Oracle-SAGE: Planning Ahead in Graph-Based Deep Reinforcement Learning

Andrew Chester (✉)[0000−0003−1662−8663], Michael Dann[0000−0002−7658−022X], Fabio Zambetta[0000−0003−4133−7913], and John Thangarajah[0000−0002−7699−6444]

School of Computing Technologies
RMIT University
`{firstname}.{lastname}@rmit.edu.au`

Abstract. Deep reinforcement learning (RL) commonly suffers from high sample complexity and poor generalisation, especially with high-dimensional (image-based) input. Where available (such as some robotic control domains), low dimensional vector inputs outperform their image based counterparts, but it is challenging to represent complex dynamic environments in this manner. Relational reinforcement learning instead represents the world as a set of objects and the relations between them; offering a flexible yet expressive view which provides structural inductive biases to aid learning. Recently relational RL methods have been extended with modern function approximation using graph neural networks (GNNs). However, inherent limitations in the processing model for GNNs result in decreased returns when important information is dispersed widely throughout the graph. We outline a hybrid learning and planning model which uses reinforcement learning to propose and select subgoals for a planning model to achieve. This includes a novel action selection mechanism and loss function to allow training around the non-differentiable planner. We demonstrate our algorithms effectiveness on a range of domains, including MiniHack and a challenging extension of the classic taxi domain.

Keywords: Reinforcement learning · GNNs · Symbolic planning.

1 Introduction

Despite the impressive advances of deep reinforcement learning (RL) over the last decade, most methods struggle to generalise effectively to different environments [15]. A potential explanation for this is that deep RL agents find it challenging to create meaningful abstractions of their input. Humans conceptualise the world in terms of distinct objects and the relations between them, which grants us the ability to respond effectively to novel situations by breaking them down into familiar components [19,29]. Within the field of reinforcement learning, this approach is best exemplified by relational RL [7]. It has been argued that we now have the tools to combine the power of deep learning with a relational perspective through the use of graph neural networks (GNNs) [2].

While some work exists in this area [14,17,20], much of it is performed on domains (e.g. block tower stability predictions) which have a particular specialised graph representation. Navigational domains are both important in real-world applications (self-driving cars, robot locomotion), and naturally suited to being represented as a graph. For example, a road network can be modelled as a graph with nodes for each intersection and edges for each road. This representation can naturally handle bridges, tunnels, and one-way streets in a way that would be challenging for a standard image based representation to capture accurately.

We hypothesise that naively applying GNNs in RL will be challenging due to their architecture limiting their ability to synthesise information dispersed across long distances in the input graph. A GNN operates by applying *message passing* steps on its input, which limits the effective receptive field for each node to its local neighbourhood [39]. We provide evidence for this hypothesis through a series of experiments on a targeted synthetic domain. Increasing the number of message passing steps indefinitely is not a feasible solution due to increased memory requirements and instability in training [31].

We propose a solution in the RL context by drawing inspiration from recent work in hybrid symbolic planning and RL methods [16,26]. Augmenting the learner with a planning system allows it to integrate data from beyond its receptive field. We illustrate this idea though the following taxi domain which serves as a running example throughout this paper. Imagine being a taxi driver (the GNN-based learner) in a busy city, deciding where to drive and which passengers to take while trying to maximise your earnings for the day. You have available to you a GPS mapping system (the planner), which allows you to plan routes and get time estimates for any destination of your choosing. Interacting with the GPS may change your decisions. For example, you may decide to head to the airport (a subgoal) where you are likely to find paying customers. The GPS however notifies you of a crash on the way and so it will take much longer than usual to get there (feedback). With that additional information, you may change your mind and decide to drive around the nearby streets looking for a passenger instead (the alternate subgoal).

This paper outlines Oracle-SAGE, a hybrid learning and planning approach in which the reinforcement learner proposes multiple subgoals for the planner to evaluate. Once the planner has generated plans to accomplish these subgoals, it returns the projected symbolic states that will result from executing each plan. The discriminator then makes a final decision by ranking these future states in order of desirability, allowing it to revise its subgoals in light of feedback from the planner. The planner thus functions as an oracle, attempting to predict the results of achieving the learner's subgoals. An advantage of this approach is that graph based input combines easily with symbolic planning models; PDDL domains are naturally similar to graphs as they are both object oriented [34].

This proposed architecture poses a number of novel challenges. First, the discriminator needs a way of ranking the projected states; we define the *path-value* as a variant of the value function for future states. Second, the learner has two components: the meta-controller and the discriminator, separated by a non-differentiable planner. In order to train both of these from learned experience, we need to define a *path-value loss function* which allows gradients to propagate around the non-differentiable component. Finally, communication between learner and planner requires both components to share a common symbolic state representation, which we model with graphs. In order to address these, our core contributions in this paper are:

- A novel subgoal (and action) selection mechanism which integrates feedback from a non-differentiable planner.
- A path-value loss function to enable training the action selection network through the non-differentiable planning component.
- A comprehensive evaluation of Oracle-SAGE's effectiveness.

We demonstrate Oracle-SAGE's general purpose effectiveness on a set of complex domains including an extended taxi domain [6] and a subset of the challenging roguelike game Nethack [32]. We benchmark against state-of-the-art learning and hybrid approaches, and show that Oracle-SAGE outperforms all competing methods. We also compare against an ablation which demonstrates the importance of all components of our proposed method.

