LEARNING HEURISTIC FUNCTIONS VIA POLICY OPTIMIZATION FOR SYMBOLIC PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Tree search efficiency in deterministic symbolic planning critically depends on heuristic functions, which are difficult to design. We revisit heuristic learning from a policy optimization perspective and formulate symbolic planning search as a Markov Decision Process (Tree-MDP), where states are partially expanded search trees, and actions select nodes for expansion. This enables standard Reinforcement Learning (RL) algorithms, such as Proximal Policy Optimization (PPO), to optimize heuristics directly. Crucially, this perspective reveals that ranking-based losses are closely related to policy optimization objectives. Building on this unified view, we propose a new loss function that combines RL principles with structural properties of symbolic search, improving robustness across test domains. Experiments on IPC 2023 learning track benchmarks support the theoretical connections and demonstrate that the approach reliably delivers strong performance.

1 INTRODUCTION

The performance of tree search algorithms such as A* (Hart et al., 1968) and Greedy Best-First Search (GBFS) (Nilsson, 1971), which are among the most widely used methods in AI, is fundamentally limited by the quality of heuristic functions. While A* with admissible heuristics guarantees optimal solutions, GBFS prioritizes *search efficiency*—finding solutions quickly by expanding fewer nodes. This work focuses on improving the search efficiency of GBFS with learned heuristics.

Because designing heuristic functions is challenging, many works use machine learning techniques (Ferber et al., 2020; Shen, 2020). One core problem is the representation of the symbolic states and the corresponding architecture of the neural network (Toyer et al., 2020; Shen, 2020; Horcík & Sír, 2024; Chen et al., 2024; Ståhlberg et al., 2022). A complementary direction focuses on the criteria (loss functions) optimized during the learning of heuristic functions. Recently proposed ranking losses (Chrestien et al., 2023; Hao et al., 2024) experimentally perform better than popular cost-to-go estimates.

Most of the above works consider the imitation learning scenario, which assumes a set of problem instances together with solutions provided by an existing solver. The opposite bootstrap learning scenario, where solutions are unavailable and bootstrapping (Arfaee et al., 2011) from unsolved problem instances is required, is considerably less understood. Bootstrap learning naturally aligns with reinforcement learning: both systems learn from delayed scalar rewards indicating solution quality, and both leverage value or policy functions to guide decisions. Yet RL algorithms remain underutilized in planning, although some works (Chen & Tian, 2019; Zoph & Le, 2017; Anderson et al., 2018; Ståhlberg et al., 2023) use RL algorithms to solve combinatorial problems without resorting to tree search. Levin tree search (Orseau et al., 2018) uses a policy trained by an RL algorithm in tree search, and Orseau & Lelis (2021) subsequently optimize the policy in bootstrap learning. Generalized Search for Planning (GSP) (Aichmüller et al., 2026) learns the cost to goal using modified Q-learning.

The fundamental obstacle is a mismatch in the problem formulation of RL methods and tree search, which forces every earlier method to optimize proxy values instead of the decisions GBFS actually makes. We therefore revisit the formalizations proposed by Bhardwaj et al. (2017); Mattick & Mutschler (2023) (called in this paper Tree-MDP) for robotic motion planning with uncertainty and optimization of branch-and-bound, and adapt it to symbolic planning. Tree-MDP recasts the forward search as a Markov Decision Process (MDP), where a state is a search tree *configuration* (including

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

open and closed lists), and an action is the selection of a state from the open list for expansion, which enables direct optimization of GBFS choices.

While value-based RL has been applied to heuristic learning (e.g., Q-learning Bernhard et al., 2018; Aichmüller et al., 2026), we pursue policy-gradient methods. To validate this, we evaluate Proximal Policy Optimization (PPO) (Schulman et al., 2017), a widely adopted off-the-shelf policy-gradient algorithm, demonstrating that Tree-MDP is compatible with standard RL tooling. Motivated by theoretical results that optimal policies can be simpler to represent than value functions Feng & Zhong (2024), and evidence that value estimation needs more information than policy learning Raileanu & Fergus (2021) we propose *policy optimization* for learning search heuristics. This choice is apt because the set of available actions (open list nodes) changes continuously as the search progresses. Unlike RL, planning assumes complete knowledge of transition dynamics; the proposed policy-gradient optimization is efficient without incurring sample inefficiency that plagues RL.

PPO on the Tree-MDP still needs a reward and a discount factor, which can be difficult to set (Shuhailo et al., 2026). The Tree-MDP suggests an alternative, once the best trajectory (in the current search tree) is available, imitating it recovers the ranking losses of Chrestien et al. (2023). Imitating the best trajectory is, however, off-policy, as it ignores the configurations actually visited by the search. The proposed policy-optimization loss ($\mathrm{L_{po}}$) fixes this.

