Learning Per-Domain Generalizing Policies Using Offline Reinforcement Learning

Nicola J. Müller[1,2], **Moritz Oster**[1,2], **Timo P. Gros**[1,2]

[1]German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
[2]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
{Nicola.Mueller, Moritz.Oster, Timo_Phillip.Gros}@dfki.de

Abstract

Learned per-domain generalizing policies are gaining popularity in classical planning, as they can solve arbitrary instances of a specific domain. They are typically trained using supervised learning (SL), where we learn to generalize beyond a training set, or reinforcement learning (RL), where we learn from scratch through trial-and-error. We argue that SL and RL should not be seen as contrasting approaches, and propose a training framework where a policy is first trained offline using SL, and then finetuned online using RL. The key method enabling this framework is offline RL. Preliminary experiments show that offline RL can indeed learn per-domain generalizing policies effectively.

1 Introduction

Learning approaches to per-domain generalization for classical planning are gaining increasing popularity in the automated planning community, as they enable training models that can solve arbitrary instances of a specific domain (Chen et al. 2024). Recent approaches used *supervised learning* (SL) to train policies that imitate optimal planners on small instances, such that they can generalize to larger instances at test time (Ståhlberg, Bonet, and Geffner 2022a, 2025). Despite their success, supervised learning approaches are fundamentally limited in the planning setting: (i) Optimal planning has a high computational cost, which limits the amount of available training data. It is also unclear whether optimal plans provide prediction targets well-suited for learning, as computing optimal policies is NP-hard for many classical planning domains (Ståhlberg, Bonet, and Geffner 2022b). (ii) Learned policies lack robustness as they are only trained on optimal trajectories, meaning that, at test time, their performance breaks down when deviating from optimal trajectories. This problem is exacerbated by the distributional shift between small training and large test instances, meaning policies need to make predictions for out-of-distribution states.

Some approaches increase robustness by periodically executing the policy and feeding the results back into the SL loop (Toyer et al. 2020; Rossetti et al. 2024; Gros et al. 2025). Nonetheless, the generalization of policies is always limited by the information in their training sets, as the distributional shift between training and test instances inevitably becomes too large when scaling the size of the latter[1].

Alternatively, per-domain generalizing policies can be trained using *reinforcement learning* (RL), which addresses limitations (i) & (ii) as the policies learn from various trajectories through trial-and-error without supervision. RL also allows us to continue the training as needed, removing the amount of training data as a limiting factor of generalization. However, existing RL approaches to policy learning customize the training loop with planning-specific components, such as Greedy Best First Search, as standard RL is inherently unstable and computationally inefficient (Rivlin, Hazan, and Karpas 2020; Ståhlberg, Bonet, and Geffner 2022b, 2023).

We argue that SL and RL should not be seen as contrasting approaches for per-domain generalization. To achieve the best of both worlds, we propose a training framework, where policies are first trained *offline* using suboptimal plans, and then finetuned *online* on automatically generated instances. This way we can efficiently pre-train policies without expensive data set generation, and then improve their performance as needed without learning from scratch.

Offline reinforcement learning (RL) methods are the key to enabling the offline-online training framework, as they adapt online RL methods to also learn policies from a fixed set of teacher trajectories (Prudencio, Maximo, and Colombini 2023). A central feature of offline RL methods is that they regularize learned policies, such that, at test time, they make conservative predictions for actions not seen during offline training, leading to robust performance during finetuning (Kumar et al. 2020). Further, offline RL methods can learn to improve over suboptimal teacher trajectories by combining the optimal parts of suboptimal trajectories, so-called stitching (Fu et al. 2020).

In Section 2, we will first give the necessary background on per-domain generalizing policies, SL, and RL. Then, we introduce our offline-online training framework and present a suitable offline RL algorithm in Section 3. Lastly, we present preliminary results for using offline RL in Section 4, and conclude by giving an outlook on the next steps of this work in Section 5.

[1]Except for very simple planning domains.

2 Background

We now introduce classical planning, per-domain generalizing policies and their two predominant learning approaches.