2 Preliminaries

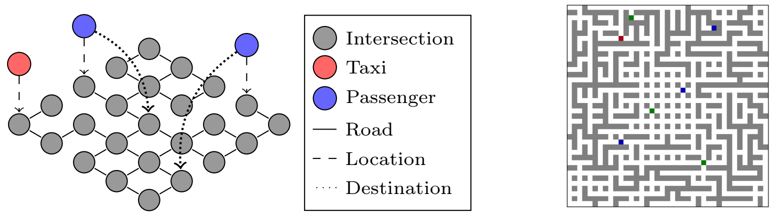
2.1 Reinforcement Learning

A Markov Decision Process (MDP) $\mathcal{M}$ is a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $s \in \mathcal{S}$ are the environment states, $a \in \mathcal{A}$ are the actions, $T : (\mathcal{S} \times \mathcal{A} \times \mathcal{S}) \rightarrow [0, 1]$ is the transition function specifying the environment transition probabilities, and $R : (\mathcal{S} \times \mathcal{A} \times \mathbb{R}) \rightarrow [0, 1]$ gives the probabilities of rewards. The agent maximises the total *return* U, exponentially discounted by a *discount rate* $\gamma \in [0, 1]$: at a given step t, $U_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$, where T is the remaining episode length. The probability of taking an action in a state is given by the *policy* $\pi(a|s) : (\mathcal{S} \times \mathcal{A}) \rightarrow [0, 1]$.

2.2 Symbolic Planning

We use the planning domain definition language (PDDL) to model symbolic planning problems [27]. A planning task consists of a *domain* D, and *instance* N. The domain is $(\mathcal{T}, \mathcal{P}, \mathcal{F}, \mathcal{O})$, where $\mathcal{T}$ are object types, $\mathcal{P}$ are Boolean predicates $P(o_1, \ldots, o_k)$, and $\mathcal{F}$ are numeric functions $f(o_1, \ldots, o_k)$, where $o_1, \ldots, o_k$ are object variables. $\mathcal{O}$ are planning operators, with *preconditions* and *postconditions* (changes to predicates and functions caused by the operator). The instance comprises (B, I, G), where B defines the objects present, I is a conjunction of predicates and functions which describes the initial state, and G is the goal which similarly describes the desired end state. In our domains, the grounded planning operators are equivalent to the set of actions in the underlying MDP, i.e. $\mathcal{A} = \mathcal{O}$, so for simplicity we simply refer to these grounded operators as actions.

We sketch here our transformation of PDDL problems into graphs; it may be applied to any PDDL domain where all predicates and numeric functions have arity no greater than 2. Each object is represented as a node, with object types represented as one-hot encoded node attributes. Unary predicates are also represented as binary node attributes, while unary functions are represented as real-valued attributes. Binary predicates and functions are represented as edge attributes between their two objects. Actions modify the graph by changing node and edge attributes, as well as edges themselves according to the postconditions of the action; for any given action a and current state s, PDDL semantics defines a transition function Δ to compute the next state: $s' = \Delta(s, a)$. See figure 1a for a visual example of the taxi domain as a graph.



(a) Graph representation (b) Image representation

Fig. 1. Taxi domain representations. In the image, passenger destinations are in green, the roads are in white, the taxi is red, and passengers are blue.

2.3 Graphs and Graph Neural Networks

In this article we assume the state is encoded as a directed multi-graph: $s = (\mathcal{V}, \mathcal{E}, u)$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges, and u is the global state; represented as a special node with no edges. All nodes $v_i \in \mathcal{V}$ and u have types and attributes associated with them, represented as a real-valued vector as described above. Similarly edges $e_j \in \mathcal{E}$ have types and attributes, and there may be multiple edges of different types between the same two nodes: $e = (v_s, v_d, e_a)$, where v_s, v_d are the source and destination nodes respectively. The length of the attribute vectors for nodes and edges depends on the domain. Computation over these graphs is done with a Graph Network (GN) [2], a framework which generalises the most common GNN architectures. As our GN functions are implemented by neural networks, we use the term GNN throughout. We denote the output of the GNN for each node, edge, and global state by v_i', e_j', u' respectively, and refer to the entire set of nodes as $\boldsymbol{v'}$.

3 GNN Information Horizon Problem

As mentioned in the introduction, we hypothesise that GNNs perform poorly when important information is distributed widely throughout the graph. In this section, we formalise this problem after providing intuition on why it manifests. We introduce a domain that is targeted to display this failure mode, and empirically demonstrate that RL methods using standard GNNs do not effectively learn, but Oracle-SAGE does.

