On the Possibility of Solving STRIPS Planning Problems with REINFORCE

Anonymous submission

Abstract

Along with the development of neural networks, there has been a growing interest in using machine learning and deep learning techniques to solve symbolic planning problems. Almost all the existing learning-based approaches rely on pretraining on a set of optimal or near-optimal trajectories to learn a heuristic function (which is latter used in a search), a Q-function, or a policy. In this paper, we explore the possibility of solving symbolic planning problems via pure reinforcement learning, without any pretraining and search. The experimental results show that, while the performance is still far from being competitive, it is indeed possible to do so, and hence, it is hope that this line of work could provide some insights into developing new RL-based approach for symbolic planning and developing online fine-tuning frameworks.

Introduction

Symbolic Planning refers to the task of finding a sequence of actions that achieves a certain goal. The framework has been widely used to model many practical scenarios, e.g., production line logistics (Helmert and Lasinger 2010) and Mars Exploration (Ai-Chang et al. 2004). It also serves as an important building block for robot manipulation, especially in task and motion planning (TAMP) (Dantam et al. 2018; Garrett et al. 2021). Symbolic planning is a combinatorial optimisation problem that has been proven to be $\mathbb{PSPACE}$-complete (Bylander 1994). A planning problem is typically solved by search-based approaches, e.g., A* search equipped with heuristics (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Helmert and Domshlak 2009; Pommerening et al. 2014; Seipp and Helmert 2018). Along with the raise of neural networks, there has been an increasing interest in deploying learning-based approaches (Toyer et al. 2018; Ståhlberg, Bonet, and Geffner 2022; Chen, Thiébaux, and Trevizan 2024; Chen, Trevizan, and Thiébaux 2024; Ståhlberg, Bonet, and Geffner 2025; Corrêa, Pereira, and Seipp 2025) on planning problems.

Almost all of those learning-based approaches rely on offline pretraining. Concretely, there are three main threads of approaches: 1) Using imitation learning to learn a policy from optimal trajectories. (Toyer et al. 2018) 2) Learning a heuristic which is then integrated into a search process (Chen, Thiébaux, and Trevizan 2024; Chen, Trevizan, and Thiébaux 2024; Corrêa, Pereira, and Seipp 2025) 3) Learning a Q-function from optimal (or suboptimal) trajectories which serves as a backbone of a policy (Ståhlberg, Bonet, and Geffner 2023; Ståhlberg, Bonet, and Geffner 2025). In this short paper, we want to investigate the possibility of directly using reinforcement learning (RL) to solve planning problems without any pretraining and search. This is a challenging task because a planning problem essentially induces an MDP which is extremely sparse in rewards. Concretely, a reward is only given when the goal is achieved. This raises the contradiction that once the goal is achieved, the problem is solved, but before that, the learning algorithm receives no training signal.

We address this issue by utilising the idea of reward shaping where we use the heuristic value of the *last state* in a collected trajectory as a proximal reward. This is very similar to the approach by Gehring et al. (2022). However, in their work, they again learn a Q-function which is then used to guide a search process. Here, we directly use the proximal reward to train a policy which reaches the goal without any search via policy gradient (REINFORCE) (Williams 1992; Sutton et al. 1999). As we will show in the later section, by using the this proximal reward, the objective function optimized by REINFORCE could have a nice mathematical interpretation.

Preliminaries

We consider symbolic planning problems formalised in the classical STRIPS framework (Fikes and Nilsson 1971). A planning problem is a tuple $\Pi = (\mathcal{F}, \mathcal{A}, s_I, g)$:

- $\mathcal{F}$ is a set of propositions called *facts*. An element $s \in 2^{\mathcal{F}}$ is called a *state*.
- $\mathcal{A}$ is a set of *actions*. An action $a \in \mathcal{A}$ is characterised by its *preconditions* $\texttt{pre}(a) \subseteq \mathcal{F}$, *add effects* $\texttt{add}(a) \subseteq \mathcal{F}$, *delete effects* $\texttt{del}(a) \subseteq \mathcal{F}$, and cost $c(a) \in \mathbb{N}$. An action a is applicable in a state s if $\texttt{pre}(a) \subseteq s$. Applying the action a in the state s will result in a new state $s' = (s \setminus \texttt{del}(a)) \cup \texttt{add}(a)$, denoted as $s \rightarrow_a s'$. Note that this notation already implies that a is applicable in s.
- $s_I \in 2^{\mathcal{F}}$ is the initial state.
- $g \subseteq \mathcal{F}$ is the goal condition.

