DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD
DASH.HARVARD.EDU

HARVARD LIBRARY
Office for Scholarly Communication

Machine Learning for Automated Planning

Citation
Zeng, Catherine Yingxuan. 2022. Machine Learning for Automated Planning. Bachelor's thesis, Harvard College.

Permanent link
https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37371729

Terms of Use
This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. Submit a story .

Accessibility

Machine Learning for Automated Planning

Catherine Zeng

A thesis submitted in partial fulfillment
for the degree of Bachelor of Arts in
Computer Science and Mathematics
Harvard College
Cambridge, MA

March 21, 2022

Abstract

Automated planning is a long-standing problem which concerns finding an action sequence to solve a task. In this thesis, we explore two problems in leveraging machine learning for automated planning: (1) learning from failed planning attempts to improve efficiency of future planning, and (2) adding goal-conditioning to action samplers in a neuro-symbolic planning framework. In both problems, we utilize neural networks to learn mappings that empirically enhance the performance of existing planning frameworks.

Acknowledgements

The work in this thesis was made possible by Tom Silver, Professor Leslie Kaelbling, and other members of the Learning and Intelligent Systems group at MIT. I am especially grateful for Tom's patient and thoughtful guidance, as well as for Prof. Kaelbling's insightful advice.

I would also like to thank my thesis readers Professor Finale Doshi-Velez and Professor Michael D. Smith for their invaluable roles as mentors and advisors during my time at Harvard.

Finally, I thank my friends and family for their unwavering love and support.

Contents

1 Introduction **3**
1.1 Methods of planning . . . 3
1.2 Our contributions . . . 5

2 Learning from Failures in Classical Planning **6**
2.1 Related work . . . 6
2.1.1 Learning inadmissible heuristics . . . 6
2.1.2 Dead-end detection . . . 7
2.1.3 Connections to reinforcement learning . . . 7
2.2 Preliminaries . . . 8
2.3 Approach . . . 9
2.3.1 Eliminable Edge Sets . . . 9
2.3.2 Eliminability in Failed Searches . . . 10
2.3.3 Learning to Predict Eliminability . . . 12
2.4 Experiments and Results . . . 13
2.4.1 Experimental Setup . . . 14
2.4.2 Results . . . 15
2.5 Discussion . . . 17

3 Goal-Conditioned Action Sampling in Integrated Task and Motion Planning **18**
3.1 Related work . . . 18
3.1.1 Task and Motion Planning . . . 18
3.1.2 Learning Action Samplers . . . 19
3.1.3 Neuro-Symbolic Relational Transition Models . . . 19
3.2 Problem Setting . . . 20
3.3 Experiments . . . 21
3.4 Results . . . 23
3.5 Discussion . . . 26

4 Conclusions **27**

1 Introduction

Automated planning is a subfield of artificial intelligence which involves finding a sequence of actions to solve a specified task. We can broadly think about automated planning as a set of states, actions, and goals. A state is a representation of the current state of the world, an action is something an agent can perform in order to move from one state to another, and a goal is a set of states that the agent wishes to be in. Thus, a planning problem involves coming up with a series of actions to take the agent from its initial state to a goal state. Planning problems can come in several varieties, depending on attributes such as whether states are continuous or discrete, whether actions are deterministic or nondeterministic, and whether there is full or partial observability of the environment.

Planning is an important question to study both in terms of practical applications and as an area of academic interest. In the real world, planning addresses the question of how an agent can accomplish a complex task without having specified instructions at every step. Self-driving cars use automated planning to navigate busy roads without human guidance [37], and household helper robots use planning to perform various chores [21]. Planning is of theoretical interest because it forms an important part of intelligent behavior. In 1997, IBM created the chess engine Deep Blue using planning strategies to outplay reigning world champion Garry Kasparov in a six-game match, scoring 3.5-2.5 [8]. More recently, in 2016, AlphaGo won against top human player Lee Sedol in a five-game match 4-1, before it defeated top-ranked player Ke Jie 3-0 at the 2017 Future of Go Summit [12][42]. Looking forward, planning will continue to play a key part in enabling artificial intelligence to meet and exceed the capabilities of humans.

1.1 Methods of planning

The classical planning setting assumes a finite state and action space, as well as a deterministic transition function mapping each state and action pair to a new state. Classical planning algorithms are often based on state space search. Search nodes are the states of the planning environment, edges

represent possible state transitions, edge distances represent action costs, and a search algorithm is applied in order to find a path to the goal state. Search algorithms may vary in terms of whether they find an optimal path to the goal state, in which action costs are minimized, or if they simply find a satisficing path.

When a planner does not have any information about the goal's location, uninformed search may be used. Uninformed search algorithms include traditional depth-first search and breadth-first search; depth-first search explores as far along each branch as possible before backtracking, while breadth-first search explores all nodes at the current depth before continuing search at the next depth level. Search may also be guided by a heuristic function, which approximates the distance to the goal from each state. Heuristic searches include greedy best-first search, which expands nodes in the order of their heuristic values, and A^* search, which expands nodes based on the estimated total path length through the node (the sum of the cost to the node and the node's heuristic value). $\alpha - \beta$ search, which is used by Deep Blue, and Monte Carlo Tree Search, used by AlphaGo, are also examples of heuristic search.

In environments with large, continuous state and action spaces, hierarchical planning may be used [30][53][36]. Hierarchical planning divides the problem into various levels of abstraction. The highest level corresponds to a highly discretized space in which solving for a plan to the goal is easy, while the lowest level corresponds to the continuous state and actions of the environment. In the robotics literature, task and motion planning solves robot planning problems using discrete task-level planning and continuous motion-level planning.

The application of classical automated planners to real-world problems can be difficult because of the difficulty of hand-crafting accurate descriptions of the planning task and because of challenges scaling up off-the-shelf planners [24]. These two problems have been addressed by a long and active line of research in machine learning for planning [24]. Approaches in this line include heuristic learning [3, 18, 41, 42, 54] and generalized policy learning [6, 19, 20, 25, 34, 45].

1.2 Our contributions

We continue the work in machine learning for automated planning by exploring two directions. In section 2, we investigate how to learn from failed planning attempts in the classical planning setting. Our main insight is that, by identifying something we call *eliminable edge sets* in previous searches, we can learn a model that predicts what edges can be eliminated in new similar searches. The work in this section was published at the ICAPS 2021 Workshop on Planning and Reinforcement Learning [55]. In section 3, we consider a more challenging setting where states and actions are hybrid discrete/continuous and planning is bi-level. The key question we investigate here is to what extent goal-conditioning can improve action sampling. For each direction of study, we will discuss the most relevant work, our problem setup, our approach, and our empirical findings and analysis. In Section 4, we conclude our explorations with some final thoughts and reflections on the connection between our two directions of study.

2 Learning from Failures in Classical Planning

Our first direction of study takes place in the classical planning setting, where states and actions are finite and state transitions are deterministic. However, even with deterministic transitions, we may fail to realize a plan to the goal if the state and action spaces are large and we have compute or time constraints. We address the question of how past failure experiences can be leveraged to plan more efficiently and effectively in a similar new problems.

Approaches to learning from previous planning experiences almost always rely on successful past experience [24] [25]. In the following related work section, we will describe two methods of learning from failures; each of these methods have limitations that we attempt to overcome.

Specifically, we introduce the new concept of *eliminable edge sets*: sets of edges that can be eliminated from a search graph without changing the solvability of the problem. We show how eliminable edge sets can be identified from failed searches, with particular ease in the case of forward search algorithms like A* and greedy best-first search (GBFS). These edges can then be used to learn to predict eliminable edges in new problems, leading to faster planning.

In experiments, we consider this approach of learning to predict eliminable edge sets in four visual navigation domains, all of which contain reversible actions and therefore lack dead-ends. Our main empirical finding is that planning with the learned edge elimination model considerably outperforms planning with the same algorithm that led to the failed search attempts. We continue with an analysis of the aspects of the domains that enable this strong performance and conclude with a discussion of remaining challenges and open questions in learning from failed search attempts.

2.1 Related work

2.1.1 Learning inadmissible heuristics

Thayer et al. [50] introduced what they called learning inadmissible heuristics during search in a 2011 paper. Assuming that a search heuristic exists during planning, Thayer et al. show that they can reason about the heuristic's error – the difference between the estimates provided by the

heuristic and the true distances between states – to learn a better heuristic during a future planning attempt. Our method is more generally applicable in that it does not assume an existing heuristic. In addition, our method is able to extract information from states that is not captured in a heuristic, and our method succeeds on failure cases mentioned in Thayer et al.'s work.