A GNN is composed of blocks which perform a *message passing* step on their input, producing updated embeddings for edges and nodes. Formally; each block consists of edge, node and global update operations:

$$e'_j = \Phi^e(v_{s_j}, v_{d_j}, e_j, u) \tag{1}$$

$$v'_i = \Phi^v(v, \rho^{e\to v}(e'_j), u) \tag{2}$$

$$u' = \Phi^u(\rho^{v\to u}(v'_i), \rho^{e\to u}(e'_j), u) \tag{3}$$

where $\Phi^{\{e,v,u\}}$ are arbitrary functions; in this work we use a single FC layer which takes the concatenated arguments as input. Similarly, $\rho^{x\to y}$ are aggregation functions which operate over an arbitrary number of inputs; we use elementwise max. It should be noted that $\rho^{e\to v}$ only aggregates over edges which are adjacent to the node v_i; $\rho^{e\to u}$ and $\rho^{v\to u}$ are global and aggregate over all edges/nodes respectively. Multiple blocks of this form are stacked to form the GNN.

From the above update equations, it can be seen that in any single message passing step, each node can only process information from its immediate neighbours. This limits the effective *receptive field* of any node to other nodes within h steps, where h is the number of blocks. In a navigational context, this limits the ability to determine connectivity between points greater than a distance of h away, or to aggregate information along a path greater than length h. This is the *information horizon problem*; for a given GNN architecture there is a limited horizon beyond which node level information cannot propagate. The planner in our model is not subject to this information horizon, and should therefore be able to accurately predict future states using the entire state information.

3.1 Synthetic Domain

We construct a synthetic bandit-like domain [4] designed to test this hypothesis. An agent chooses which of c corridors should be traversed (Figure 2). Along each corridor are l spaces, each of which contains a number of green (positive) or red (negative) tokens. The agent collects all tokens in rooms it traverses, and the final reward is equal to the number of green tokens minus the number of red tokens in the agents possession. Unlike in a traditional bandit problem, the number of tokens in every room is randomly generated at the start of each episode; to allow learning, the environment is fully observable. The agent selects a corridor by applying a softmax layer to the output of the final node in each corridor. Since there are no further choices to make after choosing a corridor, the agent moves directly to the end in a single action. Conceptually, the optimal policy is simple: sum the tokens for each path and select the one which has the highest number of green minus red.

We hypothesise, as per the information horizon problem, that a GNN should be able to learn an optimal policy as long as the number of message passing steps (h) in the GNN is at least l. For $h < l$, the information from the entire corridor will be unable to propagate through the network to be integrated into a single node for action selection. Instead, the best the GNN can do is to choose the most promising path based on the first h steps. To demonstrate this, we use SR-DRL as an example of an RL agent with GNN-based processing which is subject to the information horizon problem [17]. By contrast Oracle-SAGE (full details in section 4) is provided with a planning model that, when given a goal to reach the end of the corridor, can project the agent to the end of the corridor with the number of red and green tokens it would collect on the way. Crucially however, the planning model does not know that the final reward is equal to green minus red tokens and so cannot by itself be used to choose the best action.

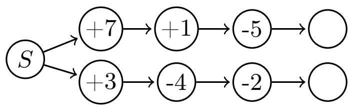


Fig. 2. Synthetic domain with $c = 2$ and $l = 3$. This is the input state graph given to the agents, not a representation of the MDP itself.

3.2 Synthetic Domain Results

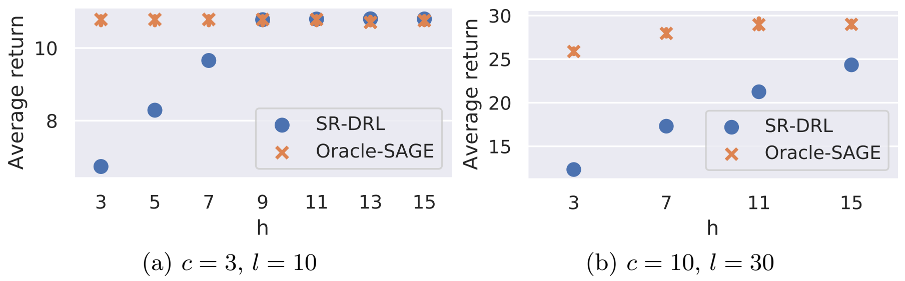


Fig. 3. Converged scores for small (a) and large (b) synthetic domain

Figure 3 shows the performance of both methods as the number of message-passing steps (h) is varied. The results in the small setting provide compelling evidence for our hypothesis. For $h \geq l$, SR-DRL performs well as it can integrate information over the entire path. However, for $h < l$ the return degrades gradually as a path that may look promising in the first h steps may then have large numbers of red tokens later. On the other hand, Oracle-SAGE performs near optimally regardless of the GNN horizon as it sends the top 3 choices to the planner to evaluate, and so when $c = 3$ the discriminator can choose directly between all options. While this shows that Oracle-SAGE addresses the information horizon problem, it may not reflect results for most environments where the number of possible goals exceeds 3.

In the large setting we see that Oracle-SAGE is no longer optimal with a short horizon, as the planner can only evaluate a fraction of the total subgoal possibilities. Nevertheless, it still outperforms SR-DRL for a given horizon length since it evaluates the top 3 promising paths. For $h = 3$; the meta-controllers first choice (which would be SR-DRL's action) is only chosen 41% of the time, in the remainder the increased information from the planner results in a revision to the chosen subgoal. This performance gap suggests that Oracle-SAGE may perform better than SR-DRL in more complex environments, even when the number of possible subgoals is large. With this promise in mind, we now describe it in greater detail.