To summarize, the contributions of this paper are

1. We revisit the Tree-MDP formulation of Bhardwaj et al. (2017) to establish a link between learning heuristic functions for symbolic planning and policy optimization in reinforcement learning. Importantly, the resulting heuristic is fully compatible with existing solvers.
2. We show that Tree-MDP lets off-the-shelf PPO optimize GBFS heuristics without expert solutions, outperforming both PPO on Linear-MDP and value-based RL with best-first search (GSP) under the same architecture.
3. We show that imitating the best trajectory in the Tree-MDP recovers the ranking losses of Chrestien et al. (2023) explaining why they outperform cost-to-goal estimation, as they directly optimize the expansion decision.
4. We derive a new loss, $\mathrm{L_{po}}$ evaluating the policy on the open lists GBFS encounters rather than on the best trajectory alone, which implicitly emphasizes costly erroneous decisions.

Terminology. Throughout this paper, we carefully distinguish between *planning states* s (individual configurations of the planning domain) and *search tree configurations* T (partial search trees with their open and closed lists of nodes). This distinction is crucial for clarity: standard MDP notation uses "state" to refer to domain configurations, but we use "configuration" for search tree states to avoid confusion. When the paper refers to an MDP state in the Tree-MDP context, it is always a configuration T, not a planning state s.

2 Problem Definition and Background

Planning Problem. This paper assumes a planning problem defined as a tuple $(\mathcal{S}, \mathcal{A}, \delta, s_0, o)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\delta\colon \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a deterministic transition function, $s_0 \in \mathcal{S}$ is the initial state, and $o\colon \mathcal{S} \to \mathbb{R}$ is an objective function quantifying state quality. The goal is to find a sequence of actions $(a_0, a_1, \ldots, a_{n-1})$ such that the resulting path $(s_0, s_1, \ldots, s_n)$ with $s_{i+1} = \delta(s_i, a_i)$ reaches a state s_n that maximizes $o(s_n)$. We assume that o depends only on the state itself (not on the path to the state), and that o attains its maximum over the set $\mathcal{R}(s_0) \subseteq \mathcal{S}$ of states reachable from s_0, which holds whenever $\mathcal{R}(s_0)$ is finite. This formulation generalizes classical planning problems, where goal states are defined as a set $G \subseteq \mathcal{S}$. Classical goal-based planning is recovered with $o(s) = 1$ if $s \in G$ and $o(s) = 0$ otherwise. To simplify notation, $\mathrm{ch}(s)$ denotes all children of s; that is, $\mathrm{ch}(s) = \{\, s' \mid (\exists a \in \mathcal{A})(s' = \delta(s, a)) \,\}$.

Formalizing Planning as MDP (Linear-MDP). The planning problem can be viewed as a standard MDP, hereafter called *Linear-MDP* to distinguish it from the Tree-MDP introduced below in Sect. 3. The Linear-MDP is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \delta, R)$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\delta\colon \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the deterministic transition function, and $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. To align planning with the MDP, the reward function is defined as in Chen & Tian (2019)

$$R(s, a) = o(\delta(s, a)) - o(s). \tag{1}$$

This formulation ensures that the cumulative reward along a path $(s_0, s_1, \ldots, s_n)$ equals $o(s_n) - o(s_0)$, depending only on the start and end states, not the path taken. The objective in the Linear-MDP is to maximize the discounted cumulative reward $\sum_{t=0}^{n-1} \gamma^t R(s_t, a_t)$ where γ is a discount factor (typically $\gamma = 1$ for episodic planning tasks).

The Linear-MDP formulation assumes that trajectories terminate when a goal state is reached (classical planning) or when a best reachable state $s^\star \in \arg\max_{s \in \mathcal{R}(s_0)} o(s)$ is found. Under this assumption and $\gamma = 1$, the cumulative reward equals $o(s^\star) - o(s_0)$ and correctly reflects solution quality. However, in tree search, the best state discovered may be a child of some intermediate state rather than the final state s_n. The cumulative reward $o(s_n) - o(s_0)$ reflects only the quality of the final state, with no way to account for the best state found earlier during the search—a mismatch between the MDP formulation, whose actions are domain operators on a single current state, and tree search, which selects expansions from an open list of frontier nodes. Section 3 addresses this fundamental mismatch by reformulating the problem: instead of states being domain configurations, states become search tree *configurations* (open and closed lists), actions are selections from the open list, and rewards measure improvements in the best state *found so far*—directly aligning MDP rewards with search progress. In the Linear-MDP, policies are parameterized by heuristic functions that depend only on individual planning states

$$\pi(a = s' | s) = \frac{e^{-h(s')}}{\sum_{s'' \in \mathrm{ch}(s)} e^{-h(s'')}}, \tag{2}$$

where selecting an action corresponds to transitioning to a successor state $s' \in \mathrm{ch}(s)$, and $h : \mathcal{S} \to \mathbb{R}$ is a learned heuristic function. The heuristic function $h(s)$ estimates which state in the open list will lead to the highest o value, guiding the search accordingly.