Solving a planning problem Π is to find a sequence of actions $\overline{a} = \langle a_0, a_1, \cdots, a_n \rangle$ with some $n \in \mathbb{N}$ which induces the trajectory $\tau = \langle s_0, a_0, s_1, a_1, \cdots, s_{n+1} \rangle$ such that $s_0 = s_I$, $s_{n+1} \supseteq g$, and $s_i \rightarrow_{a_i} s_{i+1}$ for each $0 \leq i \leq n$.

Algorithm 1: REINFORCE with heuristic reward shaping

$\pi_0 \leftarrow$ randomly initialized policy
▷ Main training loop
for $i = 0, 1, 2, \ldots$ **do**
  $H \leftarrow$ randomly selected horizon
  ▷ Use a heuristic h as the reward for the last step
  $\tau \leftarrow \langle s_0, a_0, -c(a_0), \ldots, s_{H-1}, a_{H-1}, -h(s_H), s_H \rangle$
  ▷ a_i is non-deterministically sampled from $\pi_i(s_i)$
  if $g \subseteq s_H$ **then**
    return τ ▷ A solution is found
  $\pi_{i+1} \leftarrow$ REINFORCE(π_i, τ)

The cost of the plan $\overline{a}$ is the sum of each action's cost, i.e., $c(\overline{a}) = \sum_{i=0}^{n} c(a_i)$.

A planning problem Π can be naturally formulated as a Markov Decision Process (MDP). An MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{P}_I, \gamma)$ where $\mathcal{S}$ is a set of states (state space), $\mathcal{A}$ is a set of actions (action space), $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function such that $\mathcal{T}(s, a, s')$ is the probability of transitioning to the state s' after taking action a in s, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\mathcal{P}_I : \mathcal{S} \to [0, 1]$ is the initial state distribution, and $\gamma \in (0, 1]$ is the discount factor.

A straightforward way to represent a planning problem as an MDP is as follows: The state space $\mathcal{S} = 2^{\mathcal{F}}$. The action space $\mathcal{A}$ is the same as the action set in the planning problem. The transition function $\mathcal{T}$ is such that $\mathcal{T}(s, a, s') = 1$ if $s \to_a s'$, and $\mathcal{T}(s, a, s') = 0$ otherwise. The reward function is defined as $\mathcal{R}(s, a) = -c(a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. The initial state distribution $\mathcal{P}_I$ is such that $\mathcal{P}_I(s_I) = 1$ and $\mathcal{P}_I(s) = 0$ for all $s \neq s_I$. The discount factor γ is set to 1.

Reinforcement Learning Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{P}_I, \gamma)$ be an MDP, a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ maps a state to a distribution over actions. We consider the finite-horizon setting (because it aligns with the nature of planning problems). Let $H \in \mathbb{N}$ be the horizon. Running the policy π from an initial state $s_0 \sim \mathcal{P}_I$ will generate a trajectory

$$\tau = \langle s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_H \rangle$$

where $a_i \sim \pi(s_i)$, $r_i = \mathcal{R}(s_i, a_i)$, and $\mathcal{T}(s_i, a_i, s_{i+1}) > 0$ for each $0 \leq i < H$. The return (i.e., cumulative reward) of the trajectory τ is $R(\tau) = \sum_{i=0}^{H-1} \gamma^i r_i$ where γ is the discount factor. Due to the stochasticity of the policy and the transition, the trajectory governed by π is probabilistic. The objective of reinforcement learning is to find a policy π that maximizes average return of all trajectories generated by π, i.e., the expected return $\mathbb{E}_{\tau \sim \pi}[R(\tau)]$.

