Learning Search Guidance from Failures with Eliminable Edge Sets

Catherine Zeng[1], **Tom Silver**[2]
[1]Harvard College, [2]Massachusetts Institute of Technology
catherinezeng@college.harvard.edu, tslvr@mit.edu

Abstract

What can be learned from previous planning experience when none of it was successful in finding any plans? We study this question in the planning-as-graph-search setting. Our main insight is that certain *eliminable edge sets* can be identified from failed graph searches. These edge sets can then be used to train a generalized predictor of eliminable edges, which in turn can be used to guide search on new planning problems from the same domain. Our preliminary experimental findings across four visual navigation domains suggest that this technique of learning from failed search attempts can result in substantially improved planning in terms of the number of nodes expanded before finding a plan.

I have not failed. I've just found 10,000 ways that won't work.
THOMAS EDISON

1 Introduction

Search is a central paradigm in AI planning and beyond. In the *learning for planning* literature, a perennial question is: how can we leverage past search experience in similar problems to plan more efficiently and effectively in a new problem (Jiménez et al. 2012; Jiménez, Segovia-Aguas, and Jonsson 2019)? Approaches to this question almost always rely on *successful* previous experience — planning attempts that achieved their goal. In this work, we are interested in what can be learned from *failures* to plan in previous deterministic planning problems, where at the time of learning, we have not yet found any successful plans at all.

It is not immediately obvious that anything useful can be learned without examples of planning success. One approach suggesting the plausibility of this enterprise is dead-end detection (Helmert 2004; Lipovetzky, Muise, and Geffner 2016). 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. Unfortunately, there are many domains of interest where dead-ends are limited or absent altogether (e.g., domains where all actions are reversible). In these cases, can anything be learned from failed search attempts?

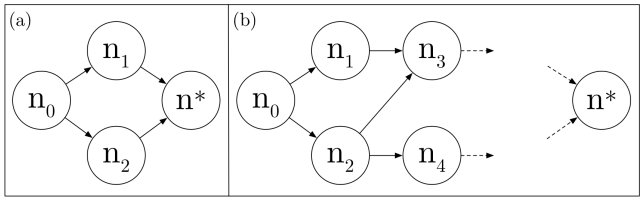


Figure 1: Two search graphs where node n_0 has the initial state and n^* has the goal. Edge labels omitted for clarity. (a) The edge set $\{(n_0, n_1), (n_1, n^*)\}$ is eliminable, since a plan through n_2 would remain in the graph with those edges removed. (b) Given a failed search attempt that has 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 this work, we consider *eliminable edge sets*: sets of edges that can be eliminated from a search graph without changing the solvability of the problem. See Figure 1 for an example. This notion of eliminability 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. 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 Related Work

Our work continues a long and active line of research in learning for planning (Jiménez et al. 2012). Approaches in this line include heuristic learning (Yoon, Fern, and Givan 2008; Arfaee, Zilles, and Holte 2011; Silver et al. 2016; Garrett, Kaelbling, and Lozano-Pérez 2016; Shen, Trevizan, and Thiébaux 2020) and generalized policy learning (Martin and Geffner 2004; Srivastava et al. 2011; Bonet and Geffner 2015; Groshev et al. 2018; Gomoluch, Alrajeh, and Russo 2019; Jiménez, Segovia-Aguas, and Jonsson 2019). In contrast to these works, we are interested in learning from failures alone, without any successful plans or demonstrations.

Learning from failed planning attempts is related to the challenge of exploration with sparse rewards in reinforcement learning (RL) (Kaelbling, Littman, and Moore 1996; Lehman and Stanley 2008; Bellemare et al. 2016; Andrychowicz et al. 2017; Nair et al. 2018; Ecoffet et al. 2019). Especially relevant is exploration in multitask RL; see Colas et al. (2020) 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 (Pathak et al. 2017; Burda et al. 2018). In our planning setting, we assume the transition model is known.

As discussed in Section 1, eliminable edge sets can be seen as a generalization of dead-ends (Helmert 2004; Lipovetzky, Muise, and Geffner 2016), which are closely related to nogoods in constraint satisfaction problems (Schiex and Verfaillie 1994; Katsirelos and Bacchus 2005). Previous work on learning dead-end detectors considers planning in factored, logical domains, and identifies formulaic representations of states that are verifiable dead-ends (Steinmetz and Hoffmann 2016, 2017a,b). In contrast, 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 mentioned above. Moreover, our proposed method provides leverage in domains that have no dead-ends to detect.