Greedy Best-First Search (GBFS). GBFS is a tree search algorithm that finds a path from the initial state s_0 to a goal state by greedily expanding the most promising state (according to the heuristic h) from the *open list* of states not yet expanded. Unlike A* with admissible heuristics, GBFS offers no optimality guarantees, prioritizing *search efficiency*: finding solutions quickly by expanding few states. Classical GBFS (Alg. 1 in Sect. A.1) checks goal membership ($s \in G$); our formulation generalizes this to maximizing the objective $o(s)$, and search terminates when a goal state is found or the time limit is exhausted. For training with RL, we replace the deterministic selection of the best state with sampling from a categorical distribution with logits $-h(s)$, which enables exploration while preserving the GBFS structure and is required by the policy-gradient methods below.

3 TREE-MDP

This section formalizes GBFS as an MDP problem, called Tree-MDP. The motivation is that applying RL algorithms to learn heuristics for planning ignores a fundamental difference: standard RL algorithms optimize policies for an agent acting in the original problem space, where states correspond to planning states and actions to domain operators. But tree search algorithms operate in a different space with search tree configurations and actions corresponding to expansion of states. It is therefore difficult to use a policy optimized for Linear-MDP in tree search (Orseau et al., 2018). Tree-MDP reformulates tree search itself as an MDP, addressing the Linear-MDP mismatch by operating directly on search tree configurations. When discussing Linear-MDP, we use standard notation $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \delta, R)$ with states s and actions a. When discussing Tree-MDP, we use notation $\tilde{\mathcal{M}} = (\tilde{\mathcal{S}}, \tilde{\mathcal{A}}, \tilde{\delta}, \tilde{R})$ with configurations T.

A *Tree-MDP configuration* represents a snapshot of the search tree built by GBFS at some point during execution. It consists of three essential components: *closed list* containing states that have been expanded, *open list* containing candidate states for expansion, and *search tree* with parent-child relationships connecting these states. Formally:

Definition 1 (Tree-MDP configuration)**.** *A configuration $T = (\mathcal{T}, \mathcal{O}, \mathcal{C})$ of a Tree-MDP consists of:*

- *The* search tree $\mathcal{T} = (V, E)$ *contains all states in the closed and open lists, stored as a directed acyclic graph (DAG) with parent pointers.*
- *The* open list $\mathcal{O} \subseteq \mathcal{S}$ *of frontier states not yet expanded.*
- *The* closed list $\mathcal{C} \subseteq \mathcal{S}$ *of states already expanded.*

Definition 2 (Tree-MDP)**.** *For a planning problem specified by $(\mathcal{S}, \mathcal{A}, \delta, s_0, o)$, the corresponding Tree-MDP is a tuple $\tilde{\mathcal{M}} = (\tilde{\mathcal{S}}, \tilde{\mathcal{A}}, \tilde{\delta}, \tilde{R})$ defined as:*

- *$\tilde{\mathcal{S}}$: the set of all configurations reachable from the initial configuration T_0 consisting of search tree $\mathcal{T}_0 = (\{s_0\}, \emptyset)$, open list $\mathcal{O}_0 = \{s_0\}$, and empty closed list $\mathcal{C}_0 = \emptyset$.*
- *$\tilde{\mathcal{A}}(T)$: the set of available actions at configuration T is the open list, $\tilde{\mathcal{A}}(T) = \mathcal{O}(T)$, since each action corresponds to selecting a state $s \in \mathcal{O}(T)$ for expansion.*
- *Transition function $\tilde{\delta}(T, s)$ producing the next configuration $T' = (\mathcal{T}', \mathcal{O}', \mathcal{C}')$ from configuration T and action s via the following procedure:*
  - *s is added to closed list, $\mathcal{C}' = \mathcal{C} \cup \{s\}$,*
  - *s is removed and children of s not in $\mathcal{C}'$ are added to open list, $\mathcal{O}' = (\mathcal{O} \cup \mathrm{ch}(s)) \setminus \mathcal{C}'$,*
  - *the search tree $\mathcal{T} = (V, E)$ becomes $\mathcal{T}' = (V \cup \mathrm{ch}(s), E \cup \{\, (s, s') \mid s' \in \mathrm{ch}(s) \setminus \mathcal{C}' \,\})$.*
- *The episode terminates when either $\mathcal{O} = \emptyset$, or when a best reachable state $s^\star$ has been added to $\mathcal{C}$ (goal is found but applicable only if maximum is known).*
- *Reward function $\tilde{R}(T, s) = \max_{s' \in V(\mathcal{T}')} o(s') - \max_{s' \in V(\mathcal{T})} o(s')$, where $T' = \tilde{\delta}(T, s)$ and $\mathcal{T}'$ / $\mathcal{T}$ is the search tree of configuration T' / T and $V(\cdot)$ is the vertex set. The reward measures improvement in the best objective value in the search tree after expanding s.*

Tree-MDP satisfies the Markov property: the next configuration $\tilde{\delta}(T, s)$ depends only on the current configuration T and the chosen action s, not on the history of how T was reached. This holds because the configuration $T = (\mathcal{T}, \mathcal{O}, \mathcal{C})$ fully encodes all necessary information: the tree $\mathcal{T}$ records which states have been explored, and the open list and closed list encode the current search frontier. All future transitions depend only on these components.

