurprisingly, more diverse than visually nearby ones. Although the majority of temporally clustered observations are correct, there is one false positive (blue floor) that would fail the consistency check in SGM and therefore would not be merged.

Adam optimizer (Kingma & Ba, 2015) with a step size of 0.0001 is used for gradient updates to finetune the pretrained network of (Savinov et al., 2019). For the controller, we use the pretrained network of (Savinov et al., 2019) with no finetuning.

For graph creation, we first collect a replay buffer of 100 episodes of random actions, each consisting of 200 steps (*i.e.* 20,100 observation images) and add each observation sequentially to SGM to simulate an online data collection process. To make the SPTM baseline tractable, we randomly subsample the replay buffer to 2,087 observations (the same size as our sparsified vertex set), as edge creation is $O(|\mathcal{V}|^2)$ and start/goal node localization is $O(|\mathcal{V}|)$. The baseline graph has $2,087$ nodes and $18,921$ edges. While we can evaluate the baseline with a graph consisting of all $20,100$ observations, this is dense oracle that has $20,100$ nodes and $1,734,524$ edges and takes hours to construct. The oracle achieves $75\%, 55\%$, and 35% success rates at easy, medium and hard goal difficulties ($55.0\% \pm 6.4\%$ overall).

When creating edges for both the baseline and SGM, we set MAXDIST $= 2$ and limit nodes to having $k = 5$ successors during k-nearest neighbor filtration. To localize against waypoints, we set ACTINGCUTOFF $= 5.75$, and to delete an edge during cleanup, we use MAXSTEPS $= 10$. For SGM node sparsification, our perceptual and two-way consistency node merging cutoffs are $\tau_p = 20$ and $\tau_a = 2$.

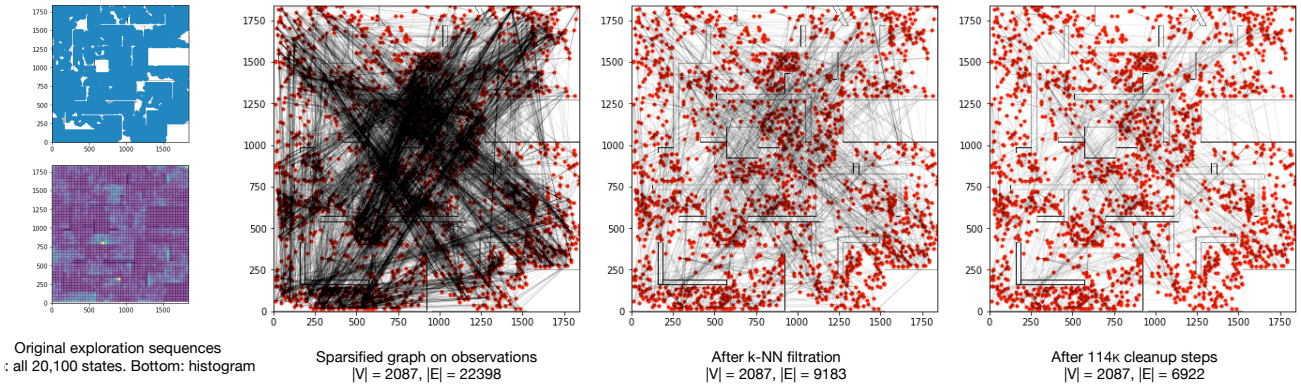


Figure 11. Construction of Sparse Graphical Memory in the ViZDoom environment. We add nodes from a source replay buffer that unevenly covers the environment (left), creating a sparsified memory. k-nearest neighbor edge filtration limits the number of errors, which are further corrected via cleanup. Observations in SGM much more evenly cover the environment, even though no coordinate (state) information is used during graph construction.