3 Problem Setting

We consider deterministic finite planning domains with states $\mathcal{S}$, actions $\mathcal{A}$, and transition function $\text{SUCC}(s, a) = s'$ 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 $s_{t+1} = \text{SUCC}(s_t, a_t)$ 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 $\text{SUCC}(s, a) = s'$. 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 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?

4 Learning A Generalized Predictor of Eliminable Edge Sets from Failed Searches

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.

4.1 Eliminable Edge Sets

Definition 4.1 (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 $s_T \in g$.

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.

4.2 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 4.2 (Expanded node, open node)**.** Given a subgraph $\mathcal{G}_{\text{sub}} = (\mathcal{V}_{\text{sub}}, \mathcal{E}_{\text{sub}})$ of a domain graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a node $n \in \mathcal{V}_{\text{sub}}$ is *expanded* if for all edges $(n, a, n') \in \mathcal{E}$, $(n, a, n') \in \mathcal{E}_{\text{sub}}$. Otherwise, the node is *open*.

Definition 4.3 (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 4.1. *Given a subgraph $\mathcal{G}_{sub} = (\mathcal{V}_{sub}, \mathcal{E}_{sub})$ of the domain graph $\mathcal{G}$, a planning problem (s_0, g) with $n_{s_0} \in \mathcal{V}_{sub}$, and an edge set $\mathcal{E}^- \subseteq \mathcal{E}_{sub}$, let $\mathcal{G}^-_{sub} = (\mathcal{V}_{sub}, \mathcal{E}_{sub} \setminus \mathcal{E}^-)$. Suppose **1)** no plan exists in $\mathcal{G}_{sub}$; **2)** all open nodes in $\mathcal{V}_{sub}$ are reachable in $\mathcal{G}^-_{sub}$; and **3)** all edges in $\mathcal{E}^-$ are reachable in $\mathcal{G}_{sub}$. Then $\mathcal{E}^-$ is eliminable.*

Proof. See Appendix A. □

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.

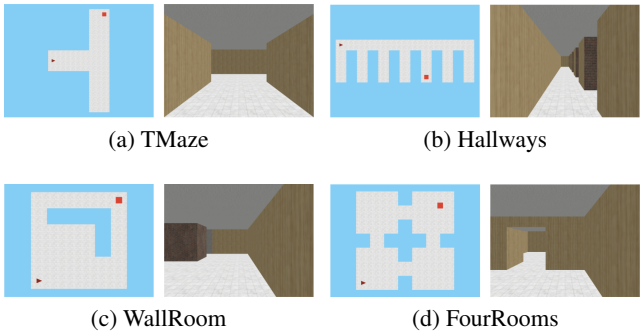
Identifying Eliminable Edges in Forward Searches. In practice, when planning with forward[1] search algorithms like GBFS or A*, we do not need to explicitly test whether edge sets satisfy the criteria of Lemma 4.1. At any given time in the execution of these algorithms, the shortest paths from the initial 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.

4.3 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:

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

[1]This work focuses on forward search, but we expect that it possible to formulate a version of Lemma 4.1 that would work for backward search as well, where expansion and reachability would emanate from the goal, rather than the initial state.



(a) TMaze (b) Hallways (c) WallRoom (d) FourRooms

Figure 2: Miniworld Domains

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

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.

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 4.1, 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 (Tang et al. 2017), 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 5.1 for details).

5 Experiments and Results

We now present preliminary experiments and results.

5.1 Experimental Setup

Domains. We test our approach in four visual navigation domains implemented in Miniworld (Chevalier-Boisvert 2018): `TMaze`, `Hallways`, `WallRoom`, and `FourRooms`. `Hallways` and `WallRoom` are custom domains original to this work. All domains involve an agent

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.

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'.[2]

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 4.3), we discretize the state space using locality-sensitive hashing (LSH). Details about the CNN and the LSH can be found in Appendix B.

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 (Kingma and Ba 2014) 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 in this paper is run with 25 random seeds, where randomness is introduced via neural network initialization, the SimHash A matrix (Appendix B), and the randomized tie-breaking during search. All search code is based on Pyperplan (Alkhazraji et al. 2020) and all neural-network code is written in PyTorch (Paszke et al. 2019).

[2]The initial state and goal are not factored into the feature representation because we train on only one problem per domain.

5.2 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 Appendix C, we report additional results that probe the impact of the unseen wrapper. We find that the CNN model without the unseen wrapper can perform substantially worse than a blind search, especially at low training percentages. The unseen wrapper allows the eliminability predictor to consistently perform better than blind search. It it is interesting to note that, even at a low `train_expansion=0.2`, the CNN can learn improvements in search that cannot be attributed solely to the unseen wrapper.