Recall that the Linear-MDP introduced in Sect. 2 has a fundamental limitation of the step-wise reward structure not inherently incentivizing the policy to reach a best reachable state $s^\star$; in experiments, we clip trajectories (see footnote 1). The Tree-MDP does not have this problem as the reward measures improvement in the best objective value discovered so far. The following proposition formalizes this.

Proposition 1 (Optimality of Tree-MDP solution)**.** *Let $(\mathcal{S}, \mathcal{A}, \delta, s_0, o)$ be a planning problem whose objective o depends only on the state and attains its maximum $o^\star = \max_{s \in \mathcal{R}(s_0)} o(s)$ over the set $\mathcal{R}(s_0) \subseteq \mathcal{S}$ of states reachable from s_0. Consider the corresponding Tree-MDP with $\gamma = 1$ (undiscounted episodic setting). Then a sequence of actions (expansions) $(s_0, s_1, \ldots, s_n)$ of Tree-MDP maximizes the cumulative reward over all sequences of actions of any length if and only if the search tree $\mathcal{T}_{n+1}$ contains a state $s^\star$ with $o(s^\star) = o^\star$, where $T_{i+1} = \tilde{\delta}(T_i, s_i)$ for $i \in \{0, \ldots, n\}$, and $\mathcal{T}_i$ is the search tree of configuration T_i.*

The proof is given in Sect. A.2. Proposition 1 shows that maximizing the reward of a Tree-MDP with $\gamma = 1$ implies finding solutions, but with $\gamma = 1$ there is no incentive to minimize the number of expansions (Shuhailo et al., 2026). In RL, this problem is usually resolved by setting a discount factor $\gamma < 1$, as has been recently discussed in the context of LLMs in Ayoub et al. (2025). Below in Sect. 4 we propose alternative solution inspired by Chrestien et al. (2023) avoiding the need of γ.

Policy parameterization. Tree-MDP provides a principled optimization objective, implying that policies should be functions of the entire configuration $\pi(\tilde{a}|T)$. However, this makes the policy (i) incompatible with standard GBFS implementations, which assume heuristics depend only on individual planning states $s \in \mathcal{S}$ and (ii) computationally expensive to represent and optimize.

This paper therefore restricts to a practical approximation: policies parameterized by heuristic functions, which depend only on individual planning states as

$$\pi(\tilde{a} = s|T) = \frac{e^{-h(s)}}{\sum_{s' \in \mathcal{O}(T)} e^{-h(s')}}, \tag{3}$$

where action $\tilde{a}$ corresponds to expanding state $s \in \mathcal{O}(T)$, and $h : \mathcal{S} \to \mathbb{R}$ is a learned heuristic function. In deterministic GBFS, the algorithm naturally expands the state with minimal heuristic value, which corresponds to the state with maximum probability under this softmax distribution.

Thus, we can deploy an optimized h directly in GBFS solvers without modification. We emphasize that restricting to softmax-parameterized policies is a deliberate approximation that trades expressiveness for computational efficiency. The full Tree-MDP allows policies that condition arbitrarily on the configuration. This general option is not pursued further, since this paper aims to study the advantage of the Tree-MDP formulation using the same functional form of h.

4 Optimizing Policies without Explicit Rewards

Both Linear-MDP and Tree-MDP policies can be optimized using RL algorithms, which we demonstrate in Sect. 6 using the PPO algorithm. We also consider an alternative approach that does not require an explicit reward function; instead, an absolute ordering of trajectories suffices. For Linear-MDP, the core idea is to sample multiple trajectories and improve the policy to prefer the best one.

Formally, assume we sample m trajectories $\{\xi_i\}_{i=1}^m$ from Linear-MDP and let $\xi^\star = (s_0^\star, s_1^\star, \ldots, s_n^\star)$ be the best trajectory according to a chosen ordering criterion. The trajectory $\xi^\star$ defines an empirical policy on the visited states: $\pi^\star(a|s) = 1$ if $s = s_i^\star$ for some $i \in \{0, \ldots, n-1\}$ and $\delta(s_i^\star, a) = s_{i+1}^\star$ for exactly one action a, and zero otherwise. The support of $\pi^\star$ equals the states in trajectory $\xi^\star$, which is sufficient for policy optimization. The current policy π_θ can be improved to match $\pi^\star$ by minimizing the KL divergence: $\mathrm{KL}(\pi^\star \| \pi_\theta)$. Sampling multiple trajectories can be computationally expensive, and therefore the method does not seem practical for Linear-MDP, although multiple rollouts are currently popular within group-relative policy optimization (Shao et al., 2024). But in Tree-MDP, the best trajectory $\xi^\star$ is cheap to obtain, because the shortest path to the best solution node can be directly extracted.