Heuristic as Proximal Reward

We consider solving the MDP induced by a planning problem via policy gradient. There are many variants of this algorithm, from its vanilla version, REINFORCE (Williams 1992; Sutton et al. 1999), to more advanced ones like Actor-Critic and PPO (Schulman et al. 2017). Since our method is orthogonal to the choice of the specific version, we will simply use REINFORCE to refer to the family of this algorithm. On a high level, REINFORCE is an on-policy algorithm which iteratively updates the policy based on the cumulative reward of a trajectory sampled according to the latest policy. However, due to the extreme sparse reward structure in the MDP formulation of a planning problem, directly deploying REINFORCE on it to solve the problem is no different from a random walk. To address this issue, we utilize the idea of reward shaping as follows: When sampling a trajectory

$$\tau = \langle s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{H-1}, a_{H-1}, r_{H-1}, s_H \rangle$$

where $s_0 = s_I$, H is a *randomly selected* horizon, and for all $0 \leq i \leq H - 1$, a_i is the action sampled according to the current policy, and r_i is the corresponding reward received, we replace r_{H-1} with a negated *heuristic value* of s_H, i.e., $r_{H-1} = -h(s_H)$, where h is a heuristic function, and for all $0 \leq i < H - 1$, we keep $r_i = -c(a_i)$ unchanged. The main procedure is shown in Alg. 1.

The idea of using a heuristic to shape the reward was also exploited by Gehring et al. (2022). In their work, they compute a heuristic value for *every* state in a trajectory and use potential-based reward shaping to rectify the reward. They also considered the discounted setting. Concretely, for the i-th step in a trajectory, their shaped reward would be

$$r_i = -c(a_i) + (\gamma h(s_{i+1}) - h(s_i))$$

where γ is the discount factor. If $\gamma = 1$, the cumulative reward of a trajectory under their shaping will be very similar to the one under our shaping. However, there is some subtle differences which could cause performance differences (see the discussion in the evaluation section), and in our setting, we only need to compute the heuristic value for the last state in a trajectory.

Mathematical Interpretation For any state s, we assume the case that the heuristic value $h(s)$ is *greater* than the optimal cost to reach the goal from s, denoted as $c^*(s)$ (that is to say, h is inadmissible). Given a horizon H and a policy π, the cumulative reward of a trajectory τ sampled from π with the shaped reward is

$$r(\tau) = \sum_{i=0}^{H-1} r_i = (\sum_{i=0}^{H-2} -c(a_i)) + (-h(s_H))$$

Since $-h(s_H) \leq -c^*(s_H)$, we have that

$$r(\tau) \leq (\sum_{i=0}^{H-2} -c(a_i)) + (-c^*(s_H))$$

The above right-hand side is the negative of the optimal cost to reach the goal under the latest policy. Thus, REINFORCE is maximizing a *lower bound* of this value. In other words, it is minimizing an *upper bound* of the optimal cost to reach the goal under the current policy.

One remark is that, if the heuristic used is admissible, i.e., $h(s) \leq c^*(s)$ for all s, the objective being maximized will then become an *upper* bound of the actual objective. This could potentially make the optimization less effective, which might be counter-intuitive. This is also partially supported by our empirical results.