2.1.2 Dead-end detection

Another approach to learning from failed planning attempts is dead-end detection [23, 32], which is closely related to nogoods in constraint satisfaction problems. A dead-end is a search state from which no plan to the goal exists. Crucially, it is sometimes possible to examine a failed search attempt and identify dead-ends; a search state with no successors is a dead-end, and a state whose only successors are dead-ends is also a dead-end. A learning approach could leverage these identified dead-ends to learn to predict dead-ends in a new planning problem, potentially speeding up the search. Previous work on learning dead-end detectors considers planning in factored, logical domains, and identifies formulaic representations of states that are verifiable dead-ends [46, 47, 48].

Unfortunately, there are many domains of interest where dead-ends are limited or absent altogether (e.g., domains where all actions are reversible). Our notion of eliminability provides leverage in domains that have no dead-ends to detect. Eliminability not only subsumes dead-end detection — all edges incident with a dead-end are clearly eliminable — but also includes problem-specific redundancy, where if there are multiple paths from initial state to goal, all but one path can be eliminated. Rather than limiting learning to logical, factored domains, we consider a planning setting where states and transition models are not necessarily logical or factored, and we take an empirical approach in the spirit of the learning for planning literature.

2.1.3 Connections to reinforcement learning

It is worth noting that learning from failed planning attempts is related to the challenge of exploration with sparse rewards in reinforcement learning (RL) [2, 4, 15, 26, 31, 35]. Especially relevant is exploration in multitask RL; see Colas et al. [11] for a recent survey. Much of the difficulty of

exploration in RL stems from the transition model being unknown. For example, several approaches attempt to learn a transition model and use prediction error to guide exploration [7, 39]. In our planning setting, we assume the transition model is known.

2.2 Preliminaries

We consider deterministic finite planning domains with states $\mathcal{S}$, actions $\mathcal{A}$, and transition function $\textsc{succ}(s, a) = s'$ with $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$. Transitions have nonnegative costs; for simplicity, we assume unit transition costs. A single planning problem consists of an initial state $s_0 \in \mathcal{S}$ and a goal $g \subseteq \mathcal{S}$. A solution to a planning problem is a plan, that is, a sequence of actions $(a_0, a_1, \ldots, a_{T-1})$ such that $s_{t+1} = \textsc{succ}(s_t, a_t)$ for $0 \leq t < T$, and $s_T \in g$. In this work, we are interested in satisficing planning, where solutions need not be optimal.

Deterministic planning can be framed as graph search. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ for a planning domain associates one node $n_s \in \mathcal{V}$ per state s and one directed edge between nodes n_s and $n_{s'}$ with label a, denoted $(n_s, a, n_{s'}) \in \mathcal{E}$, if $\textsc{succ}(s, a) = s'$. A plan for a particular problem corresponds to a path in the graph starting at n_{s_0} and ending at a node n_{s_T} such that $s_T \in g$.

We are interested in learning to plan more efficiently and effectively from previous experience. Formally, we assume access to a set of Train planning problems with varying initial states and goals. A set of held-out Test problems are used for evaluation after training. The state space $\mathcal{S}$, action space $\mathcal{A}$ and transition function $\textsc{succ}$ are fixed across all problems. To permit generalization between problems with disjoint states, we will assume access to a featurizer $\phi(s_0, g, s, a, s') \in \mathcal{F}$, which maps an initial state, goal, and transition to a feature space (e.g., images).

We are specifically interested in *learning from planning failures.* In the planning-as-graph-search setting, a failed planning attempt can be represented by the set of nodes $\mathcal{V}'$ and edges $\mathcal{E}'$ that were explored during search; these constitute a subgraph $\mathcal{G}'$ of the domain graph $\mathcal{G}$, and the attempt is a failure when no path in $\mathcal{G}'$ leads from initial state to goal. Each Train problem is associated with one such subgraph. The question motivating this research is: what, if anything, can be learned from these failed searches that will aid planning in the Test problems?

2.3 Approach

The main insight leading to our approach is the following: in examining the subgraph of a failed planning attempt, it is possible to identify sets of edges that are *eliminable*. These are edges that, in hindsight, could have been left unexplored without inhibiting planning. Our approach is to use these eliminable edge sets to learn a predictive model that will allow us to preemptively eliminate edges on the held-out TEST problems. We next formalize what it means for an edge set to be eliminable and then describe the details of our model.

2.3.1 Eliminable Edge Sets

We start with a formal definition of eliminable edge sets:

Definition 1 (Eliminable edge set)**.** *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ for a planning domain and a problem (s_0, g), a set of edges $\mathcal{E}^- \subset \mathcal{E}$ is* eliminable *if either **1)** the problem is unsolvable, or **2)** the problem is solvable in the subgraph $\mathcal{G}^- = (\mathcal{V}, \mathcal{E} \setminus \mathcal{E}^-)$, i.e. there exists a path from $n_{s_0} \in \mathcal{V}$ to a node $n_{s_T} \in \mathcal{V}$ in the subgraph $\mathcal{G}^-$, where $s_T \in g$.*

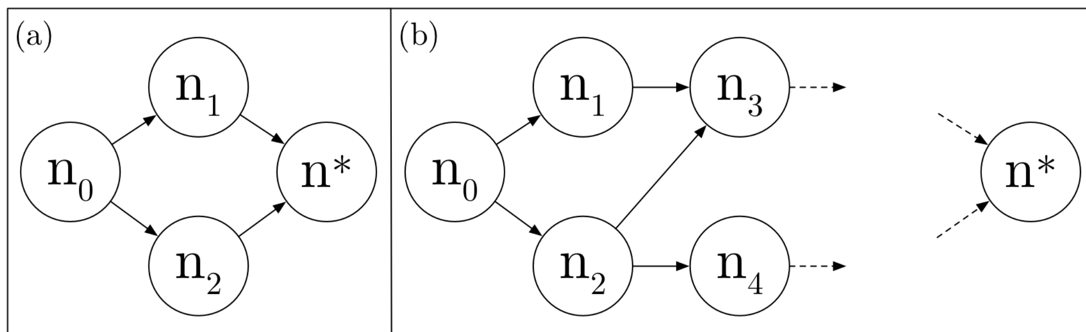


Figure 1: Two search graphs where node n_0 has the initial state and n^* has the goal. Edge labels omitted for clarity. (a) The edge set $\{(n_0, n_1), (n_1, n^*)\}$ is eliminable, since a plan through n_2 would remain in the graph with those edges removed. (b) Given a failed search attempt that has only expanded five nodes from the initial state, one can already see that the edge set $\{(n_0, n_1), (n_1, n_3)\}$ is eliminable. In this work, we use eliminable edge sets identified from failed searches to learn a predictive model that allows us to preemptively remove eliminable edge sets on new problems.

In the graph illustrated in Figure 1a, where action labels are omitted for visual clarity, the edge

set $\{(n_0, n_1), (n_1, n^*)\}$ is eliminable because the path $((n_0, n_2), (n_2, n^*))$ remains in the graph with the edge set removed. Similarly, $\{(n_0, n_2), (n_2, n^*)\}$ is eliminable, since the path through n_1 remains in the corresponding subgraph. The set $\{(n_0, n_1), (n_0, n_2)\}$, however, is not eliminable.

From this example, we can see that eliminability is importantly a property of a *set* of edges and cannot be reduced to a property of individual edges; one cannot determine whether the edge (n_0, n_1) is safe to eliminate without knowing whether the edge (n_0, n_2) will also be eliminated.

There is a clear relationship between eliminable edges and plans: given the path of a plan, the set of all edges *not* in the path is eliminable. However, as we will see in the next section, in the subgraph for a failed planning attempt, where no plan can be found, it may still be possible to identify nontrivial eliminable edge sets.

2.3.2 Eliminability in Failed Searches

Given a subgraph representing a failed planning attempt, we would like to identify an eliminable edge set. We begin with definitions familiar from graph search.

Definition 2 (Expanded node, open node)**.** *Given a subgraph $\mathcal{G}_{sub} = (\mathcal{V}_{sub}, \mathcal{E}_{sub})$ of a domain graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a node $n \in \mathcal{V}_{sub}$ is* expanded *if for all edges $(n, a, n') \in \mathcal{E}$, $(n, a, n') \in \mathcal{E}_{sub}$. Otherwise, the node is* open.

Definition 3 (Reachable node, edge)**.** *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a planning problem (s_0, g), a node $n \in \mathcal{V}$ is* reachable *if there is a path from n_{s_0} to n in $\mathcal{G}$. An edge $(n, a, n') \in \mathcal{E}$ is reachable if n is reachable.*

The following lemma gives us a mechanism to identify eliminable edge sets in certain naturally arising subgraphs.