4 Oracle-SAGE

Oracle-SAGE (Figure 4) combines reinforcement learning with symbolic planning using a shared symbolic graph representation. At a high level, the meta-controller *proposes* k subgoals to the planner. These represent an initial guess of the most promising subgoals to pursue, prior to planning. In our taxi example, a single subgoal could be: "move to location x" or "deliver passenger y". The planner then creates a plan to reach each of these subgoals and *projects* the expected future state of the world after executing the plan (e.g. a new graph with the taxi in the suggested location). These projected future states are then compared by the discriminator to *select* the final plan, which is then executed. Once the plan is complete, losses are calculated and training is performed. We now discuss the planning model itself, and then describe each of these steps in detail.

4.1 Planning Model

Our algorithm requires sufficient knowledge of the environment dynamics to predict future states in *partial* detail. More concretely, we assume that the model is suitable for short-term planning, but not necessarily for long-term planning. Such models are referred to as *myopic* planning models in [5]. For example, in the taxi domain, the model provided might only encompass the local actions of the taxi, e.g. "if you drive along this road, you get to this intersection" and "if you pick up a passenger in your current location, that passenger will be in the taxi". Critically though, it does not know anything about where passengers will appear or what their destinations will be. This makes it feasible to apply our method in domains where the full environment dynamics are complex and unknown, so long as some action consequences are easily specified, such as those introduced in Section 5.

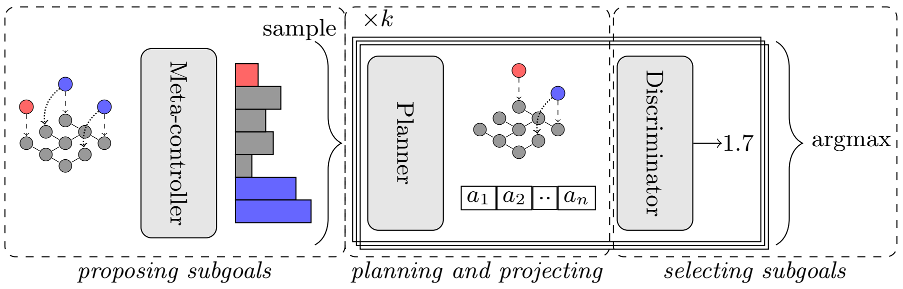


Fig. 4. Oracle-SAGE action selection mechanism. The meta-controller generates a distribution over subgoals, then samples k to be proposed to the planner. For each of the k subgoals, the planner generates a plan and a projected final state. The discriminator then predicts the values of these states and selects the plan with the highest value.

4.2 Proposing Subgoals

The idea of proposing multiple subgoals is motivated by the information horizon problem. While some subgoals (say, delivering a passenger) may look promising to the GNN-based meta-controller, it may be unable to determine the total cost of achieving that subgoal (the time taken to reach their destination). By proposing multiple subgoals to the planner, the discriminator can avoid subgoals which have unexpectedly large costs according to their projected outcome.

Concretely, the meta-controller is an actor-critic RL system that operates at a higher level than the base MDP. The action space for the meta-controller is the set of possible planning subgoals as defined by the planning model. Further details of the semantics of each environment representation can be found in the experiments.

The meta-controller is comprised of a GNN $\mathcal{G}$, the actor π and the critic V. The GNN converts the input state graph into the set of node embeddings and the global embedding: $\boldsymbol{v'}, u' = \mathcal{G}(s; \theta_G)$, where θ_G are the parameters of the GNN. The critic is implemented by a fully-connected layer parameterised by θ_v and takes only the global state as input: $V(u'; \theta_v)$. Finally the actor is implemented by a fully-connected layer parameterised by θ_a, followed by a softmax layer, which gives the policy as a probability distribution over subgoals: $\pi(g|\boldsymbol{v'}, u'; \theta_a)$. Unlike a standard RL agent, the meta-controller samples k subgoals from π without replacement, which are then passed to the planner.

4.3 Planning and Projecting

The planner uses its partial knowledge of the environment dynamics to "look ahead" and predict what would happen if the proposed subgoal were to be achieved. This may include consequences that were not taken into account by the meta-controller due to the information horizon problem. This future state may look less (or more) promising to the discriminator, and it can then select one of the proposed subgoals accordingly.

Specifically, the planner receives k subgoals from the meta-controller and processes them in parallel. For each subgoal g, the planner constructs a planning problem: (s, g) and from this determines a plan $p = [a_0, a_1, \dots, a_{n-1}]$. The plan is then applied step by step to the starting state s_0 with $s_{i+1} = \Delta(s_i, a_i)$. This gives $\tilde{s} = s_n$ as the projected state after the plan has been executed.

4.4 Selecting Subgoal

The role of the discriminator is to select which of the projected future states is best, and hence select the corresponding plan to execute. To do so the discriminator first applies a GNN $\mathcal{G}$ to $\tilde{s}$ to obtain the embedded global state vector: $\tilde{u}' = \mathcal{G}(\tilde{s}; \theta_G)$. This ensures there is a fixed-size representation regardless of the size of the state graph. This GNN shares parameters with the meta-controller; this is optional but improved performance in our experiments.