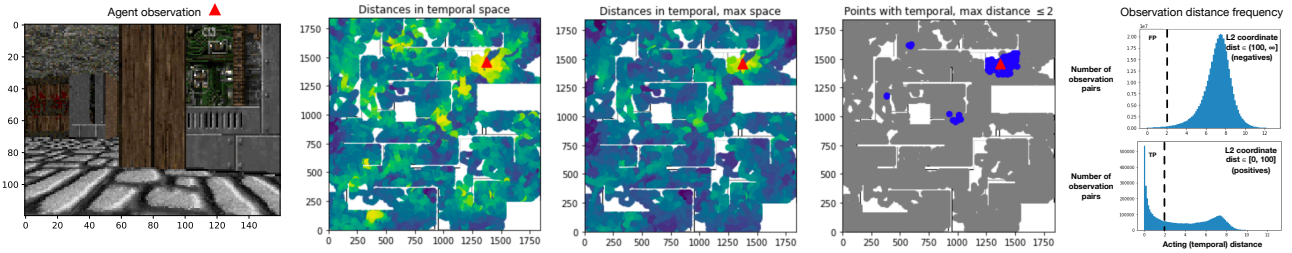


Figure 12. A visualization of the fine-tuned SPTM distance metric in the ViZDoom environment. The agent observes the reference image on left. The second image shows the acting distance between the reference image and observations previously observed throughout the maze according to $d(o_{\text{agent}}, \cdot)$. A coordinate is colored yellow if the associated observation is close to the reference in acting distance, while green and blue coordinates are distant with respect to the reference observation. Aggregating the distance pessimistically across temporal windows (third image) reduces false positives that are distant in coordinate space but close in acting distance space. In the fourth image, we threshold the aggregated distance according to τ_a. While most observations passing the threshold are near the agent, some observations in the maze are distant. These are false positives. In the rightmost figure, we show histograms of the aggregated acting distance for negative, distant pairs of observations (top) and close, positive pairs (bottom), totally 404M pairs.

For all methods, to increase the robustness of edge creation under perceptual aliasing, we aggregate the distance over temporal windows in the random exploration sequence. For observations $o_t^{(i)}$ in episode i and $o_{t'}^{(j)}$ in episode j, we set the distance to the maximum pairwise distance between observations $o_{t-2}^{(i)}, o_{t-1}^{(i)}, o_t^{(i)}, o_{t+1}^{(i)}, o_{t+2}^{(i)}$ and observations $o_{t'-2}^{(j)}, o_{t'-1}^{(j)}, o_{t'}^{(j)}, o_{t'+1}^{(j)}, o_{t'+2}^{(j)}$, aggregating over up to 25 pairs. In contrast, SPTM aggregated with the median and compared only 5 pairs. Our aggregation is more pessimistic as our replay buffer is created with random exploration that suffers from extensive perceptual aliasing rather than a human demonstrator that mostly stays in the center of hallways. We visualize the graph construction process for SPTM in Figure 11.

SafetyGym: In the SafetyGym environment we employ a contrastive objective to discriminate between temporally close observations and random samples from the replay buffer. The contrastive objective is a multiclass cross entropy over logits defined by a bilinear inner product of the form $f(z_t, z_{t'}) = z_t^T W z_{t'}$, where W is a parameter matrix, and the distance scores are probabilities $d = \exp(-f(z_t, z_{t'}))$. To embed the observations, which are 64×64 rgb images, we use a 3 layer convolutional network with ReLU activations with a dense layer followed by a LayerNorm to flatten the output to a latent dimension of 50 units. We then train the square matrix W to optimize the ccontrastive energy function. As before, we use Adam (Kingma & Ba, 2015) with a step size of 0.0001 for optimization.

In SafetyGym experiments, we use a β-VAE maximum likelihood generative model to learn visual features. The β-VAE has an identical architecture to the temporal distance metric but without the square matrix W. Each observation is transformed into its visual embedding, which is stored in the node of the graph. When a new observation is seen, to isolate visually similar neighbors, we compute the L2 distance between the latent embedding of the observation