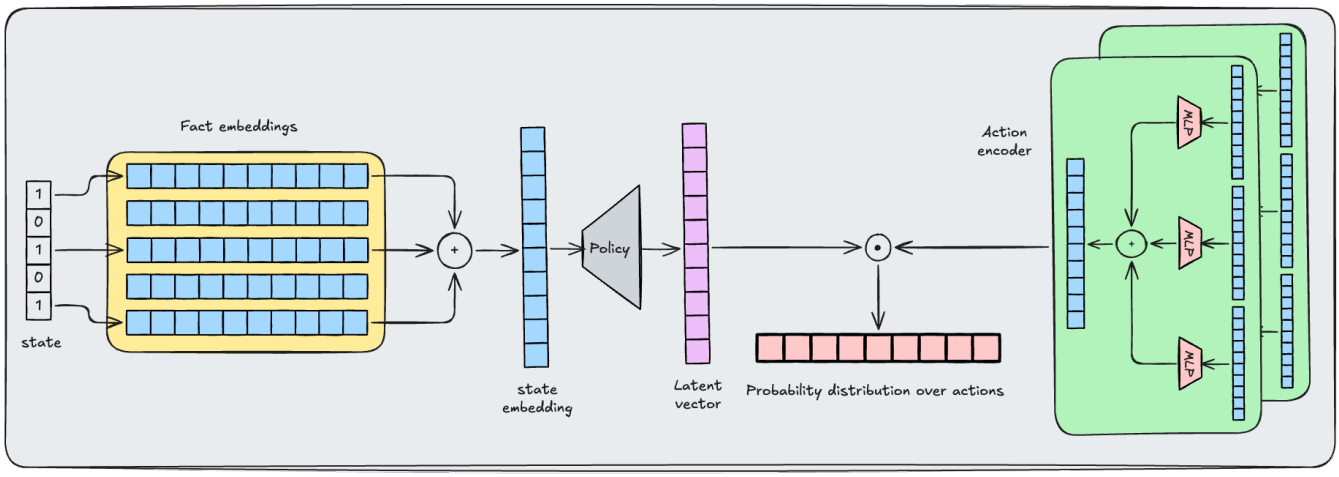

Figure 1: An example of state and action encodings with five facts. The fact embeddings are shared across both state and action encodings and are learnable. The three MLPs in the action encoder are also shared across all actions.

Encoding When using REINFORCE to optimize the policy, the policy is represented as a neural network which takes as input an encoding of a state and outputs a distribution over all actions. One straightforward way to encode a state is to use its binary representation, i.e., the input is a binary vector whose length is the number of facts in the planning problem, and every bit in the vector represents the presence or absence of a fact in a state. Similarly, the output of the neural network is also a vector whose dimension is the number of actions. The problem with this simple encoding is that the number of facts and actions in a planning problem can be very large while only a small portion of them are activated in a state or are applicable. This makes the neural network become very sparse and thus hard to train.

The mainstream approach to resolve this issue is to represent a planning problem as a graph and then use a graph neural network (GNN) to process it. However, such an approach normally requires a large GNN and is subject to the expressiveness of GNNs (Barceló et al. 2020). Hence, this approach is not suitable when using purely online reinforcement learning. In this paper, we consider a simple encoding method inspired by word embedding (Mikolov et al. 2013).

Concretely, for each fact $f \in \mathcal{F}$ in the planning problem, we assign it a *learnable* vector $\mathbf{v}_f$. The encoding of a state s, $\mathbf{e}(s)$, is then simply the sum of the vectors of all facts that are true in s, i.e., $\mathbf{e}(s) = \sum_{f \in s} \mathbf{v}_f$. Note that this encoding is invariant to the permutation of facts.

As mentioned earlier, if we simply let the output of the policy network be a vector of dimension equal to the number of actions, we will again encounter the issue of sparsity and make training difficult. To address this issue, we let the policy network output a latent vector and construct an action encoder to map each action into the same latent space. Then, to get a distribution over actions, we compute the inner product between the output of the policy network and the encoding of each action, and then apply a softmax function to the results.

The action encoder consists of three independent multi-layer perceptrons (MLPs), corresponding to the preconditions, add effects, and delete effects of an action. The input of these MLPs is the embedding of respective component of the action which is obtained in the same way as the state encoding. For instance, for an action a, the embedding of its preconditions $\texttt{pre}(a)$ is $\sum_{f \in \texttt{pre}(a)} \mathbf{v}_f$, which serves as the input to the corresponding MLP. The final encoding of an action is the sum of the outputs of the three MLPs. Note that these MLPs are shared across all actions, which reduces the number of parameters and resolves the issue of sparsity. An example of the encoding is shown in Fig. 1.

Domain	**With Encoder**		**Without Encoder**		**Total**
	hFF	hMax	hFF	hMax	
barman	0.00%	0.00%	0.00%	0.00%	20
blocksworld	25.71%	25.71%	37.14%	31.43%	35
floortile	0.00%	0.00%	0.00%	0.00%	20
grid	40.00%	20.00%	40.00%	20.00%	5
gripper	45.00%	25.00%	50.00%	35.00%	20
logistics	67.86%	28.57%	60.71%	28.57%	28
miconic	54.67%	42.00%	38.67%	33.33%	150
rovers	42.50%	32.50%	40.00%	35.00%	40
satellite	22.22%	13.89%	19.44%	13.89%	36
scanalyzer	40.00%	40.00%	40.00%	40.00%	30
sokoban	0.00%	0.00%	0.00%	0.00%	30
storage	53.33%	50.00%	56.67%	53.33%	30
transport	23.33%	13.33%	20.00%	13.33%	30
visitall	0.00%	0.00%	0.00%	0.00%	20
Average	**29.62%**	**20.79%**	**28.76%**	**21.71%**	—

Table 1: The percentage of instances for each domain for which a solution can be found within 1M training steps.