It then ranks the projected states in order of desirability, taking previously accumulated rewards into account. To explain the intuition here, imagine you have just delivered a passenger to a remote destination and received a large reward. The state immediately following this seems unpromising - you are stranded in the middle of nowhere with no passengers in sight - but the plan itself is a good choice due to the accumulated reward. To address this issue, we define a path value function, $V_p(u', \tilde{u}'; \theta_p)$ which takes as input the (embedded) current state u' and projected future state $\tilde{u}'$. It is trained to predict the expected future reward for being in state s and then (after some number of steps) being in state $\tilde{s}$. The path value function is implemented as a fully-connected layer parameterised by θ_p, which is trained using the path value loss defined below. This process is repeated for each of the k projected states and the plan with the highest path value is chosen for execution.

4.5 Executing Plan

In our environments, the symbolic actions in the plan correspond to the actions in the base MDP, so these are simply executed in sequence until the plan terminates. If the planning model is abstract (i.e. operates at a higher level than the MDP actions), then a low-level RL controller could be trained to achieve each symbolic planning step as in [5,26]. The meta-controller operates on a temporally extended scale; one subgoal might correspond to dozens of atomic actions in the underlying MDP. Consequently the experience tuples we store for an n step plan are of the form $(s_t, \tilde{s}_t, U_{t:t+n}, s_{t+n}, g_t)$, where $U_{t:t+n} = \sum_{i=0}^{n} \gamma^i r_{t+i}$ is the accumulated discounted reward for the plan.

4.6 Training

We use A2C [28] to train our agent, but any policy gradient method could be applied. The overall loss is the sum of four components: the policy loss, value loss, and entropy loss as usual for A2C, as well as the novel path value loss.

$$\mathcal{L} = \mathcal{L}_P + \kappa_1 \cdot \mathcal{L}_V + \kappa_2 \cdot \mathcal{L}_E + \kappa_3 \cdot \mathcal{L}_{PV} \quad (4)$$

where:

- Policy loss: $\mathcal{L}_P(\theta_G, \theta_v, \theta_a) = -\ln(\pi(g_t|\boldsymbol{v'_t}, u'_t; \theta_a)) \cdot A(u'_t, a; \theta_v)$
- Value loss: $\mathcal{L}_V(\theta_G, \theta_v) = (V(u'_t; \theta_v) - (U_{t:t+n} + \gamma^n \cdot V(u'_{t+n})))^2$
- Entropy loss: $\mathcal{L}_E(\theta_G, \theta_a) = H(\pi(g_t|\boldsymbol{v'_t}; \theta_a))$
- Path value loss: $\mathcal{L}_{PV}(\theta_G, \theta_p) = (V_p(u'_t, \tilde{u}'_t; \theta_p) - (U_{t:t+n} + \gamma^n \cdot V(u'_{t+n})))^2$

and κ_i are hyperparameters, H is the entropy, and A is the advantage function.

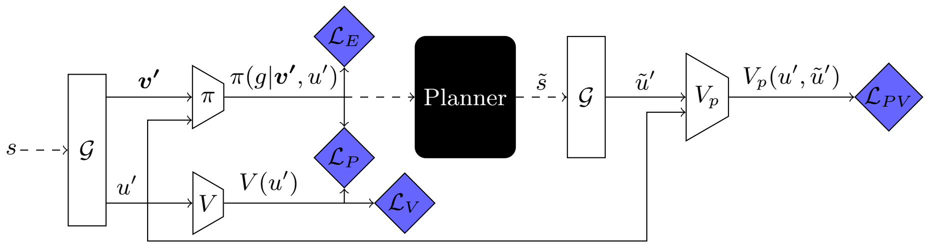


Fig. 5. Visualisation of loss calculations and gradient flows. Rectangles represent GNNs; trapeziums represent FC layers. Blue diamonds are loss components, and the black rectangle is a non-differentiable symbolic planner. Solid lines show computation with gradient flow in the opposite direction, dashed lines do not admit gradients.

5 Experiments

We empirically test two hypotheses: 1. Does Oracle-SAGE mitigate the information horizon problem in complex navigation tasks? 2. Are the state projection and discriminator critical to Oracle-SAGE's success, or is planning sufficient?

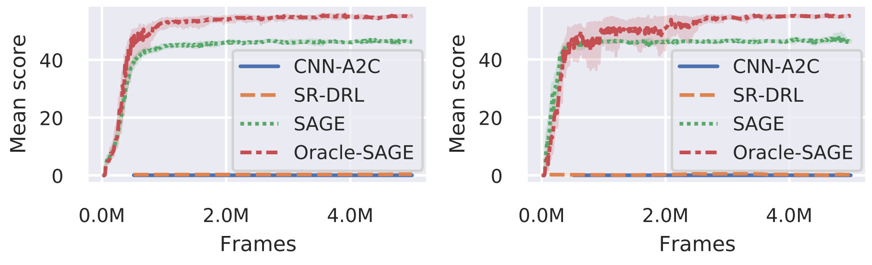
To evaluate the first hypothesis we compare against SR-DRL, a graph based RL model, which suffers from the information horizon problem [17]. To evaluate the second, we compare against an ablation of Oracle-SAGE which proposes a single subgoal to the planner and therefore has no discriminator. This is similar to SAGE, albeit it leverages a graph-based instead of image-based input [5]. For completeness, we also show results for standard RL approaches using image-based representations of our domains; for these, the information horizon problem does not apply, but they do not have the benefit of the semantically richer graph-based representation or planning model [28,32]. Results are averaged over 5 random seeds, and the number of proposed subgoals (k) is set to 3.[1]