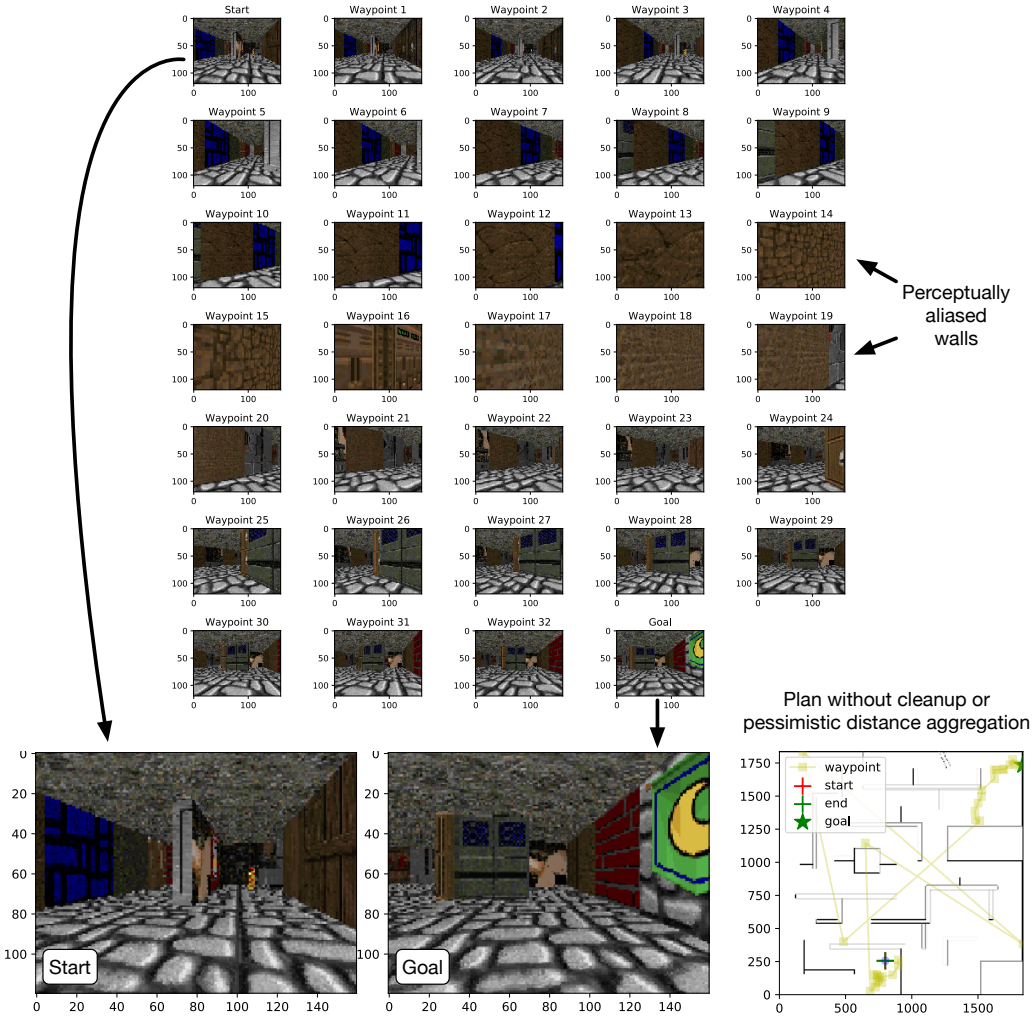


Figure 13. Failure mode in ViZDoom planning when cleanup and pessimistic distance aggregation are not used. While waypoints in the plan between start and goal observations are closely grouped for much of the path, the planner exploits the perceptual aliasing of walls in the environment as a shortcut through the environment. Pessimistic aggregation of the distance metric can help with the issue, but does not fully resolve the problem. By stepping through the environment during cleanup, we can remove the remaining untraversable edges.

and all other nodes in the graph. Since computing the L2 distance is a simple vector operation, it is much more computationally efficient than querying the distance function, which requires a forward pass through a neural network, $O(|\mathcal{V}|)$ times at each timestep.

Visually and temporally clustered observations relative to an agent's current observation are shown in Figure 10. For constructing both dense and sparse graphs, we use $tau_p = 6$ and $tau_a = 5$ for consistency check cutoffs as well as MAXSTEPS $= 9$ for drawing edges and a nearest-neighbor filter of $k = 6$.

B. Perceptual Aliasing with Learned Distance

A common issue with learning temporal distances from images is perceptual aliasing, which occurs when two images are visually similar but temporally far apart. We examine a heatmap of learned distances in ViZDoom in Figure 12. Although most temporally close observations are correctly clustered around the agent's location, there are several false proximity clusters throughout the map due to visual similarity between parts of the maze. Perceptual aliasing results in wormhole connections throughout the graph where two distance nodes are connected by an edge, which creates an attractor for path planning. We show an example path planned by an agent that is corrupted by perceptual aliasing in Figure 13. In its plan, the agent draws a connection between two visually identical but temporally distance walls, which corrupts its entire plan to reach the goal.

False positives can be reduced further by aggregating the distance pessimistically across temporal windows. However, doing so does not eliminate them altogether. The presence

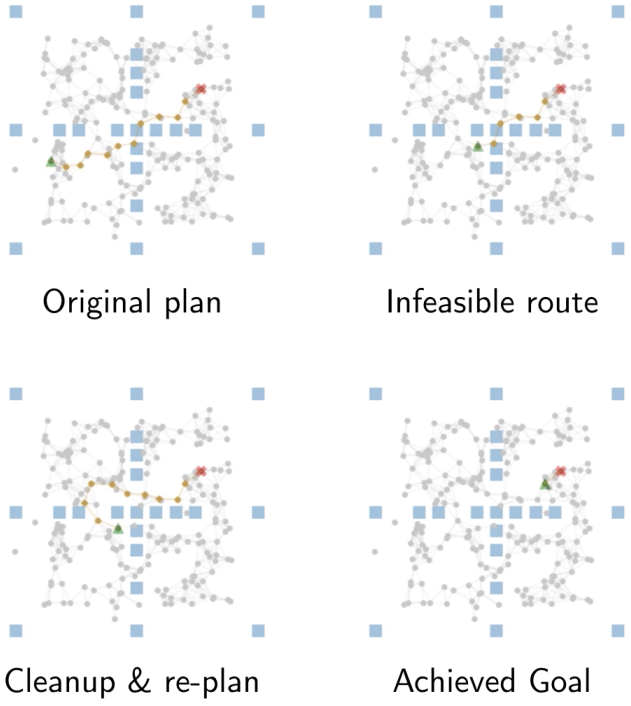


Figure 14. Evaluation of SGM in SafetyGym. This figure is a top-down view abstraction of the SafetyGym environment made to cleanly represent the sparse graph. The actual environment is more visually complex and observations are first-person view images.