Evaluation

We ran the experiments on some benchmark domains from the International Planning Competition (IPC). We used two heuristics in our empirical evaluation: hFF (Hoffmann and Nebel 2001) and hMax (Bonet and Geffner 2001). The latter

Domain	Proximal	Potential	Total
barman	0.00%	0.00%	20
blocksworld	25.71%	31.43%	35
floortile	0.00%	0.00%	20
grid	40.00%	40.00%	5
gripper	45.00%	40.00%	20
logistics	67.86%	53.57%	28
rovers	42.50%	40.00%	40
sokoban	0.00%	0.00%	30
visitall	0.00%	0.00%	20
Average	**24.56%**	**22.78%**	—

Table 2: The comparison between using heuristics at the last state (proximal) and using them with potential-based reward shaping (potential).

one is an admissible heuristic while the first one is not. For each heuristic, we further had two settings: with and without an action encoder. The REINFORCE algorithm we used is PPO (Schulman et al. 2017).

We logged the percetage of solved instances for each domain within the budget of 1M PPO training steps. The results are shown in Tab. 1. As can be seen, using the action encoder could in general improve the performance. Furthermore, using hFF as the proximal reward is consistently better than using hMax despite that the latter one is admissible. This is not surprising because, as discussed earlier, when using an inadmissible heuristic like hFF as the proximal reward, REINFORCE is maximizing a lower bound of the actual objective (if the heuristic is admissible, it is maximizing the upper bound). The optimization process would be more effective if the lower bound is tighter, which is the case with hFF. Despite that REINFORCE failed completely on some domains, we could conclude that it is indeed possible to solve STRIPS planning problems wihtout any pretraining and search. For some of those failed domains, we hypothesize that the reason for the failure is the existence of dead-end states. In our setting, we set the heuristic value of a dead-end state to a relatively large number (e.g., 1000). This could further exacerbate the known high variance issue of REINFORCE.

We further compared our setting against using potential-based reward shaping with heuristics (Gehring et al. 2022), that is, running PPO with the potential-based shaped reward. The two approaches were run with the same setting: using hFF as the heuristic and using the action encoder. We again logged the percentage of solved instances for each domain within 1M PPO training steps.

At the first glance, it is surprising that using the potential-based reward shaping has a worse performance. Intuitively, using the potential-based shaping could provide a more informative reward signal per step, that is, in the middle of a sampled trajectory, if the policy takes a state which has a smaller heuristic value to the next state which has a larger heuristic value, a large negative reward will be given. In the opposite case, a positive reward will be given. In our setting, there is no such local awareness.

To study this more carefully, we take a closer look at the objective function being optimized when the potential-based reward shaping is used. Consider a trajectory

$$\tau = \langle s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_{H-1}, a_{H-1}, r_{H-1}, s_H \rangle$$

where $s_0 = s_I$, H a horizon, and for all $0 \leq i \leq H-1$, r_i is the potential-based shaped reward, and a_i is sampled according to the current policy. For simplicity, we consider the case where the discount factor γ is set to 1. Consequently, $r_i = -c(a_i)+(h(s_i)-h(s_{i+1}))$ for all $0 \leq i \leq H-1$. Clearly, the cumulative reward of τ is thus $(\sum_i -c(a_i)-h(s_H))+h(s_0)$. One could see that this objective is essentially the same as the one optimized in our setting except that there is an additional constant term $h(s_0)$. As we have discussed, our objective function is a lower bound of the actual objective, and the optimization will be more effective if this bound is tight. However, by adding this additional constant term $h(s_0)$, the bound might become looser. Theoretically, having this additional constant term will not affect the optimization over the objective function. However, in practice, the issue of high variance of REINFORCE could make a difference.

Conclusion

In this paper, we explored the idea of using reward shaping to solve symbolic planning problems directly with policy gradient without any pretraining and search. Despite that the performance is still far from practical, we showed that it is indeed possible to do so. It is thus hoped that this could provide some new insights into using RL in symbolic planning, particularly in the context that most existing RL-based approaches for planning focus on Q-function learning, and policy gradient methods are less explored. Additionally, almost all existing learning-based approaches for planning rely on offline learning. There is a lack of online fine-tuning frameworks for planning, and the message of this paper could be a step towards developing such frameworks.