5.1 Taxi Domain

We extend the classic Taxi domain [6] as follows. A single taxi operates in a randomly generated 20×20 grid world comprised of zones with different levels

[1] Code available at `https://github.com/AndrewPaulChester/oracle-sage`.

of connectivity (Figure 1b). During each episode, passengers appear in random cells with random destinations, with up to 20 present at once. The taxi receives a reward of 1 for every passenger that is delivered to their destination. As described previously, the planning model can predict the movements of the taxi, but does not know where future passengers will appear, so constructing a single optimal plan for the entire episode is impossible. The subgoal space of the meta-controller is to deliver any passenger or to move to any square. The image based benchmark is a standard CNN-based A2C agent [28].



(a) $h = 5$ (b) $h = 30$

Fig. 6. Taxi domain results. Each line is averaged over 5 seeds, with 95% CI shaded.

Figure 6 shows the results in the taxi domain, we first discuss results with $h = 5$. Neither CNN-A2C or graph-based SR-DRL learn to reliably deliver passengers in this environment. The large grid makes it challenging to randomly deliver passengers, resulting in a very sparse reward environment. Even worse, the randomisation of the maze-like road network at every episode prevents memorisation of a lucky action sequence.

By contrast, SAGE performs quite well, quickly learning to deliver around 46 passengers per episode. This success can be attributed to its ability to construct plans to reach far off subgoals; the planner can handle the low-level navigation reliably. It still falls short of Oracle-SAGE though, which delivers about 20% more passengers on average. By investigating the model choices in more detail, we can see why this occurs. Oracle-SAGE assigns nearly equal probabilities to its top 3 choices, and after receiving the projected future state is approximately equally likely to choose any one of them. However the average length of a plan in Oracle-SAGE is 34 compared to 42 for SAGE. This indicates that Oracle-SAGE's discriminator is learning to choose the shorter plans; i.e. deliver passengers that are closer to the taxi, thereby getting the same reward in a shorter time. The meta-controller is unable to learn to distinguish between these subgoals since the passenger destinations are beyond the information horizon.

The results with a horizon of 30 are largely similar to those with the smaller horizon. We start to see instability in the policies with these deep GNNs, but no sign of any benefit, even though the horizon length is comparable to the average plan length. This may indicate that even when the information is not strictly outside of the GNN horizon, the relevant features are too hard to learn in a reasonable time when compared to the more semantically compact projected state representation. We were unable to train with a horizon longer than 30 due to instability and memory constraints.

5.2 MiniHack Domain

We also show results using MiniHack [32], an environment suite built on top of the dungeon crawling game Nethack [22]. Our custom MiniHack domain consists of a number of rooms connected by randomised corridors, with a staircase in all rooms except the one the agent starts in. Each room contains a couple of stationary traps, and a random subset of rooms in each episode contain deadly monsters. The agent receives a reward at each step of -0.1, with +10 for successfully descending a staircase and -10 for being killed by a trap or monster. The maximum length of each episode is 100 steps. The planning model provided to the agent is similar to that in the taxi domain; it is restricted to movement actions, it has no knowledge of monsters, traps or staircases. The subgoal space of the meta-controller is to move to any visible square, or to fire rocks with the sling present in the players starting inventory. We use RND as described in the original MiniHack paper [32] as the image based benchmark for this domain.

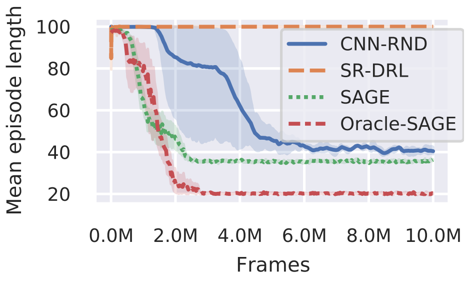


Fig. 7. MiniHack results; deaths count as a length of 100

The results in figure 7 show that Oracle-SAGE outperforms SAGE in this domain. While SAGE can learn to set subgoals of reaching staircases in rooms without monsters, it fails to distinguish between near and far staircases as the distances are beyond the information horizon. As such, it essentially chooses a safe staircase at random, resulting in a higher average episode length and hence lower average score. By contrast Oracle-SAGE can propose a number of different staircases as potential options, and then evaluate the time taken to get to each one according to the projected future state. This allows it to choose the closest safe staircase reliably, improving performance.

SR-DRL fails to learn in this environment. The rooms with stairs often have monsters in them, which makes them very dangerous for agents early in training that are still acting largely randomly. The monsters are likely to kill the agent before it stumbles upon the stairs, and so it learns to avoid being killed by monsters at the cost of never entering any of the rooms with stairs. Due to its intrinsic exploration bonus, RND continues to explore and does eventually learn to reach the closest set of stairs; the other rooms are too challenging to reach even with intrinsic exploration. As a result, it reliably converges to a suboptimal policy: aim for the closest stairs regardless of whether there is a monster present.