$\mathrm{L}_{\mathbf{gbfs}}^{\mathbf{softmax}}$ loss. Let $\xi^\star = (s_0^\star, s_1^\star, \ldots, s_n^\star)$ be the shortest path to the best state. Since $\xi^\star$ can be viewed as a sequence of actions in Tree-MDP, the corresponding configurations are $(T_0^\star, T_1^\star, \ldots, T_n^\star)$, where $T_{i+1}^\star = \tilde{\delta}(T_i^\star, s_i^\star)$. In this case, minimizing the KL divergence between the empirical distribution $\pi^\star$ and π_θ of Tree-MDP derived from $\xi^\star$ equals

$$\mathrm{KL}(\pi^\star \| \pi_\theta) = \sum_{i=1}^{n} \sum_{s \in \mathcal{O}(T_i^\star)} \pi^\star(s|T_i^\star) \log \frac{\pi^\star(s|T_i^\star)}{\pi_\theta(s|T_i^\star)} \tag{4}$$

where $\mathcal{O}(T_i^\star)$ is the open list of configuration $T_i^\star$ after expanding states $(s_0^\star, s_1^\star, \ldots, s_{i-1}^\star)$. Recognizing that the empirical policy $\pi^\star$ is zero everywhere except when expanding the state on the optimal trajectory, the above can be rewritten as

$$-\sum_{i=1}^{n} \log(\pi_\theta(s_i^\star|T_i^\star)) = -\sum_{i=1}^{n} \log \frac{e^{-h(s_i^\star)}}{\sum_{s' \in \mathcal{O}(T_i^\star)} e^{-h(s')}}, \tag{5}$$

where we have substituted the policy parameterization compatible with GBFS from Eq. (3). Equation (5) recovers the $\mathrm{L}_{\mathrm{gbfs}}$ loss with logsoftmax, as defined in Eq. (9).

Policy-optimizing loss. The above loss function in Eq. (5) may be off-policy because the best trace $\xi^\star$ may differ significantly from the sequence of states expanded by GBFS. Therefore, if the current policy π_θ is far from $\pi^\star$ and consistently makes mistakes (expanding many states outside $\xi^\star$), no weight can emphasize the large inefficiency. We therefore propose to define a loss function, which considers every configuration of Tree-MDP observed during the search.

Let $(\bar{s}_0, \bar{s}_1, \ldots, \bar{s}_{\bar{n}})$ denote the sequence of states expanded by GBFS, creating a sequence of Tree-MDP configurations $(\bar{T}_1, \bar{T}_2, \ldots, \bar{T}_{\bar{n}+1})$ with corresponding open lists. Furthermore, let $\xi^\star$ again be the shortest trajectory to the best state corresponding to expanding states $(s_0^\star, s_1^\star, \ldots, s_n^\star)$. The best known action at Tree-MDP configuration $\bar{T}_i$ is to expand the state from the open list closest to $s_n^\star$,

$$\bar{s}_{\bar{T}_i}^\star = \operatorname*{arg\,max}_{s_j^\star \in \mathcal{O}(\bar{T}_i)} j, \tag{6}$$

which defines the corresponding empirical policy as long as at least one state from $\xi^\star$ remains in the open list. The loss function for minimizing the KL divergence can be simplified as above to

$$-\sum_{i=1}^{\bar{n}} \log(\pi_\theta(\bar{s}_{\bar{T}_i}^\star|\bar{T}_i)) = -\sum_{i=1}^{\bar{n}} \log \frac{e^{-h(\bar{s}_{\bar{T}_i}^\star)}}{\sum_{s' \in \mathcal{O}(\bar{T}_i)} e^{-h(s')}}. \tag{7}$$

The loss defined in Eq. (7) is called *policy-optimizing* ($\mathrm{L_{po}}$). An important feature of $\mathrm{L_{po}}$, absent in Eq. (5), is that states on optimal plan not expanded multiple times implicitly receives higher weight, because they appear in multiple open lists. This weighting is implicit, unlike in Policy-guided Heuristic Search (PHS) (Orseau & Lelis, 2021), where weights have an explicit formula.