Classical Planning

A classical planning problem P can be represented as a pair $P = \langle D, I \rangle$, consisting of a domain D and an instance I (Ghallab, Nau, and Traverso 2004). Intuitively, the domain D describes the class of problems, whereas an instance I describes a specific problem. The domain D defines a set of predicate symbols p, and a set of action schemas describing the arguments, preconditions, effects, and costs of actions. The instance I defines a set of objects $o \in O$, the initial state $Init$, and a set of goal conditions $Goal$, where $Init$ and $Goal$ are sets of ground atoms $p(o_0, \ldots, o_n)$. Together, D and I encode a state model $S(P) = \langle \mathcal{S}, s_0, \mathcal{S}_G, Act, A, f \rangle$, consisting of a set of states $s \in \mathcal{S}$, the initial state s_0, the set of goal states $\mathcal{S}_G$, the set of ground actions Act, the sets of applicable actions for each state $A(s)$, and the transition function $f : \mathcal{S} \times Act \to \mathcal{S}$.

A plan is an action sequence $\vec{a} = \langle a_0, \ldots, a_{T-1} \rangle$ that transitions from the initial state s_0 to any goal state $s_T \in \mathcal{S}_G$. The cost of a plan $\vec{a}$ is defined as the sum over the costs of its actions

$$cost(\vec{a}) = \sum_{k=0}^{T-1} cost(a_k).$$

A plan $\vec{a}^*$ is optimal if it has the lowest possible cost.

Per-Domain Generalizing Policies

A policy $\pi : \mathcal{S} \to Act$ is a function returning action decisions for given states, and π^* is optimal if for every state s it returns an action a that starts an optimal plan $\vec{a}^*$ from s. We say π is a *per-domain generalizing policy*, if it can be applied to states from *any* instance I of a *fixed* domain D. As computing an optimal per-domain generalizing policy is NP-hard for many classical planning domains (Ståhlberg, Bonet, and Geffner 2022b), recent work has focused on learning near-optimal policies using machine learning methods, such as graph neural networks (GNNs) (Ståhlberg, Bonet, and Geffner 2022a). These methods typically train a policy π on a set of training instances, such that at test time, it can also solve instances not seen during training. In particular, π should be able to solve instances that are larger, i.e., have more objects, than the training instances, because the size of an instance typically correlates with the length of its optimal plans, and thus how difficult it is to solve. The two predominant approaches for learning per-domain generalizing policies are *supervised learning* (SL) and *reinforcement learning* (RL).

Supervised Learning

When using SL, we train a policy π to imitate an optimal planner on a set of training instances, such that at test time, it generalizes beyond its training data.

In SL, we are given a training data set of pairs (x_i, y_i), where x_i is an example from the input space $\mathcal{X}$ and y_i is the corresponding label from the output space $\mathcal{Y}$ (Goodfellow et al. 2016). Our goal is to learn a predictor $g : \mathcal{X} \to \mathcal{Y}$ that fits the training data well according to a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_0^+$.

A simple approach for learning policies using SL is to model a regression task with a data set consisting of pairs $(s_t, cost(\vec{a}^*))$, where $\vec{a}^*$ is an optimal plan starting from s_t (Ståhlberg, Bonet, and Geffner 2022a). We can then train a predictor $V : \mathcal{S} \to \mathbb{R}$, called a value function, that given a state s_t predicts the cost of an optimal plan $cost(\vec{a}^*)$. This allows us to compute a policy π as

$$\pi(s_t) = \arg\min_a cost(a) + V(s_{t+1}).$$

To train V, a natural choice of loss function is the *mean squared error* (MSE) loss

$$L^{\text{MSE}}(\theta) = \left[V(s_t) - cost(\vec{a}^*)\right]^2.$$

We note that alternative SL approaches exist in the literature, such as learning to rank successors (Chrestien et al. 2023).

Reinforcement Learning

When using RL, we train a policy π without any supervision through trial-and-error learning. We only provide rewards indicating how well π solves instances.