6 Related Work

There are two broad paradigms for addressing sequential decision problems: planning and RL [10]. A wide variety of prior work integrates these two strategies, such as model-based RL [12,13,33], learning based planners [9], and hierarchical hybrid planning/learning architectures [16,21,26]. These approaches vary in the amount of information provided to the agent. Model-based RL frequently assumes no information except direct environment interaction, instead learning the models of the environment from scratch. This requires the least effort for human designers, but accurate models often require large amounts of experience to learn. At the other end of the spectrum, planning based approaches often assume access to a perfect environment model [9]. While this can reduce the sample complexity dramatically, it is impractical for many domains of interest as the true environment dynamics are unknown. This work is aimed at a middle ground, where we assume access to a *myopic* [5] model of the world which is suitable for short term but not long term planning. We contrast this with an *abstract* planning model, which is suitable for long term planning but does not contain the necessary details to act directly in the environment.

RL-Planning Hybrids. Much recent work augments RL systems with symbolic planning models to reduce the sample complexity [16,21,23,26]. These all assume access to an abstract planning model and generate a single fixed plan at the start of each episode, often from a human provided goal. These techniques are incompatible with our environments where only myopic models are available, and changes in the environment require replanning during an episode. For example in our taxi domain, to perform well the agent must deliver passengers that are not present at the start of the episode, which is outside of these methods' capabilities. Another branch of work assumes access to an abstract state space mapping, and applies tabular value iteration at the high level to guide a low-level RL policy [30,37,38]. These methods assume that the abstract state space is small, and cannot scale to our domains which require function approximation at the high level. Most similar to our current approach, SAGE assumes only a myopic planning model [5], but lacks feedback from the planner. Finally we note that none of the approaches in this section operate on graph-structured input domains.

Graph-Based RL. We can categorise work that combines GNNs with RL based on the source of the graphs used. Some work assumes this graph takes a special form and is provided separately to the state observations of the agent, such as a representation of the agent's body [36], or a network of nearby agents in a multi-agent setting [18]. In others, the graph is derived directly from the observation itself using a domain-specific algorithm, such as text based adventure games [1]. Finally are those methods in which the graph forms the state observation itself. Some of these have specialised graph representations or action selection mechanisms which restrict their applicability to a single application domain, such as block stacking [14,24] or municipal maintenance planning [20]. Concurrently to our work, Beeching et al. perform navigation over a graph in a realistic 3D environment [3]. They explicitly target scenarios where a symbolic planning model is not applicable, but restrict themselves to pure navigation tasks to reach a provided endpoint, rather than the general reward maximisation objective in our work. Symnet [8] uses GNN based RL to solve relational planning tasks. While this is domain independent, it requires a complete description of the environment dynamics in RDDL (a probabilistic PDDL variant) and so is not applicable in our domains where some environment dynamics are unknown. The most closely related approach to ours is SR-DRL which uses a similar problem set-up, but does not have access to a planning model and so is subject to the information horizon problem [17].

GNN Receptive Field. Our description of the GNN information horizon problem draws on prior work regarding the receptive fields of GNNs outside of the RL context [35]. Some authors have tried to address this by using spectral convolution methods, which aggregate information from a wider neighborhood in a single GNN block [25]. These approaches are computationally intensive and do not generalise well across graphs with different structures [39]. Another approach is to deepen the GNN to expand the receptive field [11,31]. While this is promising, it requires commensurately more resources to train, and as demonstrated in our taxi experiments does not improve performance in our RL setting.

7 Conclusion

GNNs show promise in extending relational RL algorithms with modern function approximation, allowing for object-centric reasoning. In this paper we formalised the GNN information horizon problem in deep RL, and showed empirically on a synthetic domain that it leads to degraded performance for our benchmarks on large graphs. This motivated Oracle-SAGE; a graph-based hybrid learning and planning algorithm which incorporates planning predictions into its decision process to mitigate such a problem. We demonstrated its effectiveness against a range of benchmarks and ablations on an extended taxi and a MiniHack domain.

A limitation of this work is that the top-k action selection and planning projection requires additional computational resources when compared to competing approaches. We have also assumed that the provided PDDL actions map directly onto the base MDP. Including a low-level goal-directed RL controller that works as a layer under the planning model would allow this approach to be applied to a wider set of domains [16,21,26]. Finally, our future work may also investigate interim experience augmentation [5] to increase sample efficiency in domains with long plans.