Lemma 1. Given a subgraph $\mathcal{G}_{\text{sub}} = (\mathcal{V}_{\text{sub}}, \mathcal{E}_{\text{sub}})$ of the domain graph $\mathcal{G}$, a planning problem (s_0, g) with $n_{s_0} \in \mathcal{V}_{\text{sub}}$, and an edge set $\mathcal{E}^- \subseteq \mathcal{E}_{\text{sub}}$, let $\mathcal{G}^-_{\text{sub}} = (\mathcal{V}_{\text{sub}}, \mathcal{E}_{\text{sub}} \setminus \mathcal{E}^-)$. Suppose **1)** no plan exists in $\mathcal{G}_{\text{sub}}$; **2)** all open nodes in $\mathcal{V}_{\text{sub}}$ are reachable in $\mathcal{G}^-_{\text{sub}}$; and **3)** all edges in $\mathcal{E}^-$ are reachable in $\mathcal{G}_{\text{sub}}$. Then $\mathcal{E}^-$ is eliminable.

Proof. To review, there are four graphs here:

- $\mathcal{G}$ is the full domain graph.
- $\mathcal{G}_{\text{sub}}$ is the subgraph of the failed search, that is, the nodes and edges that were explored during a search where no plan was found.
- $\mathcal{G}^{-}_{\text{sub}}$ is the subgraph of the failed search with the edges in $\mathcal{E}^{-}$ eliminated.
- Let $\mathcal{G}^{-} = (\mathcal{V}, \mathcal{E} \setminus \mathcal{E}^{-})$ be the subgraph of the full domain graph with the edges in $\mathcal{E}^{-}$ eliminated.

If the planning problem is unsolvable, any edge set is eliminable, and the conclusion is trivial. Otherwise, there exists a plan in $\mathcal{G}$, that is, a path from $n_{s_0} \in \mathcal{V}$ to some $n_{s_T} \in \mathcal{V}$ with $s_T \in g$. This path must contain a node n_{open} that is open in $\mathcal{G}_{\text{sub}}$; otherwise, the path would contain only nodes that are expanded in $\mathcal{G}_{\text{sub}}$, and $\mathcal{G}_{\text{sub}}$ would contain a plan, violating assumption (1). By assumption (2), n_{open} is reachable in $\mathcal{G}^{-}_{\text{sub}}$, and therefore also reachable in $\mathcal{G}^{-}$, since $\mathcal{G}^{-}_{\text{sub}}$ is a subgraph of $\mathcal{G}^{-}$. It remains to show that there is a path from n_{open} to n_{s_T} in $\mathcal{G}^{-}$.

Consider a path from n_{open} to n_{s_T} in $\mathcal{G}$. Let n'_{open} be the last open node in this path, i.e., the closest open node to the goal n_{s_T}. Since it is open, n'_{open} is reachable in $\mathcal{G}^{-}_{\text{sub}}$ and therefore also in $\mathcal{G}^{-}$. Now consider the edges on the path from n'_{open} to n_{s_T} and suppose that one or more are in $\mathcal{E}^{-}$, the eliminated set. Because $\mathcal{E}^{-} \subseteq \mathcal{E}_{\text{sub}}$, these edges are in $\mathcal{E}_{\text{sub}}$. Furthermore, by assumption (3), each such edge is reachable in $\mathcal{G}_{\text{sub}}$, and thus all of the constituent nodes are reachable in $\mathcal{G}_{\text{sub}}$ as well, including n_{s_T}. But we assumed in (1) that $\mathcal{G}_{\text{sub}}$ does not contain a plan, so this is a contradiction. Therefore no edges along the path from n'_{open} to n_{s_T} are eliminated, and thus a plan in $\mathcal{G}^{-}$ (from n_{s_0} to n'_{open} to n_{s_T}) is maintained after the elimination of $\mathcal{E}^{-}$. □

Thus to test whether an edge set is eliminable in a subgraph for a failed planning attempt (where no plan exists), one could check whether each edge is reachable in the subgraph, and that there are paths from the initial state to all open nodes in the graph with the edges removed. For example, consider the graph shown in Figure 1b, where the five non-goal nodes are expanded, but descendants of n_3 and n_4 are open. In examining the five-node subgraph, we can see that the edge

set $\{(n_0, n_1), (n_1, n_3)\}$ must be eliminable, since both edges are reachable, and a path from n_0 to n_3 and a path from n_0 to n_4 remain in the graph after those edges have been removed.

Identifying Eliminable Edges in Forward Searches. In practice, when planning with forward[1] search algorithms like GBFS or A*, we do not need to explicitly test whether edge sets satisfy the criteria of Lemma 1. At any given time in the execution of these algorithms, the shortest paths from the initial state to all open nodes are maintained. The edges in these paths are *not* eliminable. The complement of this set — all previously visited edges that are not in the paths — constitute an eliminable set. We can therefore extract an eliminable edge set directly from a failed forward search.

2.3.3 Learning to Predict Eliminability

After applying the above technique to each problem (s_0, g) in the TRAIN set, we obtain one set of eliminable edges per problem, denoted $\mathcal{D}^-_{s_0,g}$. For each problem, let $\mathcal{D}^+_{s_0,g}$ be the complement of $\mathcal{D}^-_{s_0,g}$: edges that were explored in the failed planning attempt for (s_0, g), but are not in the eliminable set. From these problem-specific sets, we can construct:

$$\mathcal{D}^- = \{(s_0, g, s, a, s') : (n_s, a, n_{s'}) \in \mathcal{D}^-_{s_0,g}\}$$
$$\mathcal{D}^+ = \{(s_0, g, s, a, s') : (n_s, a, n'_s) \in \mathcal{D}^+_{s_0,g}\}.$$

Recall that we have access to a featurizer $\phi(s_0, g, s, a, s') \in \mathcal{F}$. Applying the featurizer to all points in $\mathcal{D}^-$ and $\mathcal{D}^+$, we arrive at a dataset that is amenable to standard binary classification. After training a classifier $f_\theta : \mathcal{F} \to \{0, 1\}$ with parameters θ, we can use the learned model to eliminate edges during search on a new problem. In practice, rather than pruning edges entirely, we will learn a probabilistic classifier and use the predicted probability that an edge is eliminable to determine the order of expansion during GBFS.

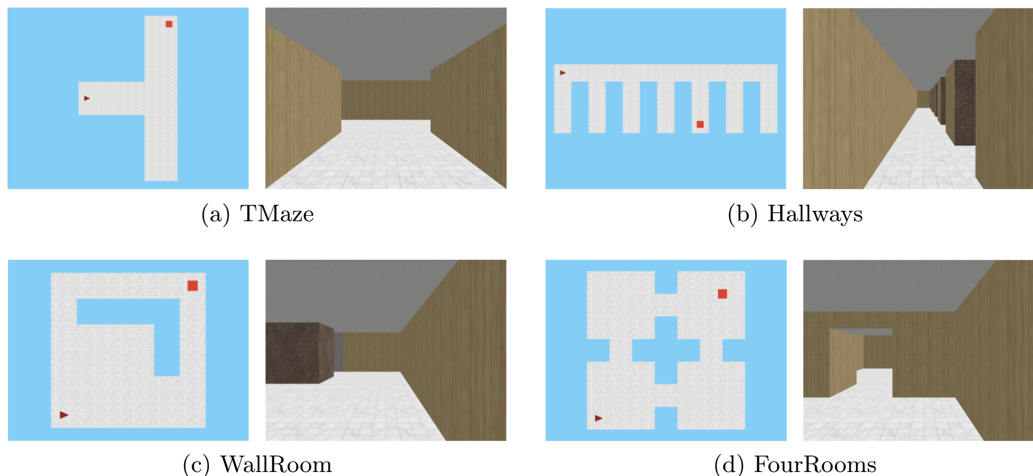
[1] This work focuses on forward search, but we expect that it possible to formulate a version of Lemma 1 that would work for backward search as well, where expansion and reachability would emanate from the goal, rather than the initial state.

It is important to emphasize that we are not learning to predict whether an edge is eliminable in any universal sense; as we saw in Section 2.3.1, individual edge eliminability is not well-defined. Instead, the learned model f_θ can be used to predict a *certain* eliminable edge set for any given problem. In other words, given a problem (s_0, g), f_θ acts like an *indicator function*: all edges (n_s, a, n'_s) for which $f_\theta(\phi(s_0, g, s, a, s')) = 1$ are in the eliminable edge set.