RL problems are modeled as *Markov decision processes* (MDPs) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mu \rangle$, which consist of a set of states $s \in \mathcal{S}$, a set of actions $a \in \mathcal{A}$, a transition probability function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{D}(\mathcal{S})$, and an initial state distribution $\mu \in \mathcal{D}(\mathcal{S})$ (Sutton, Barto et al. 1998). Additionally, we define a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that assigns a value to every state-action pair. The objective of RL is to learn an optimal policy

$$\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \cdot r(s_t, a_t)\right] \forall s_0 \sim \mu,$$

which maximizes the expected discounted sum of rewards, called the return G_0, for all initial states s_0. An optimal Q-value function $Q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ computes the expected return G_t when taking action a_t in state s_t and then following π^* from the state s_{t+1} onward, i.e.,

$$Q^*(s_t, a_t) = r(s_t, a_t) + \gamma \cdot \mathbb{E}\left[\sum_{k=t+1}^{T} \gamma^{k-t+1} \cdot r(s_k, a_k)\right].$$

Given a, possibly suboptimal, Q-value function Q we can compute a policy π as

$$\pi(s_t) = \arg\max_a Q(s_t, a).$$

In the classical planning setting, we define an MDP $\mathcal{M}$ for every instance I of a domain D according to the state model $S(P)$. Further, we define the reward function as $r(s, a) = -\text{cost}(a)$, such that, without discounting, i.e, $\gamma = 1.0$, maximizing the return corresponds to minimizing plan cost (Ferber, Helmert, and Hoffmann 2020). Hence, we will omit γ in examples specific to planning.

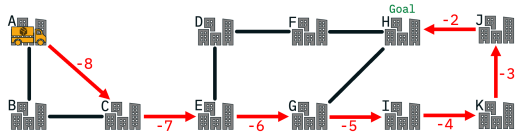
Deep Q-Network (DQN) is one of the standard learning algorithms in the RL literature (Mnih et al. 2013). Given

a collected experience (s_t, a_t, r_t, s_{t+1}), DQN computes the loss

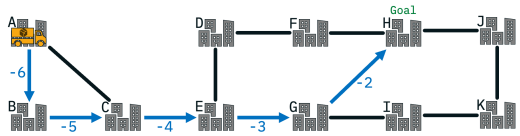
$$L^{\text{TD}}(\theta) = \left[r(s_t, a_t) + \gamma \cdot \max_{a'} Q_{\theta'}(s_{t+1}, a') - Q_\theta(s_t, a_t)\right]^2,$$

where the term $r(s_t, a_t) + \gamma \cdot \max_{a'} Q_{\theta'}(s_{t+1}, a')$ is called the temporal difference (TD) target. The so-called target network $Q_{\theta'}$ copies the parameters of Q_θ at a frequency smaller than Q_θ's update frequency to increase training stability. L^{TD} is well suited for online learning, as we can compute a loss for every collected experience, enabling frequent policy updates.

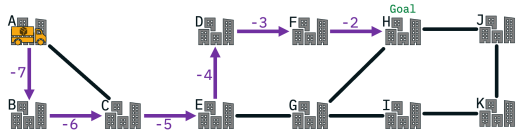
3 Offline Reinforcement Learning for Per-Domain Generalizing Policies



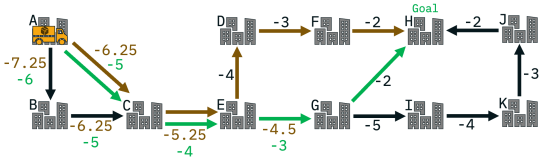
(a) Trajectory τ_r induced by plan $\vec{a}_r$.



(b) Trajectory τ_b induced by plan $\vec{a}_b$.



(c) Trajectory τ_p induced by plan $\vec{a}_p$.



(d) Q-values learned using $L^{\text{TD}}_{\text{off.}}$ (brown) and L^{IQL} (green). Black values are the same for both. Trajectories τ_{br} and τ_g are induced by the corresponding policies.

Figure 1: Examples for the simplified Transport domain.

In this section, we introduce a training framework that combines SL and RL. We then derive an offline RL algorithm that allows us to use SL and RL with a single loss function.

Offline Training and Online Finetuning

SL is an *offline* approach where the training data is provided beforehand, whereas RL is an *online* approach where the training data is collected on the fly. As such, SL seems more appealing for learning per-domain generalizing policies, as we can learn from existing optimal planners, whereas the trial-and-error learning of RL is known to be inefficient and unstable (Ladosz et al. 2022). On the other hand, RL allows us to use *any* instance for training, whereas in SL, our training data is limited to instances solvable by optimal planners.