References

1. Ammanabrolu, P., Riedl, M.: Playing text-adventure games with graph-based deep reinforcement learning. In: NAACL (2019)
2. Battaglia, P., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., Pascanu, R.: Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261 (2018)
3. Beeching, E., Peter, M., Marcotte, P., Debangoye, J., Simonin, O., Romoff, J., Wolf, C.: Graph augmented deep reinforcement learning in the GameRLand3D environment. arXiv preprint arXiv:2112.11731 (2021)
4. Berry, D.A., Fristedt, B.: Bandit Problems: Sequential Allocation of Experiments (Monographs on Statistics and Applied Probability) (1985)
5. Chester, A., Dann, M., Zambetta, F., Thangarajah, J.: SAGE: Generating symbolic goals for myopic models in deep reinforcement learning. arXiv preprint arXiv.2203.05079 (2022)
6. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. JAIR **13**, 227–303 (2000)
7. Džeroski, S., De Raedt, L., Driessens, K.: Relational reinforcement learning. Machine Learning **43**(1), 7–52 (2001)
8. Garg, S., Bajpai, A., Mausam: Size independent neural transfer for RDDL planning. In: ICAPS (2019)
9. Garg, S., Bajpai, A., Mausam: Symbolic network: Generalized neural policies for relational MDPs. In: ICML (2020)
10. Geffner, H.: Model-free, model-based, and general intelligence. arXiv preprint arXiv:1806.02308 (2018)
11. Godwin, J., Schaarschmidt, M., Gaunt, A.L., Sanchez-Gonzalez, A., Rubanova, Y., Veličković, P., Kirkpatrick, J., Battaglia, P.: Simple GNN regularisation for 3D molecular property prediction and beyond. In: ICLR (2022)
12. Ha, D., Schmidhuber, J.: Recurrent world models facilitate policy evolution. In: NeurIPS (2018)
13. Hafner, D., Lillicrap, T., Norouzi, M., Ba, J.: Mastering atari with discrete world models. In: ICLR (2021)
14. Hamrick, J.B., Allen, K.R., Bapst, V., Zhu, T., McKee, K.R., Tenenbaum, J.B., Battaglia, P.W.: Relational inductive bias for physical construction in humans and machines. In: Proceedings of the Annual Meeting of the Cognitive Science Society (2018)
15. Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D.: Deep reinforcement learning that matters. In: AAAI (2018)
16. Illanes, L., Yan, X., Icarte, R.T., McIlraith, S.A.: Symbolic plans as high-level instructions for reinforcement learning. In: ICAPS (2020)
17. Janisch, J., Pevný, T., Lisý, V.: Symbolic relational deep reinforcement learning based on graph neural networks. arXiv preprint arXiv:2009.12462 (2020)
18. Jiang, J., Dun, C., Huang, T., Lu, Z.: Graph convolutional reinforcement learning. In: ICLR (2019)
19. Kemp, C., Tenenbaum, J.B.: The discovery of structural form. PNAS **105**(31), 10687–10692 (2008)

20. Kerkkamp, D., Bukhsh, Z., Zhang, Y., Jansen, N.: Grouping of maintenance actions with deep reinforcement learning and graph convolutional networks:. In: ICAART (2022)
21. Kokel, H., Manoharan, A., Natarajan, S., Ravindran, B., Tadepalli, P.: RePReL: Integrating relational planning and reinforcement learning for effective abstraction. In: ICAPS (2021)
22. Küttler, H., Nardelli, N., Miller, A.H., Raileanu, R., Selvatici, M., Grefenstette, E., Rocktäschel, T.: The NetHack learning environment. In: NeurIPS (2020)
23. Leonetti, M., Iocchi, L., Stone, P.: A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. AIJ **241**, 103–130 (2016)
24. Li, R., Jabri, A., Darrell, T., Agrawal, P.: Towards practical multi-object manipulation using relational reinforcement learning. In: ICRA (2020)
25. Liu, Z., Chen, C., Li, L., Zhou, J., Li, X., Song, L., Qi, Y.: GeniePath: Graph neural networks with adaptive receptive paths. In: AAAI (2019)
26. Lyu, D., Yang, F., Liu, B., Gustafson, S.: SDRL: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In: AAAI (2019)
27. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL - the planning domain definition language. Technical Report, Yale Center for Computational Vision and Control (1998)
28. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
29. Navon, D.: Forest before trees: The precedence of global features in visual perception. Cognitive Psychology **9**(3), 353–383 (1977)
30. Roderick, M., Grimm, C., Tellex, S.: Deep abstract Q-networks. In: AAMAS (2018)
31. Rong, Y., Huang, W., Xu, T., Huang, J.: DropEdge: Towards deep graph convolutional networks on node classification. In: ICLR (2020)
32. Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Kuttler, H., Grefenstette, E., Rocktäschel, T.: MiniHack the planet: A sandbox for open-ended reinforcement learning research. In: NeurIPS Track on Datasets and Benchmarks (2021)
33. Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., Silver, D.: Mastering Atari, Go, chess and shogi by planning with a learned model. Nature **588**(7839), 604–609 (2020)
34. Sievers, S., Röger, G., Wehrle, M., Katz, M.: Theoretical foundations for structural symmetries of lifted PDDL tasks. In: ICAPS (2019)
35. Topping, J., Di Giovanni, F., Chamberlain, B.P., Dong, X., Bronstein, M.M.: Understanding over-squashing and bottlenecks on graphs via curvature. arXiv preprint arXiv:2111.14522 (2021)
36. Wang, T., Liao, R., Ba, J., Fidler, S.: NerveNet: Learning structured policy with graph neural networks. In: ICLR (2018)
37. Winder, J., Milani, S., Landen, M., Oh, E., Parr, S., Squire, S., desJardins, M., Matuszek, C.: Planning with abstract learned models while learning transferable subtasks. In: AAAI (2020)
38. Wöhlke, J., Schmitt, F., van Hoof, H.: Hierarchies of planning and reinforcement learning for robot navigation. In: ICRA (2021)
39. Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications. AI Open **1**, 57–81 (2020)