Relation of $\mathrm{L}^{\mathbf{softmax}}_{\mathbf{gbfs}}$ loss to Ranking Losses. The softmax policy parameterization and the $\mathrm{L}^{\mathrm{softmax}}_{\mathrm{gbfs}}$ loss defined in Eq. (9) reveal a connection between Tree-MDP and recent ranking-based loss functions. The ranking loss $\mathrm{L_{gbfs}}$ proposed in Chrestien et al. (2023); Hao et al. (2024) for imitation learning is defined as

$$\mathrm{L_{gbfs}}(h, \xi) = \sum_{i=1}^{n} \sum_{s' \in \mathcal{O}(T_i^\star) \setminus \{s_i^\star\}} \ell\big(h(s_i^\star) - h(s')\big), \tag{8}$$

where ℓ is a convex surrogate of a step function (e.g., logistic loss). The goal of inner sum $\sum_{s' \in \mathcal{O}(T_i^\star) \setminus \{s_i^\star\}} \ell\big(h(s_i^\star) - h(s')\big)$ in Eq. (8) is to force the heuristic value of state $s_i^\star$ to have a smaller value than other states in the open list $\mathcal{O}(T_i^\star) \setminus \{s_i^\star\}$. This last property can be similarly enforced by replacing the pairwise ranking with a log-softmax function as

$$\mathrm{L_{softmax}}(h, \xi) = -\sum_{i=1}^{n} \log \frac{e^{-h(s_i^\star)}}{\sum_{s' \in \mathcal{O}(T_i^\star)} e^{-h(s')}}, \tag{9}$$

which recovers Eq. (5). Due to this similarity, the loss in Eq. (5) is called the $\mathrm{L}^{\mathrm{softmax}}_{\mathrm{gbfs}}$ loss.

According to Chrestien et al. (2023); Hao et al. (2024); Horčík et al. (2025), heuristics optimized via ranking-based losses almost always outperform those estimating cost-to-goal. This empirical observation admits a theoretical explanation in the Tree-MDP framework: ranking losses directly optimize the expansion policy, whereas cost-to-goal estimation learns state values, an indirect proxy for search efficiency.

5 Related Work

Learning Heuristics with RL. Orseau et al. (2018) propose a tree-search algorithm that uses policy networks trained on the Linear-MDP by A3C. A subsequent work (Orseau & Lelis, 2021) extends to the bootstrap learning protocol (Arfaee et al., 2011), removing the need for pre-trained policies. Our approach differs in: (i) formulation of the policy directly on the open list of Tree-MDP rather than the Linear-MDP, (ii) our framework enables standard RL algorithms to optimize heuristics without modification, and (iii) the weighting emphasis on errors is implicit rather than explicit. Actor-critic methods are used in Ståhlberg et al. (2023) to learn a policy over successor states, which corresponds to the Linear-MDP formulation. Chervov et al. (2025) learns a Rubik's Cube heuristic by generating states via random walks from the solved state and training a neural network to predict their distance to the goal, which then guides search. This approach is difficult to transfer to symbolic planning, as the goal is usually not defined as a single fully known state.

Heuristic Learning Objectives and Loss Functions. Classical approaches to optimization of heuristic functions learn the cost-to-goal $h^*(s)$, e.g., via Q-learning (Bernhard et al., 2018; Aichmüller et al., 2026). For GBFS, however, the relative ranking of states in the open list matters more than absolute values, and ranking losses have been shown to outperform cost-to-goal estimation (Chrestien et al., 2023; Horčík et al., 2025; Hao et al., 2024). The Tree-MDP perspective explains this through the relation to policy optimization.

Representation Learning for Symbolic Planning. The design of state representations and network architectures for heuristics is an active area (Ståhlberg et al., 2022; Horcík & Sír, 2024; Toyer et al., 2020; Chen et al., 2024; Ferber et al., 2020), with contributions including the use of graph neural networks for relational reasoning, attention mechanisms for learning object interactions, and specialized encodings for symbolic planning. These works are orthogonal to this paper, which focuses on *what* to optimize (policy for Tree-MDP) rather than *how* to represent states. Our experiments build on this research by using the ObjectBinary architecture with parameters from Horčík et al. (2025), which has shown strong performance on the IPC 2023 domains (Taitler et al., 2024).

Tree-Based Learning and Search. Tree-MDP is similar to the formulation of Bhardwaj et al. (2017) for robotic motion planning in initially unknown environments imitating optimal plans

provided by an oracle. Here, the formalism is adapted to symbolic planning and used to bootstrap learning without an oracle.

The work od Scavuzzo et al. (2022) share Tree-MDP name, but the formulations to optimize Mixed Integer Linear Programming branching rules are different. This work differs by deriving a new loss function $\mathrm{L_{po}}$, relating it to ranking losses, and applying it to symbolic planning. Additionally, our actions select states from the GBFS open list, the reward tracks the best objective in the tree (Prop. 1), and the resulting state-level heuristic runs in unmodified GBFS solvers.