To combine the best of both worlds, we propose a two-stage training framework that first trains a policy *offline*, and then finetunes it *online*:

1. *Offline Training:* We train a policy using SL on a data set of suboptimal plans, reducing the cost of data set generation. The data set also contains multiple suboptimal plans for each instance, increasing the robustness of policies.
2. *Online Finetuning:* We finetune the pre-trained policy using RL on automatically generated instances. Hence, learning is more efficient and stable because the policy is not trained from scratch. The instances can be generated as needed, e.g., we can gradually increase their size.

This offline-online training framework allows us to leverage planners to efficiently train policies without being limited by a fixed data set. However, the key challenge lies in transitioning from offline training to online finetuning, as we change from an SL to an RL loss function, which can cause catastrophic policy updates due to changing the optimization objective (Nair et al. 2020). In the following section, we introduce *offline reinforcement learning* (RL) which allows us to train offline and finetune online using a single loss function.

Offline Reinforcement Learning.

The goal of offline RL is to learn a policy π from a fixed set of trajectories

$$\tau = \langle (s_0, a_0, r_0, s_1), \ldots, (s_{T-1}, a_{T-1}, r_{T-1}, s_T) \rangle,$$

induced by an unknown teacher policy. A simple approach is to learn π by applying DQN to the trajectories τ. This means we can convert DQN to an offline algorithm by changing the source of training experiences from collecting on the fly, to a set of teacher trajectories. Crucially, this also means that we can finetune a policy trained using offline DQN by changing the data source again to collecting on the fly.

A challenge to using DQN offline is that, in its loss L^{TD}, the TD target $\max_{a'} Q_{\theta'}(s_{t+1}, a')$ computes the maximum estimate over *all* applicable actions in the successor state. This includes actions never observed in the training data, which can cause Q_θ to drastically overestimate their state-action values, leading to poor online performance (Kumar et al. 2020). We can avoid this by instead computing the loss

$$L^{\text{TD}}_{\text{off.}}(\theta) = \left[r(s_t, a_t) + \gamma \cdot Q_{\theta'}(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t)\right]^2.$$

However, $L^{\text{TD}}_{\text{off.}}$ should only be used if the training trajectories τ are optimal. Otherwise, we can observe multiple different sample returns G_t for the same state-action pairs (s_t, a_t), and thus $L^{\text{TD}}_{\text{off.}}$ is minimized when Q_θ approximates their average return instead of the maximum. This can cause the resulting policy π to perform *worse* than the training trajectories.

Consider the example in Figure 1, which shows an instance of the simplified Transport domain. The truck is located at city A and has loaded a package p which needs to be delivered to city H. We abbreviate states by the truck's

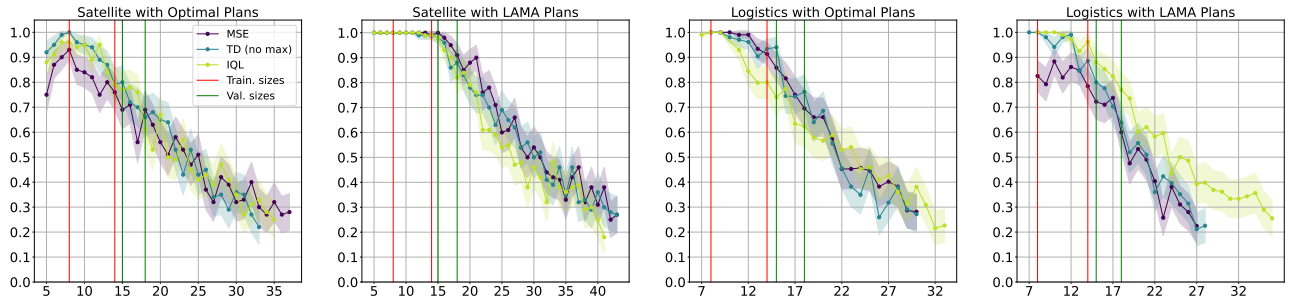


Figure 2: Average coverage over size for policies trained using L^{MSE}, $L^{\text{TD}}_{\text{off.}}$, or L^{IQL}. Vertical lines represent the data set sizes.