Erring on the Side of Non-Eliminability. In predicting eliminability, false positives (incorrectly predicting "eliminable") are more problematic than false negatives, since pruning or down-weighting a crucial edge could doom or delay search. In early experiments, we found that learned predictors would sometimes predict false positives as the search considered edges that were substantially different from those seen in the training data. To remedy this, inspired by previous work in exploration for RL [49], we introduce an *unseen wrapper* around our classifier that determines whether an edge is sufficiently different from previously seen edges by some metric, and if so, assigns it a zero probability of eliminability (see Section 2.4.1 for details).

2.4 Experiments and Results

We now present preliminary experiments and results.



(a) TMaze (b) Hallways

(c) WallRoom (d) FourRooms

Figure 2: Miniworld Domains

2.4.1 Experimental Setup

Domains. We test our approach in four visual navigation domains implemented in Miniworld [9]: `TMaze`, `Hallways`, `WallRoom`, and `FourRooms`. `Hallways` and `WallRoom` are custom domains original to this work. All domains involve an agent navigating to a goal. Figure 2 (left images) shows top views of the environments, where the agent's initial position is portrayed by the red arrow and a goal position portrayed by the red square. States are comprised of the agent position and direction. There are three possible actions from each state: move forward, turn 90° left, or turn 90° right. Each state is associated with a first-person image (Figure 2, right images). The featurizer $\phi(s_0, g, s, a, s')$ is a concatenation of the images for states s and s'.

In domains such as `TMaze` and `Hallways`, we expect our model to learn that edges corresponding to moving the agent forward into an empty hallway with no exits are likely to be eliminable. It is less clear *a priori* that the model will learn useful information in `WallRoom` or `FourRooms`.

Model Class. We parameterize the eliminability classifier f_θ as a convolutional neural network (CNN). For the unseen wrapper (Section 2.3.3), we discretize the state space using locality-sensitive hashing (LSH).

CNNs are implemented in PyTorch version 1.5.0. The image features are passed through two convolutional layers with kernel size 10 and stride 1 and two max pooling layers with kernel size 2 and stride 2, followed by three fully connected layers with ReLU activation with hidden dimensions 120 and 84.

LSH maps high dimensional input to discrete hash codes, such that similar inputs are mapped to the same hashes. In particular, we use SimHash [49], an LSH which measures similarity by angular distance. Given a vector $x \in \mathbb{R}^D$, SimHash retrieves a binary code $h = \text{sgn}(Ax) \in \{-1, 1\}^k$, where A is a $k \times D$ matrix with i.i.d. entries drawn from from a standard Gaussian distribution $\mathcal{N}(0, 1)$. Larger k values correspond to fewer collisions and finer granularity in discretization (we use $k = 500$). For our purposes, x is a flattened representation of $\phi(s_0, g, s, a, s')$, the image features for an edge.

Data Collection. Each of the four domains has one training task. To collect failed searches, we run blind best-first search (i.e., breadth-first search) with a predetermined number of node expansions such that no plan to the goal is found. The number of node expansions is determined as a proportion of the number of nodes an average best-first search takes to solve the problem. For example, `train_expansion=0.2` indicates that, for a task where blind search takes roughly 300 node expansions to solve, the training search expanded 60 nodes. We verified that no plans were found in any training problem.

Training. The CNN is trained with binary cross-entropy loss, using the Adam optimizer [28] with learning rate 0.0001 for 320 epochs.

Testing. Each domain has nine test tasks, with variation in the initial states and goals such that none are identical to the training task. During test search, the model predicts eliminability of edges as they are encountered. The probability of eliminability is used as a heuristic to guide GBFS.

Additional Details. Each experiment configuration shown is run with 25 random seeds, where randomness is introduced via neural network initialization, the SimHash A matrix, and the randomized tie-breaking during search. All search code is based on Pyperplan [1] and all neural-network code is written in PyTorch [38].

2.4.2 Results

As shown in Figure 3, in each of the four domains and across various training expansion amounts, the eliminability predictor improves search in test problems compared to a blind best-first search. Improvements generally increase as the training expansion amount increases. In simpler domains such as `TMaze`, we can see a case of diminishing returns; `train_expansion=0.4` achieves almost the same as `train_expansion=0.6`, which has indiscernable performance from `train_expansion=0.8`.

In figures 4 and 5 we probe the impact of the unseen wrapper. These experiments compare four models: `blind`, `blind_unseen`, `cnn`, and `cnn_unseen`. `blind` is a simple best-first search, whereas `blind_unseen` is the best-first search with the unseen wrapper, i.e. prioritizing unseen edges. `cnn`

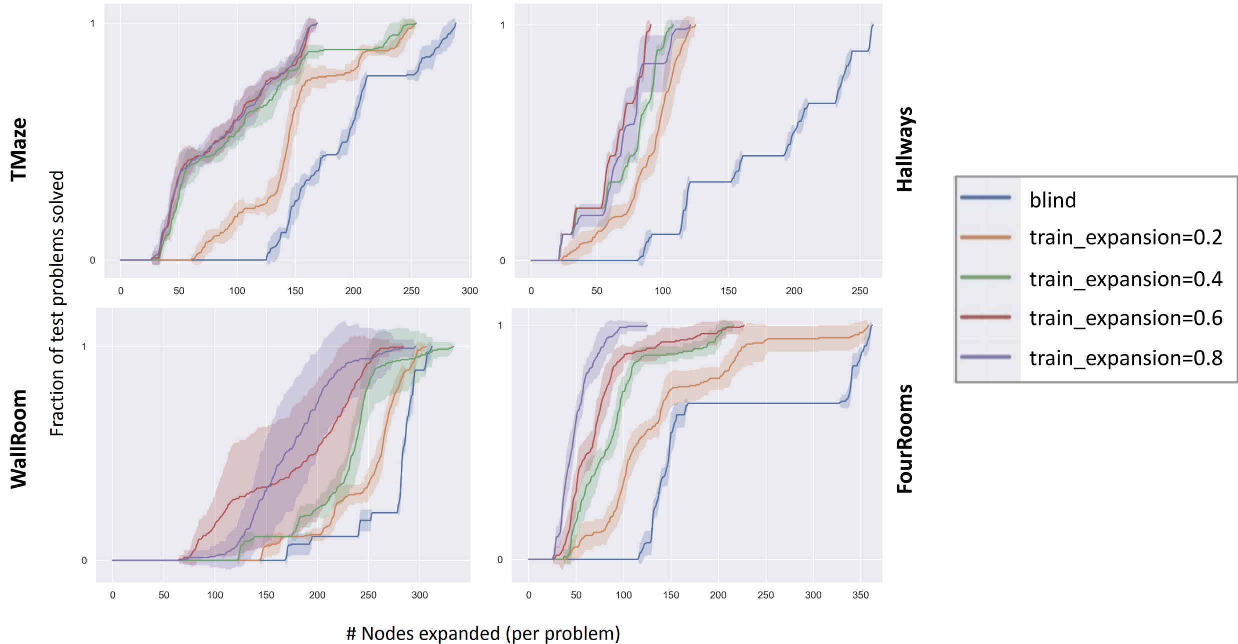


Figure 3: Model performance for different amounts of training expansion, where training expansion refers to the number of nodes expanded during training search as a fraction of the number of node expansions required for a blind best-first search search to solve the problem. In all domains and across different training expansion amounts, our approach improves upon a blind best-first search.

is the CNN model with no additions, and `cnn_unseen`, the CNN with the unseen wrapper, is the model we used for the main results.

Figure 4 shows model comparisons for `training_expansion=0.2`, and Figure 5 shows model comparisons for `training_expansion=0.8`. We see that, especially when search is limited during training, the CNN model without the unseen wrapper can occasionally perform quite poorly, sometimes significantly worse than a blind search. Although the unseen wrapper can be a detriment to average performance, it acts as a "safety net" and consistently prevents the CNN model from performing much worse than blind search. Note that at both high and low `training_expansion`, the CNN model with the unseen wrapper can provide substantial improvements to search in comparison to the `blind` and `blind_unseen` models.