Future Work One promising line of future work is to replace the heuristic function with a learnable distance function $f(s, s')$ which estimates the distance from one state s to another state s' (note that s' could also be a partial state) and which could be learned on-the-fly along with reinforcement learning. In other words, the role of the heuristic function in our method is now played by this distance function which predicts the distance from the last state in a trajectory to the goal. This is closely related to the idea of hindsight experience replay (Andrychowicz et al. 2017) and universal value function approximation (Schaul et al. 2015).

References

Ai-Chang, M.; Yglesias, J.; Chafin, B.; Dias, W.; Maldague, P.; Bresina, J.; Charest, L.; Chase, A.; Hsu, J.-J.; Jonsson, A.; Kanefsky, B.; Morris, P.; and Rajan, K. 2004. MAPGEN: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission. *IEEE Intelligent Systems*, 19(1): 8–12.

Andrychowicz, M.; Crow, D.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; and

Zaremba, W. 2017. Hindsight Experience Replay. In *Proceedings of the 31st Advances in Neural Information Processing Systems, NeurIPS 2017*, 5048–5058. Curran Associates.

Barceló, P.; Kostylev, E. V.; Monet, M.; Pérez, J.; Reutter, J. L.; and Silva, J. P. 2020. The Logical Expressiveness of Graph Neural Networks. In *Proceedings of the 8th International Conference on Learning Representations, ICLR 2020*, 1–11. OpenReview.

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence*, 129(1-2): 5–33.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69(1-2): 165–204.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. W. 2024. Learning Domain-Independent Heuristics for Grounded and Lifted Planning. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence, AAAI 2024*, 20078–20086. AAAI.

Chen, D. Z.; Trevizan, F. W.; and Thiébaux, S. 2024. Return to Tradition: Learning Reliable Heuristics with Classical Machine Learning. In *Proceedings of the 34th International Conference on Automated Planning and Scheduling, ICAPS 2024*, 68–76. AAAI.

Corrêa, A. B.; Pereira, A. G.; and Seipp, J. 2025. Classical Planning with LLM-Generated Heuristics: Challenging the State of the Art with Python Code. *CoRR*, abs/2503.18809.

Dantam, N. T.; Kingston, Z. K.; Chaudhuri, S.; and Kavraki, L. E. 2018. An incremental constraint-based framework for task and motion planning. *International Journal of Robotics Research*, 37(10): 1134–1151.

Fikes, R.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3/4): 189–208.

Garrett, C. R.; Chitnis, R.; Holladay, R.; Kim, B.; Silver, T.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Integrated Task and Motion Planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1): 265–293.

Gehring, C.; Asai, M.; Chitnis, R.; Silver, T.; Kaelbling, L. P.; Sohrabi, S.; and Katz, M. 2022. Reinforcement Learning for Classical Planning: Viewing Heuristics as Dense Reward Generators. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling, ICAPS 2022*, 588–596. AAAI.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009*, 162–169. AAAI.

Helmert, M.; and Lasinger, H. 2010. The Scanalyzer Domain: Greenhouse Logistics as a Planning Problem. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010*, 234–237. AAAI.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of the 1st International Conference on Learning Representations, ICLR 2013*, 1–12. OpenReview.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-Based Heuristics for Cost-Optimal Planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling, ICAPS 2014*, 226–234. AAAI.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal Value Function Approximators. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, 1312–1320. JMLR.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *Journal of Artificial Intelligence Research*, 62: 535–577.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022. Learning Generalized Policies without Supervision Using GNNs. In *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022*, 474–483. IJCAI.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2023. Learning General Policies with Policy Gradient Methods. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023*, 647–657. IJCAI.

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

Sutton, R. S.; McAllester, D. A.; Singh, S.; and Mansour, Y. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Proceedings of the 12th Advances in Neural Information Processing Systems, NIPS 1999*, 1057–1063. MIT.

Toyer, S.; Trevizan, F. W.; Thiébaux, S.; and Xie, L. 2018. Action Schema Networks: Generalised Policies With Deep Learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 6294–6301. AAAI.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8: 229–256.