of false positives further supports the argument for sparsity. With sparsity and cleanup, it is possible to remove the majority of incorrect edges to yield robust plans.

C. Re-planning with a Sparse Graph

We show an example of an evaluation rollout, which includes a cleanup step when the agent encounters an impossible waypoint in Figure 14. The agent creates an initial plan, moves along the proposed waypoints until it encounters an obstacle. Unable to pass the wall (represented by the blue blocks), the agent removes the edge between two nodes across the wall and re-plans. Its second plan has no obstacles and it is therefore able to reach its goal.

Project video and code are available at https://mishalaskin.github.io/sgm/.

References

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

Bailey, T. and Durrant-Whyte, H. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117, 2006.

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., Tb, D., Muldal, A., Heess, N., and Lillicrap, T. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *ICML*, 2017.

Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

Doran, J. E. and Michie, D. Experiments with the graph traverser program. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 1966.

Durrant-Whyte, H. and Bailey, T. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

Eysenbach, B., Salakhutdinov, R. R., and Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. In *NeurIPS*, 2019.

Ferns, N., Panangaden, P., and Precup, D. Metrics for finite markov decision processes. In *UAI*, 2004.

Foo, P., Warren, W. H., Duchon, A., and Tarr, M. J. Do humans integrate routes into a cognitive map? map-versus landmark-based navigation of novel shortcuts. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 2005.

Garcia, C. E., Prett, D. M., and Morari, M. Model predictive control: theory and practicea survey. *Automatica*, 1989.

Gillner, S. and Mallot, H. A. Navigation and acquisition of spatial knowledge in a virtual maze. *Journal of cognitive neuroscience*, 1998.

Givan, R., Dean, T., and Greig, M. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, 2003.

Hargraves, C. R. and Paris, S. W. Direct trajectory optimization using nonlinear programming and collocation. *Journal of guidance, control, and dynamics*, 10(4):338–342, 1987.

Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2017.

Huang, Z., Liu, F., and Su, H. Mapping state space using landmarks for universal goal reaching. In *NeurIPS*, 2019.

Kaelbling, L. P. Learning to achieve goals. In *International Joint Conference on Artificial Intelligence*, pp. 10941099. Citeseer, 1993.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *ICLR*, 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y. (eds.), *ICLR*, 2014.

LaValle, S. M. Rapidly-exploring random trees: A new tool for path planning, 1998.

Li, L., Walsh, T. J., and Littman, M. L. Towards a unified theory of state abstraction for mdps. In *ISAIM*, 2006.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *ICLR*, 2016.

Liu, K., Kurutach, T., Tung, C., Abbeel, P., and Tamar, A. Hallucinative topological memory for zero-shot visual planning. *arXiv preprint arXiv:2002.12336*, 2020.

Mishkin, D., Dosovitskiy, A., and Koltun, V. Benchmarking classic and learned navigation in complex 3d environments. *arXiv preprint arXiv:1901.10915*, 2019.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. Visual reinforcement learning with imagined goals. In *NeurIPS*, 2018.

Pong, V., Gu, S., Dalal, M., and Levine, S. Temporal difference models: Model-free deep RL for model-based control. In *ICLR*, 2018.

Ray, A., Achiam, J., and Amodei, D. Benchmarking safe exploration in deep reinforcement learning, 2019.

Rubinstein, R. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190, 1999.

Savinov, N., Dosovitskiy, A., and Koltun, V. Semi-parametric Topological Memory for Navigation. *ICLR*, 2019.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *ICML*, 2015.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *ICML*, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Shang, W., Trott, A., Zheng, S., Xiong, C., and Socher, R. Learning World Graphs to Accelerate Hierarchical Reinforcement Learning. *arXiv e-prints*, art. arXiv:1907.00664, Jul 2019.

Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., and Funkhouser, T. Semantic scene completion from a single depth image. In *CVPR*, 2017.

Tolman, E. C. Cognitive maps in rats and men. *Psychological review*, 1948.

Wang, R. F. and Spelke, E. S. Human spatial representation: Insights from animals. *Trends in cognitive sciences*, 2002.