TreeQN (Farquhar et al., 2018) builds small trees to improve estimates of Q-value estimation during search rather than to model the search process. MCTSNets (Guez et al., 2018) replace all MCTS components with trainable networks, whereas we study GBFS and its connection to RL. AlphaZero (Silver et al., 2018) optimizes MCTS for games rather than classical planning.

6 Experiments

The experiments are designed to (i) show the advantage of the Tree-MDP formulation by comparing heuristics optimized by the same algorithm (PPO) in Linear-MDP and Tree-MDP formulations; (ii) compare the proposed $\mathrm{L_{po}}$ to the L^* ranking loss and to its softmax variant; and (iii) compare the learned heuristics to prior art methods and to domain-independent planners.

Compared Algorithms. We experimentally compare the following loss functions and methods. $\mathrm{L_{gbfs}}$ (Chrestien et al., 2023), defined in Eq. (8), was originally proposed for imitation learning but can be adapted to bootstrap learning by training on the shortest trajectory to the best node in the search tree. As ranking losses have been shown to outperform cost-to-goal estimation across a range of methods (Chrestien et al., 2023; Horčík et al., 2025), we treat $\mathrm{L_{gbfs}}$ as the state of the art. To observe the effect of replacing inequalities in $\mathrm{L_{gbfs}}$ with the `softmax` function, we tested the variant as defined in Eq. (9), further denoted as $\mathrm{L^{softmax}_{gbfs}}$. As explained in Sect. 4, $\mathrm{L^{softmax}_{gbfs}}$ can be viewed as optimizing a policy that selects which states to expand from an open list. The proposed $\mathrm{L_{po}}$, which applies the softmax loss to the open lists observed during the search, was implemented as defined in Eq. (7). The comparison also includes recently proposed **Generalized Search for Planning** (GSP) (Aichmüller et al., 2026), which learns a heuristic by Q-learning on Bellman targets supplied by its own weighted A^* searches. For a fair comparison, we replaced the original GNN architecture with ObjectBinary (Horcík & Sír, 2024) for all experiments. To compare Tree-MDP with Linear-MDP formulations using the same RL algorithm, we used the popular Proximal Policy Optimization (PPO) (Schulman et al., 2017) algorithm to optimize the policy defined by Eq. (3), making it compatible with GBFS, and called it **Tree-PPO**. The combination of Linear-MDP with PPO, called **Linear-PPO**, directly applies RL methods to combinatorial problems (e.g., Ståhlberg et al., 2023; Chen & Tian, 2019) and optimizes policies with policy gradient methods on Linear-MDP. The policy was parametrized by Eq. (2) and uses the same ObjectBinary encoding as other methods. **Levin Tree Search** (denoted as LevinTS), proposed in Orseau et al. (2018), is not trained separately: it reuses the policies trained by Linear-PPO and only changes the tree search algorithm, which is no longer GBFS. **Policy-guided Heuristic Search** PHS (Orseau & Lelis, 2021) is the closest prior art to this work. It follows Orseau et al. (2018), but optimizes the policy in a bootstrap model. The heuristic uses an estimate of cost-to-goal, which means that the training set can be created only from solved problem instances, not from partially solved instances where cost-to-goal is unknown. PHS uses two neural networks: one for policy and one to estimate cost-to-goal. The policy network is parametrized again by Eq. (2), that is, normalized over a state's children. **GBFS with FastForward heuristic** represents vanilla GBFS with domain-independent heuristics FastForward (FF) heuristic (Hoffmann & Nebel, 2001). The heuristic is in the `SymbolicPlanners.jl` package (Zhi-Xuan, 2022) used for our experiments. **LAMA** planner (Richter & Westphal, 2010) represents modern performance domain independent planner. LAMA planner is the only planner in the comparison which is not implemented on top of `PDDL.jl` and `SymbolicPlanners.jl` libraries. We do not compare with heuristics learned from solutions provided by a separate planner, such as Chen et al. (2024); Ståhlberg et al. (2022); Horčík et al. (2025); Chrestien et al. (2023), which are trained in the imitation learning scenario. Their training data does not depend on the learned heuristic, whereas this paper studies learning from the heuristic's own searches, where no solutions are available.