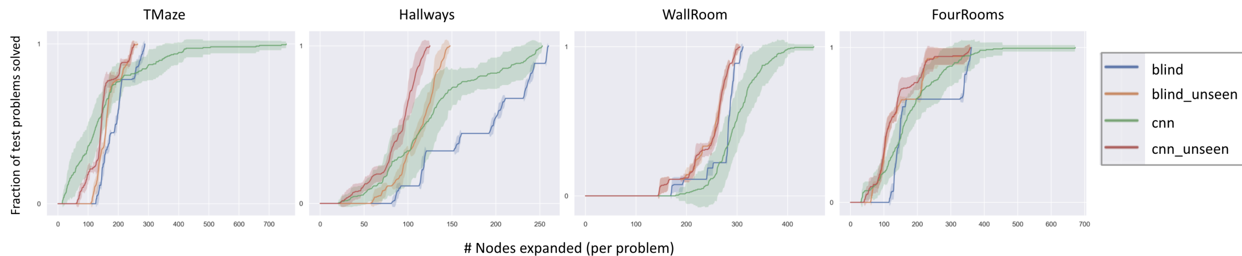


Figure 4: Models comparison at `training_expansion=0.2`

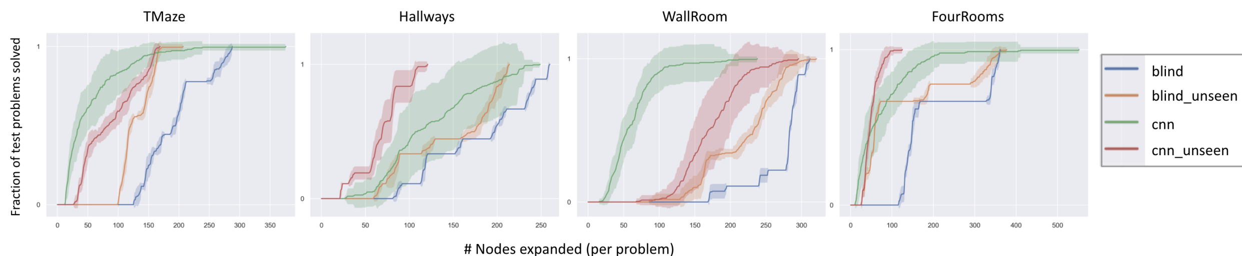


Figure 5: Models comparison at `training_expansion=0.8`

2.5 Discussion

In this direction of study, we investigated what can be learned from failed planning attempts alone, finding generalized eliminable edge set predictors to be a promising candidate. Among many possible future directions, we are most eager to (1) apply the approach in other domains, e.g., IPC tasks [33]; (2) study the approach in settings where nontrivial domain-independent (e.g., delete-relaxation) heuristics are available; (3) investigate learning *during* a single search, using what is learned to bootstrap planning in the rest of the problem; (4) consider further connections to the RL literature on exploration, perhaps adapting our approach to the RL setting where the transition model is unknown.

3 Goal-Conditioned Action Sampling in Integrated Task and Motion Planning

Our second direction of study takes us to a more challenging integrated task and motion planning setting. Here, the states and actions are hybrid discrete/continuous, and the planning is bi-level. For example, when we wish to Place a block, our discrete action Place takes in additional continuous parameters, perhaps the position and orientation of the placement. We have the higher level task plan – to move a block, Pick then Place – but also the lower level motion plan that determines the smooth trajectory towards the block to pick and the trajectory towards the destination to place.

The main question we investigate here is how to increase the effectiveness of planning when the task planning is not "downward-refinable", i.e. a task plan might not correspond to a valid motion plan. This common situation occurs when the abstractions made by task planning overlook potential infeasibilities that may arise during motion planning. A task plan to move a block into a box might not work if the task-level abstraction does not capture the fact that the block does not fit into the box.

We extend Neuro-Symbolic Relational Transition Models (NSRTs) [10], a class of models that attempts to address the challenges in task and motion planning by combining symbolic planning and neural models. Specifically, we add goal-conditioning to the learned action sampler in NSRTs and show empirical improvements on two simulated robotic tasks.

3.1 Related work

3.1.1 Task and Motion Planning

Task and motion planning has been extensively studied by the robotics community. Early work approached the problem assuming downward-refinability, finding a task plan first before finding suitable continuous parameter values [36]. The downward-refinability assumption could be relaxed if there is a mechanism to try alternative task plans when motion planning fails [44]. Alternatively,

in environments in which it is more costly to repeatedly create task-plans that are not downward-refinable than to first find satisfying continuous parameters for task-level actions, it makes sense to satisfy continuous constraints before task-level planning [43] [16] [22]. Yet another approach is to interleave the search for task-level actions and satisfying continuous parameters [13] [14]. See [17] for a survey including a more thorough discussion of these methods.

As discussed in [17], machine learning for task and motion planning may be helpful in three ways: (1) learning models, (2) learning search guidance, and (3) learning sampling guidance. Our previous direction of study broadly falls under the second category, and our current direction of study falls under the third category.

3.1.2 Learning Action Samplers

There have been a number of previous studies on learning the action samplers that sample continuous parameters for task-level actions. For example, [52] improves action sampling using risk-aware sampling and diversity-aware sampling. The former uses a Bayesian estimate of the scoring function to select action instances for planning; the latter encourages diversity by adapting a kernel used in the sampling process based on previous planning experience. [27] learns an action sampler from both on-target trajectories (towards the goal) and off-target trajectories, using GANs to estimate an importance ratio to learn a target distribution that assigns low probabilities to inefficient actions. As far as we are aware, our work is the first to study the effects of goal-conditioning a learned action sampler.

3.1.3 Neuro-Symbolic Relational Transition Models

As previously mentioned, we work with NSRTs [10]. NSRTs use symbolic AI planning in an outer loop to guide continuous planning with neural models in an inner loop. Unlike previous work in the task and motion planning setting, NSRTs do not assume knowledge of an abstract or continuous transition model. The models learn four components: (1) abstract actions that act on abstract states, (2) an abstract transition model that defines the effects of abstract actions on abstract

states, (3) a low-level transition model over continuous states and actions, and (4) action samplers that map abstract actions to continuous actions. In our work, we focus on learning for the action sampler, and so we differ slightly from NSRTs by using a known continuous transition model. In the next section, we will introduce formalisms used by NSRTs to better describe our work in the models' context.

3.2 Problem Setting

In the task and motion planning literature, *predicates* are a discrete set of named relations over objects. Predicates such as HOLDING(`block1`) discretize the continuous state space by abstracting away the continuous pose and physical properties of `block1`. A predicate applied to a specific object is called a *ground atom*, whereas a predicate applied to typed placeholder variables is called a *lifted atom*. For example, HOLDING(`block1`) is a ground atom, and HOLDING($?b$) is a lifted atom where $?b$ is an unspecified block-type object. NSRTs assume that the set of predicates are given.

Following the notations in [10], we have that a Neuro-Symbolic Relational Transition Model (NSRT) is a tuple $\langle O, P, E, h, \pi \rangle$ where:

- $O = (o_1, \cdots, o_k)$ is an ordered list of parameters, each of a specified object type.
- P is a set of symbolic preconditions, each of which is a lifted atom over the parameters O.
- E is a set of symbolic effects, each of which is a lifted atom over the parameters O.
- h is low level transition model mapping continuous states and actions to the next continuous states. In the original work, this is learned; in our study we use a ground truth model.
- $\pi(a|v)$ is an action sampler, a neural network that defines a conditional distribution over actions $a \in \mathcal{A}$, where A is the continuous action space and v is a vector of attribute values.

In the original work, the authors learn and plan with a collection of NSRTs. The collective preconditions P and effects E allow for task-level planning in the abstract state and action space, and the collective h and π allow for refining the task-level plan into environment actions in continuous space.

We can now discuss the details of the action sampler. First, we let σ denote an injective substitution mapping each parameter o_i of an NSRT to an object. This mapping allows us to ground an NSRT with objects; a ground NSRT can be thought of as an abstract action. Then, given a continuous state s and a ground NSRT with substitution σ, we can define the context of s as $v_\sigma = s[\sigma(o_1)] \circ \cdots \circ s[\sigma(o_k)]$, where $\circ$ denotes vector concatenation. In other words, v_σ is the concatenation of attribute values of the objects acted on by the NSRT. After data collection, trajectories are partitioned into transitions $(v_\sigma, a, v_{\sigma'})$ for learning; we refer the reader to [10] for more details on the partitioning process. For the action sampler π, we use the transitions $(v_\sigma, a, \cdot)$ to learn a distribution $P(a|v_\sigma)$. Specifically, a fully connected network predicts the mean and variance of a Gaussian distribution that maximizes the likelihood of the action a given v_σ.

The goal of a task and motion planning problem is a set of ground atoms, e.g. $\{\textsc{Above}(\texttt{block1}, \texttt{block2}), \textsc{Above}(\texttt{block2}, \texttt{block3})\}$. In our current experiments, the goal is a single ground atom. We can therefore create a goal attribute vector g by concatenating the attribute values of the objects in the goal atom. To goal-condition the action sampler, we learn the distribution $P(a|v_\sigma, g)$ by augmenting each transition with g to become $(v_\sigma, a, \cdot, g)$.