6 Discussion and Conclusion

In this work, 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 (Long and Fox 2003); (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.

References

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.

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* .

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

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.

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*.

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

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

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

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* .

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

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.

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*.

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

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.

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

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

Katsirelos, G.; and Bacchus, F. 2005. Generalized nogoods in CSPs. In *AAAI*, volume 5, 390–396.

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

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)*.

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.

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

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

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.

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.

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.

Schiex, T.; and Verfaillie, G. 1994. Nogood recording for static and dynamic constraint satisfaction problems. *International Journal on Artificial Intelligence Tools* 3(02): 187–207.

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*.

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.

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* .

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.

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

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

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.

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

A Proof of Lemma 4.1

Lemma 4.1. *Given a subgraph $\mathcal{G}_{sub} = (\mathcal{V}_{sub}, \mathcal{E}_{sub})$ of the domain graph $\mathcal{G}$, a planning problem (s_0, g) with $n_{s_0} \in \mathcal{V}_{sub}$, and an edge set $\mathcal{E}^- \subseteq \mathcal{E}_{sub}$, let $\mathcal{G}^-_{sub} = (\mathcal{V}_{sub}, \mathcal{E}_{sub} \setminus \mathcal{E}^-)$. Suppose **1)** no plan exists in $\mathcal{G}_{sub}$; **2)** all open nodes in $\mathcal{V}_{sub}$ are reachable in $\mathcal{G}^-_{sub}$; and **3)** all edges in $\mathcal{E}^-$ are reachable in $\mathcal{G}_{sub}$. 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 $\mathcal{E}^-$. □

B Experimental Details

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 (Tang et al. 2017), 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.

C Additional Experiments

We ran additional experiments to analyze the effect of the unseen wrapper. In these experiments, we 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` 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.

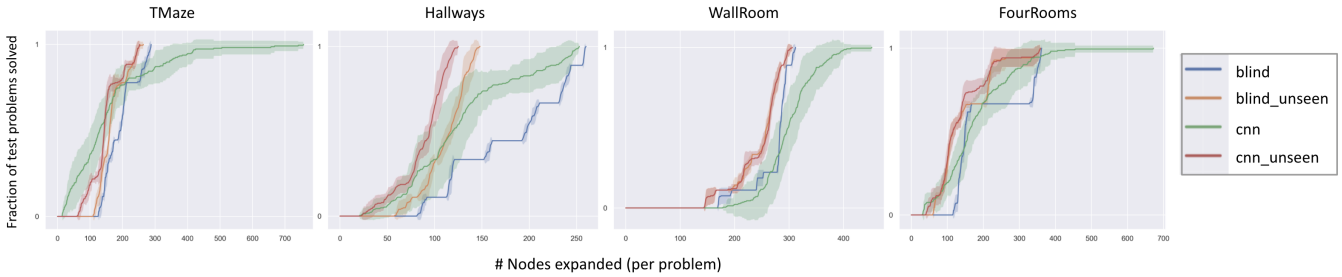


Figure 4: Models comparison at `training_expansion=0.2`

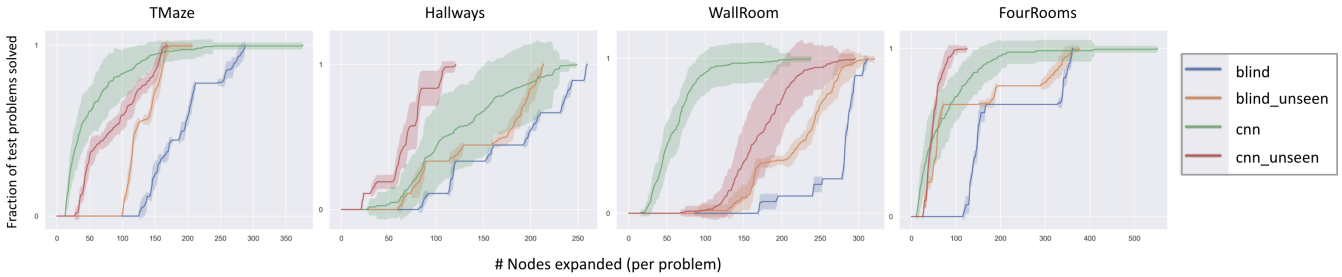


Figure 5: Models comparison at `training_expansion=0.8`