position, e.g, the initial state is $at(A)$. There are only $drive$ and $drop$ actions with cost 1. The colored arrows in Figures 1a,b,c represent three training trajectories τ_r, τ_b, and τ_p induced by the suboptimal plans $\vec{a}_r$, $\vec{a}_b$, $\vec{a}_p$, respectively. Each arrow is annotated with the return G_t that was earned from time step t onward. Figure 1d shows Q-values for each $drive$ action according to two Q-values functions, where the colored arrows represent their induced policies.

At the state $at(E)$, the optimal action is $drive(E, G)$. Only $\vec{a}_b$ and $\vec{a}_r$ select this action, whereas $\vec{a}_p$ selects the suboptimal action $drive(E, D)$. However, after selecting $drive(E, G)$, only $\vec{a}_b$ executes the optimal action sequence $\langle drive(G, H), drop(p, H)\rangle$. Hence, for the state-action pair $(at(E), drive(E, G))$, τ_b yields a return of -3, whereas τ_r yields return of -6. Now, a Q-value function Q_θ that minimizes $L^{\text{TD}}_{\text{off.}}$ (brown & black values) would predict the average return $Q(at(E), drive(E, G)) = -4.5$. Subsequently, we see in Figure 1d that the induced policy π_{br} takes the suboptimal action $drive(E, D)$ at the state $at(E)$, although a better action was observed during training. We could prevent this averaging over the returns for $(at(E), drive(E, G))$ by removing the training trajectory τ_r, but then, at the initial state, we would not observe the optimal action $drive(A, C)$.

In summary, we want to learn a Q-value function Q_θ from suboptimal trajectories, while ensuring that Q_θ approximates the maximum returns. *Implicit Q-learning* (IQL) (Kostrikov, Nair, and Levine 2021) achieves this by defining the loss function

$$L^{\text{IQL}}(\theta) = L_2^\tau \left[r(s_t, a_t) + \gamma \cdot Q_{\theta'}(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t)\right],$$

where $L_2^\tau(u) = |\tau - \mathbb{1}(u < 0)| \cdot u^2$. For $\tau > 0.5$, underapproximations of TD targets are penalized more than overapproximations, and thus for $\tau \approx 1.0$, L^{IQL} enforces approximating the maximum over TD targets. In practice, common values for τ are between 0.7 and 0.9.

Consider the example in Figure 1d again. The green & black values are the Q-values learned using L^{IQL}. We see that Q_θ predicts $Q_\theta(at(E), drive(E, G)) = -3$, meaning that at the state $at(E)$, the induced policy π_g takes the optimal action $drive(E, G)$. Starting from the initial state, following π_g yields an optimal trajectory τ_g with a return of -5. Note that this was *not* achieved by the training trajectories. Hence, by approximating the maximum over observed returns, Q_θ learned to combine the suboptimal trajectories τ_r and τ_b into a single optimal trajectory τ_g. This phenomenon is called *stitching* (Fu et al. 2020). We hypothesize that stitching can also occur in classical planning for trajectories from different instances. This is because neural network-based Q-value functions generalize over their training data, meaning states that are from different instances but share similar characteristics, will be mapped to the same internal representation.

4 Preliminary Experiments

We train policies using the GNN approach of Ståhlberg, Bonet, and Geffner (2022a) on the Satellite and Logistics domains of the IPC'23 (Taitler et al. 2024) using the L^{MSE}, $L^{\text{TD}}_{\text{off.}}$, and L^{IQL} losses. The training is repeated using two data sets that share the same instances but contain either optimal or suboptimal plans computed using Fast-Downward's *seq-opt-merge* configuration (Helmert 2006) or LAMA (Richter and Westphal 2010). For each configuration, the policy with the best validation loss is evaluated using Gros et al.'s scaling behavior evaluation (2025).