3.3 Experiments

We test goal-conditioned action sampling on two simulated 2D robotic environments, `NarrowHallway` and `Cover`. Refer to Fig. 6 for a visual rendering of the environments.

The `NarrowHallway` environment involves a robot arm and two cans in a narrow corridor. The robot arm has a fixed base position and can extend linearly any distance from the base (as long as it remains within the hallway). The goal of each task in this environment is for the robot arm to pick up a pre-specified desired can. However, in the initial state, the robot arm is unable to pick up the desired can because the other can obstructs its path. Therefore, to complete the task, the robot arm should first pick up the obstructing can, then place the obstructing can, and finally pick up the goal. All of these pick and place actions must be collision-free, where collisions are defined as picking from or placing at a position such that there is an angle of less than $45°$ between the robot

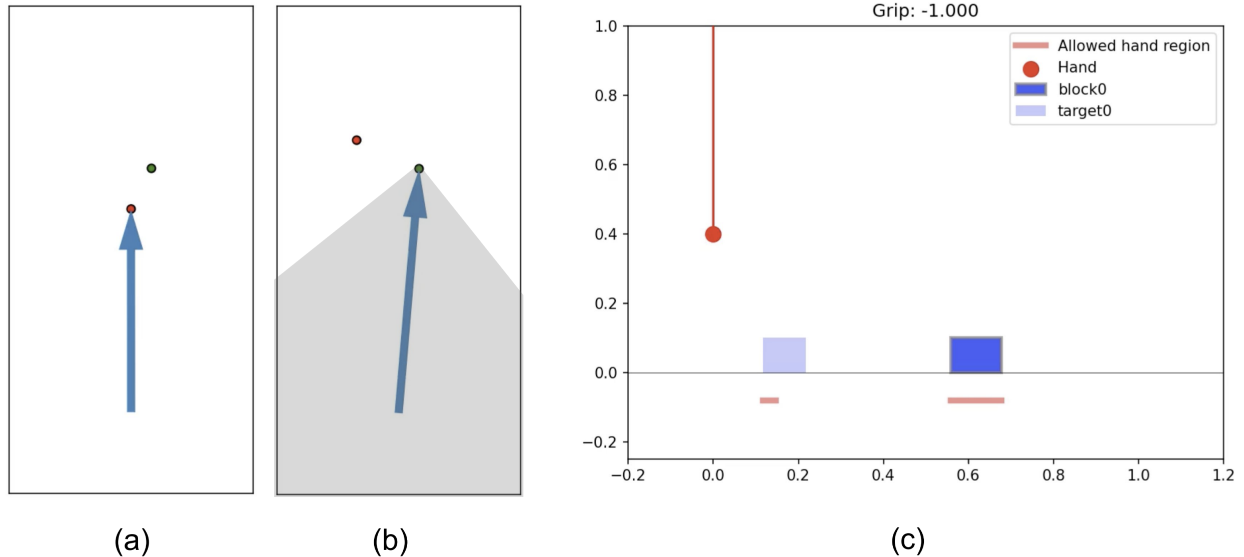


Figure 6: (a) In the `NarrowHallway` environment, a robot arm with a fixed base near the entrance of a narrow hallway must pick up the desired goal can represented by the green circle, but is initially obstructed by another can. (b) The grey area shows the area of collision when picking the goal can, and other cans in this area will lead to a failure in picking. (c) The `Cover` environment involves a vertical robot arm picking up a block and placing it so that the entire block covers the target. The robot arm must be above the allowed hand regions when picking and placing.

base, the pick/place position, and another can (see Fig 6b). The predicates for the `NarrowHallway` environment are HandEmpty(?robot), Holding(?can, ?robot). The tasks in the environment only vary based on the position of the desired can, which can appear either to the right-and-top or left-and-top of the obstructing can.

The `Cover` environment involves a vertical robot arm, a block, a target region, and allowed hand regions. The block and target region are located on a surface, and therefore their position varies along one dimension. The goal of tasks in this environment is for the robot arm to pick up the block and place it such that the block covers the entire width of the target region. While picking and placing, the robot arm must be above the allowed hand regions. Therefore, to fully cover the target region while placing the block in the allowed hand region, it may be necessary for the robot arm to pick up the block at a position offset from the center. The predicates for the `Cover` environment are Cover(?block, ?target), HandEmpty(?robot), Holding(?block, ?robot). The tasks in

the environment differ based on the block position, the target position, and the allowed hand region under the target. The target is always offset from the center of the target, either on the left or the right.

For the experiments, we vary the number of action samples per step, which is the number of attempts the action sampler has to refine the high-level plan of abstract actions. The maximum attempted number of high-level plans is kept the same within environments (2 for `NarrowHallway` and 1 for `Cover`, which are the minimum numbers needed for finding a successful plan). Each experiment is run over 10 seeds, which controls randomization over state initialization and model initialization. We train on 50 tasks and test on 50 tasks; the tasks are currently of the same complexity but we hope to increase the complexity of the test tasks in the future.

3.4 Results

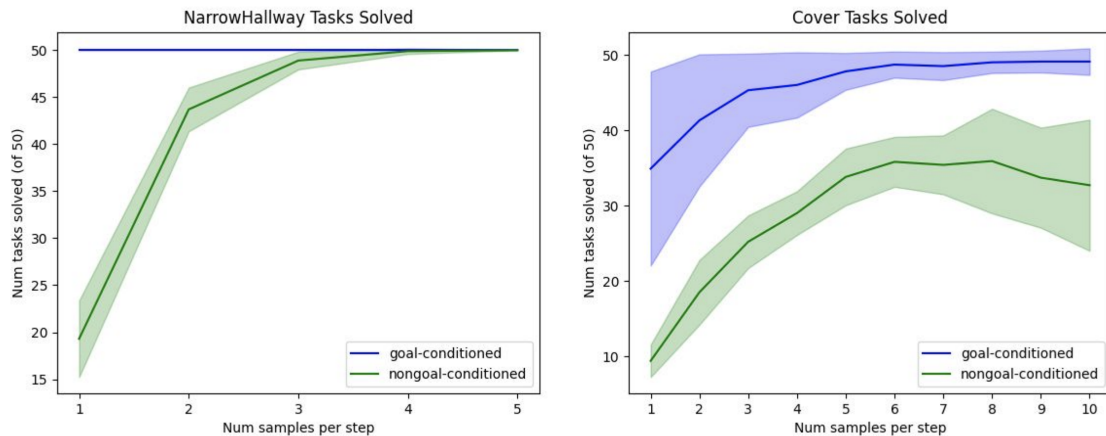


Figure 7: Number of tasks solved (out of 50) for goal-conditioned vs non-goal-conditioned solving action sampling strategies.

Our first set of experiments compares the success rate of goal-conditioned action sampling versus non-goal-conditioned action sampling. As shown in Fig. 7, goal-conditioned sampling increases the proportion of solved tasks in both `NarrowHallway` and in `Cover`. In `NarrowHallway`, goal-conditioning sampling can solve the tasks perfectly starting from 1 sample per step (solving 50 of

50 tasks with 0 standard deviation). The non-goal-conditioned sampler increases in success as the number of samples per step is increased, but only achieves the same performance at 5 samples per step. In `Cover`, the goal-conditioned sampler solves an average of 35 out of 50 tasks with high variance with 1 sample per step, but solves all 50 tasks consistently with under 10 samples per step. The non-goal-conditioned sampler solves an average of 10 out of 50 tasks with 1 sample per step, and its success generally increases with the number of samples per step. There is an unexpected increase of variance and decrease in success rate above 8 samples per step, but we believe this may be due to our smaller sample size of 10 seeds. We may further investigate this observation in the future with a larger sample size.