Looking at Figure 2, we see that all loss functions yield policies with similar performance when training on optimal plans. On Satellite, we see that all policies trained on LAMA plans achieve similar performance, which is better than their counterparts trained on optimal plans. This unexpected result suggests that there may be a benefit to learning from suboptimal plans in general. On Logistics, we see that policies trained using the L^{MSE} or $L^{\text{TD}}_{\text{off.}}$ loss perform worse when training on LAMA plans, whereas L^{IQL} yields a policy that scales four sizes larger. This suggests IQL can indeed leverage the information in suboptimal plans through stitching.

5 Conclusion

We proposed a training framework for per-domain generalizing policies where a policy is first trained offline using suboptimal plans, and then finetuned online using automatically generated instances. This combines the efficiency of SL with the flexibility of RL. Offline RL is the key method for leveraging suboptimal plans during offline training, and subsequently transitioning to online finetuning. Our preliminary results suggest that well performing policies can indeed be learned from suboptimal plans. Next, we want to test online finetuning, while also considering alternative offline RL methods. We also want to construct data sets specifically for testing the stitching capabilities of policies.

Acknowledgements

This work was partially supported by the German Federal Ministry of Education and Research (BMBF) as part of the project MAC-MERLin (Grant Agreement No. 01IW24007), by the German Research Foundation (DFG) - GRK 2853/1 "Neuroexplicit Models of Language, Vision, and Action" - project number 471607914, and by the European Regional Development Fund (ERDF) and the Saarland within the scope of (To)CERTAIN.

References

Chen, D. Z.; Hao, M.; Thiébaux, S.; and Trevizan, F. 2024. Graph Learning for Planning: The Story Thus Far and Open Challenges. *arXiv preprint arXiv:2412.02136*.

Chrestien, L.; Edelkamp, S.; Komenda, A.; and Pevny, T. 2023. Optimize planning heuristics to rank, not to estimate cost-to-goal. *Advances in Neural Information Processing Systems*, 36: 25508–25527.

Ferber, P.; Helmert, M.; and Hoffmann, J. 2020. Reinforcement learning for planning heuristics. In *Proceedings of the 1st workshop on bridging the gap between AI planning and reinforcement learning (PRL)*, 119–126.

Fu, J.; Kumar, A.; Nachum, O.; Tucker, G.; and Levine, S. 2020. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Goodfellow, I.; Bengio, Y.; Courville, A.; and Bengio, Y. 2016. *Deep learning*, volume 1. MIT press Cambridge.

Gros, T. P.; Müller, N. J.; Fiser, D.; Valera, I.; Wolf, V.; and Hoffmann, J. 2025. Per-Domain Generalizing Policies: On Validation Instances and Scaling Behavior. arXiv:2505.00439.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Kostrikov, I.; Nair, A.; and Levine, S. 2021. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*.

Kumar, A.; Zhou, A.; Tucker, G.; and Levine, S. 2020. Conservative q-learning for offline reinforcement learning. *Advances in neural information processing systems*, 33: 1179–1191.

Ladosz, P.; Weng, L.; Kim, M.; and Oh, H. 2022. Exploration in deep reinforcement learning: A survey. *Information Fusion*, 85: 1–22.

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

Nair, A.; Gupta, A.; Dalal, M.; and Levine, S. 2020. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*.

Prudencio, R. F.; Maximo, M. R.; and Colombini, E. L. 2023. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE Transactions on Neural Networks and Learning Systems*, 35(8): 10237–10257.

Richter, S.; and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Rivlin, O.; Hazan, T.; and Karpas, E. 2020. Generalized planning with deep reinforcement learning. *arXiv preprint arXiv:2005.02305*.

Rossetti, N.; Tummolo, M.; Gerevini, A. E.; Putelli, L.; Serina, I.; Chiari, M.; and Olivato, M. 2024. Learning general policies for planning through GPT models. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 500–508.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022a. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 629–637.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022b. Learning generalized policies without supervision using gnns. *arXiv preprint arXiv:2205.06002*.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2023. Learning general policies with policy gradient methods. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, 647–657.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2025. Learning More Expressive General Policies for Classical Planning Domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 26697–26706.

Sutton, R. S.; Barto, A. G.; et al. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; et al. 2024. The 2023 international planning competition.

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. Asnets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68: 1–68.