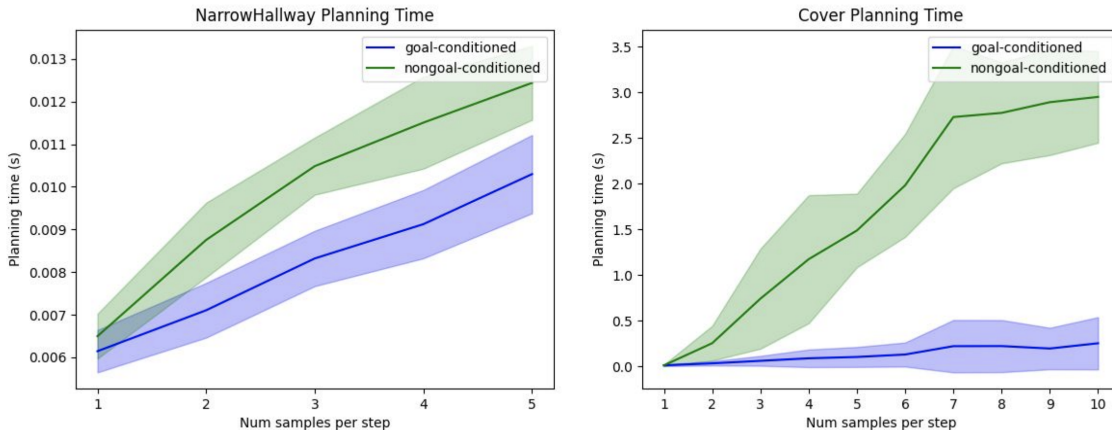


Figure 8: Planning time for goal-conditioned vs non-goal-conditioned solving action sampling strategies.

Our second set of experiments compares the planning time of goal-conditioned action sampling versus non-goal-conditioned action sampling (Fig. 8). Here, we see that in both environments, goal-conditioned action sampling results in planning that takes less time than non-goal-conditioned action sampling. This is expected because goal-conditioned action sampling is likely to succeed in fewer refinement attempts. In addition, the planning time increases as the number of allowed samples per step increases. This is also expected, as the number of tasks that are refined to completion increases along the x-axis.

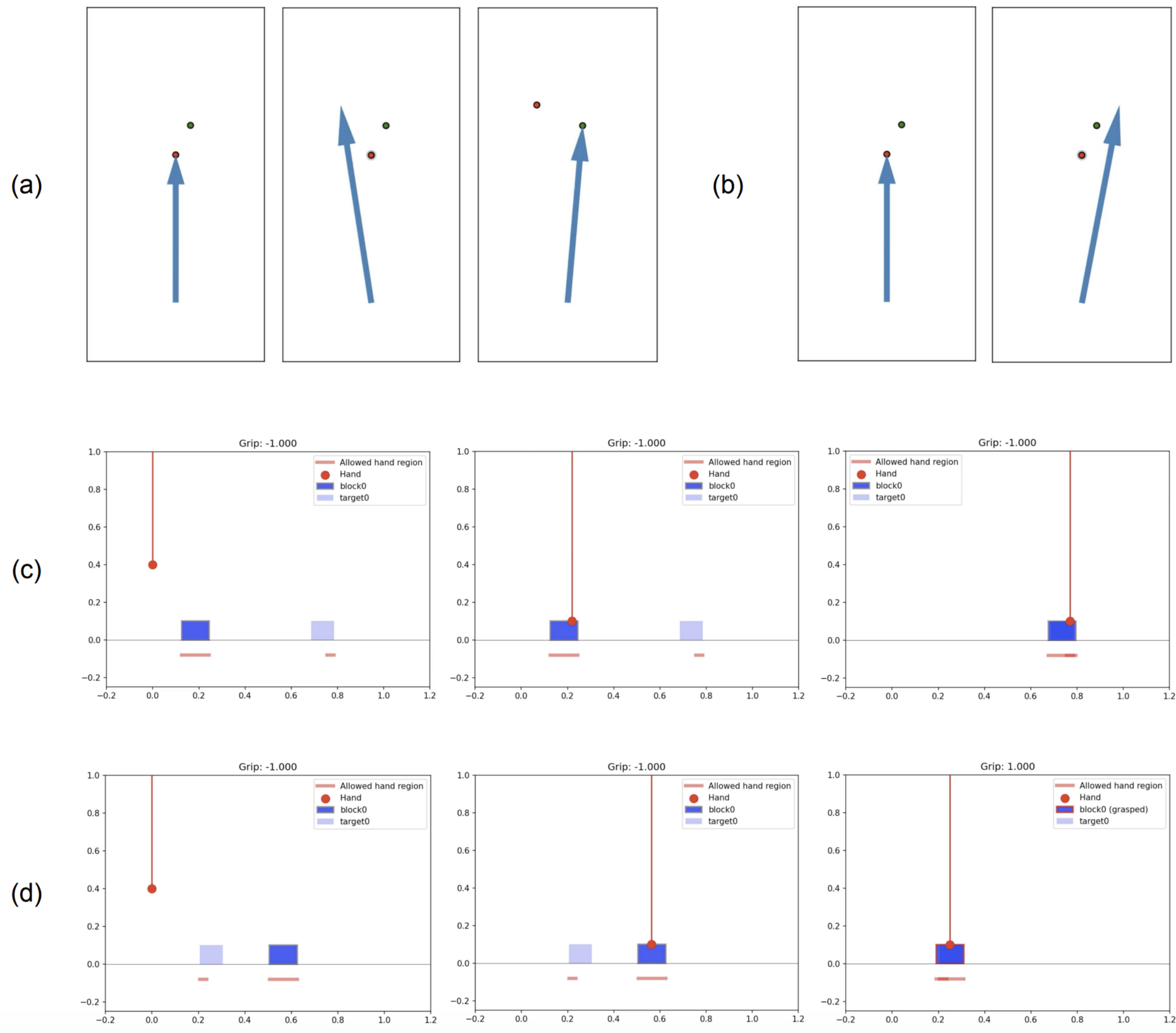


Figure 9: Examples of goal-conditioned and non-goal-conditioned sequences, where key frames are selected to be shown. (a) Goal-conditioned action sampling in `NarrowHallway` solves the task successfully by choosing to place the obstructing can at a position that will no longer obstruct the goal. (b) A common failure case in non-goal-conditioned action sampling is when the robot attempts to place the obstructing can behind the goal, but the action fails because the desired can obstructs the placement. (c) In `Cover`, goal-conditioned action sampling picks up the block at an offset so that the block will successfully cover the entire target. (d) When not goal-conditioned, the sampler may choose to pick up the block in the middle of the block, resulting in the robot arm outside the allowed hand regions when placing the block to cover the target region.

In Fig. 9, we qualitatively observe the differences between goal-conditioned and non-goal-conditioned action sampling. In the NarrowHallway environment (Fig. 9(a)(b)), we see that goal-conditioned action sampling allows the robot to place the undesired can in a position that is neither obstructed by the goal nor would potentially obstruct the goal in the future. Non-goal-conditioned action sampling cannot make guarantees about either, and typically fails during placement of the undesired can. In the Cover environment (Fig. 9(c)(d)), goal-conditioned sampling allows the robot to pick the block in a position such that the placement successfully covers the target region. Non-goal-conditioned sampling fails because sampling the pick action is not informed by the offset between the target and target's allowed hand region; the placement that covers the target region requires the robot arm to be outside the allowed hand region.

3.5 Discussion

Therefore, in this direction of study, we found that goal-conditioning the action sampler in the NSRTs framework increases the success rate and decreases the time of planning. Our current study is limited to two simple tasks in which it is relatively clear that a relationship exists between the continuous parameters chosen to refine abstract actions and the attribute values corresponding to the objects in the goal atom. In the future, we would like to observe the effects of goal-conditioning the action sampler in more complex tasks.

To do so, we must also increase the generalization capabilities of our goal-conditioned action sampler. Our current approach assumes that there is only one goal atom, in which case a concatenation of the attribute values of the objects in the goal atom is sufficient to represent the goal. We will handle multiple goal atoms using graph neural networks [40], which will allow us to better represent multiple objects and their symbolic relationships as specified by the goal atoms.

4 Conclusions

In this thesis, we first explored learning from failures in classical planning. We defined *eliminable edge sets*, and we showed how to find eliminable edge sets in search graphs. In experiments on navigation tasks, we found that learning a model of edge eliminability can increase the efficiency of planning on a new similar task; this improvement can be seen even if the failed search made little progress towards the goal. We then investigated goal-conditioning action samplers in the NSRTs framework, and we found that goal-conditioning the sampler increases success rate and decreases planning time on two simulated robotic tasks.

Our two directions of study were both motivated by a desire for efficient planning. In our first study, we wanted our planner to avoid repeatedly making similar mistakes, such as walking down a dead-end hallway. Afterwards, we were curious about how we can learn to reason about such failures in more complex, hierarchical planning settings. A real world motivating example analogous to `Cover` is a robot attempting to place a tall canister upright into a bin – the robot that grasps the canister from the side when picking up the canister will encounter a failure when attempting to place it. How could we have the robot autonomously understand that the reason for failure was the grasp orientation (top vs side), rather than the position of the side grasp? There is a significant body of work on failure diagnosis and plan repair [29][5][51], but we realized a cleaner solution in this case is to goal-condition the grasp, thus grasping correctly on the first try and avoiding failure.

More broadly, our methodology in learning eliminable edge sets works by extracting useful information from the state features that is not captured by the state abstraction. In our navigation task, the visual information from the agent's perspective was able to guide the search better than only knowing the positional coordinates of the agent. This suggests that eliminable edge sets or similar reasoning about the search graph may be helpful in future work on learning state abstractions. Additionally, although our goal-conditioned action sampling is currently limited to the NSRTs framework, we hope our study motivates future work in investigating goal-conditioning in other learned samplers.

References

[1] Alkhazraji, Y.; Frorath, M.; Grützner, M.; Helmert, M.; Liebetraut, T.; Mattmüller, R.; Ortlieb, M.; Seipp, J.; Springenberg, T.; Stahl, P.; and Wülfing, J. 2020. Pyperplan. `https://doi.org/10.5281/zenodo.3700819`. doi:10.5281/zenodo.3700819. URL `https://doi.org/10.5281/zenodo.3700819`.

[2] Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; and Zaremba, W. 2017. Hindsight experience replay. *arXiv preprint arXiv:1707.01495* .

[3] Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence* 175(16-17): 2075–2098.

[4] Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, 1471–1479.

[5] Bidot, J.; Schattenberg, B.; and Biundo, S. 2008. Plan Repair in Hybrid Planning. *Advances inArtificial Intelligence* .

[6] Bonet, B.; and Geffner, H. 2015. Policies that generalize: Solving many planning problems with the same policy. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[7] Burda, Y.; Edwards, H.; Storkey, A.; and Klimov, O. 2018. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894* .

[8] Campbell, M.; Hoane, A. J.; and Hsu, F.-h. 2002. Deep Blue. *Artificial Intelligence* 134: 57–83.

[9] Chevalier-Boisvert, M. 2018. gym-miniworld environment for OpenAI Gym. `https://github.com/maximecb/gym-miniworld`.

[10] Chitnis, R.; Silver, T.; Tenenbaum, J. B.; Lozano-Perez, T.; and Kaelbling, L. P. 2021. Learning STRIPS operators from noisy and incomplete observations. *arXiv preprint arXiv:2105.14074* .

[11] Colas, C.; Karch, T.; Sigaud, O.; and Oudeyer, P.-Y. 2020. Intrinsically Motivated Goal-Conditioned Reinforcement Learning: a Short Survey.

[12] DeepMind. 2020. AlphaGo: The Story So Far. `https://deepmind.com/research/case-studies/alphago-the-story-so-far`.

[13] Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic Attachments for Domain-Independent Planning Systems. *International Conference on Automated Planning and Scheduling* .

[14] Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009. Integrating Symbolic and Geometric Planning for Mobile Manipulation. *IEEE International Workshop on Safety, Security and Rescue Robotics* .

[15] Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; and Clune, J. 2019. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995* .

[16] Garrett, C.; Lozano-Perez, T.; and Kaelbling, L. 2017. FFRob: Leveraging symbolic planning for efficient task and motion planning. *International Journal of Robotics Research* .

[17] Garrett, C. R.; Chitnis, R.; Holladay, R.; Kim, B.; Silver, T.; Kaelbling, L. P.; and Lozano-Pérez, T. 2020. Integrated Task and Motion Planning. *Annual Review of Control, Robotics, and Autonomous Systems* .

[18] Garrett, C. R.; Kaelbling, L. P.; and Lozano-Pérez, T. 2016. Learning to rank for synthesizing planning heuristics. *arXiv preprint arXiv:1608.01302* .

[19] Gomoluch, P.; Alrajeh, D.; and Russo, A. 2019. Learning classical planning strategies with policy gradient. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 637–645.

[20] Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2018. Learning generalized reactive policies using deep neural networks. In *Proceedings of the International Conference on Automated Planning and Scheduling.*

[21] Gutiérrez, M. A.; Manso, L. J.; Núñez, P.; and Bustos, P. 2018. Planning object informed search for robots in household environments. *IEEE International Conference on Autonomous Robot Systems and Competitions* .

[22] Hauser, K.; Ng-Thow-Hing, V.; and Gonzalez-Baños, H. 2011. Randomized multi-modal motion planning for a humanoid robot manipulation task. *International Journal of Robotics Research* .

[23] Helmert, M. 2004. A Planning Heuristic Based on Causal Graph Analysis. In *ICAPS*, volume 16, 161–170.

[24] Jiménez, S.; De La Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* 27(4): 433–467.

[25] Jiménez, S.; Segovia-Aguas, J.; and Jonsson, A. 2019. A review of generalized planning. *The Knowledge Engineering Review* 34: e5.

[26] Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4: 237–285.

[27] Kim, B.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. Guiding Search in Continuous State-action Spaces by Learning an Action Sampler from Off-target Search Experience. *Association for the Advancement of Artificial Intelligence (AAAI)* .

[28] Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

[29] Kleer, J.; and C, W. B. 1987. Diagnosing multiple faults. *Artificial Intelligence* .

[30] Knoblock, C. A. 1992. An Analysis of ABSTRIPS. *Artificial Intelligence Planning Systems* .

[31] Lehman, J.; and Stanley, K. O. 2008. Exploiting open-endedness to solve problems through the search for novelty. In *Eleventh International Conference on Artificial Life (ALIFE XI).*

[32] Lipovetzky, N.; Muise, C.; and Geffner, H. 2016. Traps, invariants, and dead-ends. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 26.

[33] Long, D.; and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research* 20: 1–59.

[34] Martin, M.; and Geffner, H. 2004. Learning generalized policies from planning examples using concept languages. *Applied Intelligence* 20(1): 9–19.

[35] Nair, A.; McGrew, B.; Andrychowicz, M.; Zaremba, W.; and Abbeel, P. 2018. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 6292–6299. IEEE.

[36] Nilsson, N. J. 1984. Shakey the Robot.

[37] Paden, B.; Čáp, M.; Yong, S. Z.; Yershov, D.; and Frazzoli, E. 2016. A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles. *IEEE Transactions on Intelligent Vehicles* 1: 33–55.

[38] Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[39] Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 16–17.

[40] Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and G, M. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20: 61–80.

[41] Shen, W.; Trevizan, F.; and Thiébaux, S. 2020. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling.*

[42] Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529(7587): 484–489.

[43] Simeon, T.; JP, L.; J, C.; and A, S. 2004. Manipulation planning with probabilistic roadmaps. *International Journal of Robotics Research* .

[44] Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined Task and Motion Planning Through an Extensible Planner-Independent Interface Layer. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE.

[45] Srivastava, S.; Immerman, N.; Zilberstein, S.; and Zhang, T. 2011. Directed search for generalized plans using classical planners. *Proceedings of the International Conference on Automated Planning and Scheduling* .

[46] Steinmetz, M.; and Hoffmann, J. 2016. Towards clause-learning state space search: Learning to recognize dead-ends. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

[47] Steinmetz, M.; and Hoffmann, J. 2017. Search and Learn: On Dead-End Detectors, the Traps they Set, and Trap Learning. In *IJCAI*, 4398–4404.

[48] Steinmetz, M.; and Hoffmann, J. 2017. State space search nogood learning: Online refinement of critical-path dead-end detectors in planning. *Artificial Intelligence* 245: 1–37.

[49] Tang, H.; Houthooft, R.; Foote, D.; Stooke, A.; Chen, O. X.; Duan, Y.; Schulman, J.; DeTurck, F.; and Abbeel, P. 2017. # Exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*, 2753–2762.

[50] Thayer, J. T.; Dionne, A.; and Ruml, W. 2011. Learning Inadmissible Heuristics During Search. *International Conference on Automated Planning and Scheduling (ICAPS)* .

[51] van der Krogt, R.; and de Weerdt, M. 2005. Plan Repair as an Extension of Planning. *International Conference on Automated Planning and Scheduling (ICAPS)* .

[52] Wang, Z.; Garrett, C. R.; Kaelbling, L. P.; and Lozano-Perez, T. 2021. Learning compositional models of robot skills for task and motion planning. *International Journal of Robotics Research* .

[53] Yang, Q.; and D, T. J. 1990. ABTWEAK: Abstracting a Nonlinear, Least Commitment Planner. *Association for the Advancement of Artificial Intelligence (AAAI)* .

[54] Yoon, S.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *Journal of Machine Learning Research* 9(Apr): 683–718.

[55] Zeng, C.; and Silver, T. 2021. Learning Search Guidance from Failures with Eliminable Edge Sets. *ICAPS Workshop on Planning and Reinforcement Learning* .