	L_{gbfs}	$L^{softmax}_{gbfs}$	L_{po}	Tree-PPO	LPPO	PHS	LevinTS	GSP	FF	LAMA
Blocks	73.0 ± 6.0	75.0 ± 2.0	$\mathbf{83.0 \pm 5.0}$	33.0 ± 6.0	23.0 ± 3.0	36.0 ± 2.0	25.0 ± 2.0	4.0 ± 0.0	24.0	38.0
Childsnack	$\mathbf{51.0 \pm 5.0}$	$\mathbf{51.0 \pm 6.0}$	45.0 ± 2.0	13.0 ± 3.0	4.0 ± 7.0	13.0 ± 8.0	6.0 ± 0.0	5.6 ± 0.0	0.0	33.0
Ferry	$\mathbf{68.0 \pm 2.0}$	67.0 ± 0.0	67.0 ± 1.0	60.0 ± 2.0	1.0 ± 1.0	53.0 ± 6.0	8.0 ± 0.0	51.0 ± 0.0	54.0	67.0
Floortile	6.0 ± 3.0	3.0 ± 1.0	$\mathbf{32.0 \pm 1.0}$	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.0 ± 1.0	0.0 ± 0.0	1.0	12.0
Miconic	$\mathbf{74.0 \pm 1.0}$	73.0 ± 0.0	73.0 ± 0.0	66.0 ± 1.0	1.0 ± 1.0	56.0 ± 0.0	23.0 ± 0.0	28.0 ± 2.0	59.0	<u>89.0</u>
Rovers	$\mathbf{39.0 \pm 1.0}$	38.0 ± 1.0	38.0 ± 2.0	34.0 ± 2.0	30.0 ± 1.0	32.0 ± 2.0	9.0 ± 1.0	4.0 ± 0.0	26.0	<u>51.0</u>
Satellite	60.0 ± 5.0	60.0 ± 2.0	$\mathbf{61.0 \pm 4.0}$	46.0 ± 5.0	40.0 ± 5.0	20.0 ± 17.0	12.0 ± 0.0	14.0 ± 0.0	41.0	<u>73.0</u>
Sokoban	31.0 ± 4.0	33.0 ± 4.0	$\mathbf{34.0 \pm 1.0}$	$\mathbf{34.0 \pm 1.0}$	0.0 ± 0.0	19.0 ± 1.0	19.0 ± 1.0	19.0 ± 1.0	18.0	<u>38.0</u>
Spanner	72.0 ± 2.0	72.0 ± 0.0	$\mathbf{73.0 \pm 1.0}$	64.0 ± 9.0	69.0 ± 3.0	56.0 ± 9.0	33.0 ± 0.0	55.0 ± 19.0	33.0	33.0
Transport	52.0 ± 3.0	49.0 ± 2.0	$\mathbf{61.0 \pm 3.0}$	41.0 ± 1.0	45.0 ± 2.0	7.0 ± 0.0	16.0 ± 9.0	7.0 ± 0.0	29.0	58.0
Average	53.0 ± 0.0	52.0 ± 0.0	$\mathbf{57.0 \pm 1.0}$	39.0 ± 0.0	21.0 ± 1.0	29.0 ± 3.0	15.0 ± 1.0	18.8 ± 0.0	29.0	49.0

Table 1: Coverage (%) on the test instances of each domain, reported as mean $\pm$ standard deviation over three runs, in the IPC 2023 dataset. FF and LAMA are deterministic, so we do not report a standard deviation for them. The best mean coverage of a learned heuristic in each row is shown in bold, and a planner result is underlined when it matches or exceeds it.

Experimental settings. Unless stated otherwise, all policies are parametrized as defined in Eq. (3), which (i) allow compatibility with existing GBFS solvers; (ii) enable comparison to prior work under the same conditions, and (iii) elegantly handle variable number of actions. The heuristic function $h_\theta(s)$ within Eq. (3) and value functions in PPO, PHS, and GSP are implemented as a graph neural network with ObjectBinary encoding and features on edges described in Horčík et al. (2025), which was selected because it is lightweight and, according to the same study, performs among the best on IPC 2023 problems. The networks consist of: one graph message-passing layer with hidden dimension 32, followed by `sum` and `max` pooling functions, then a two-layer fully-connected network with hidden dimension 32 and output dimension 1 with ReLU activations.

GBFS follows Sect. 2, with the modification that during training and testing, the deterministic selection of the best state is replaced with stochastic selection from the policy distribution (with temperature $\tau = 1$) to promote exploration. Linear-PPO samples the next state from the learned policy during both training and testing until solving the problem or the time limit is exhausted.[1]

Objective function (o) is equal to the size of the intersection of atoms in a state with the atoms in the goal specification. This means that maximizing the objective function solves the problem. We use this objective to provide a denser training signal. We find the best state in unsolved problem instances by maximizing o while minimizing distance to the root (each action has unit cost). The reward in Linear-PPO is defined as in sections 2 and 3, and in Tree-PPO as in Def. 2.

Evaluation and metrics. All methods were compared on domains from the IPC 2023 learning track (Taitler et al., 2024) with a default train/test split. This means that the bootstrap protocol is first run on easy training instances, then the reported coverage is computed from 90 testing instances. We measure *coverage* as the percentage of test instances solved within 30 s. We report the mean and standard deviation from three runs (seeds 1, 2, and 3). Since all methods have hyperparameters (see Sect. A.3), we select the best hyperparameters according to the IPC plan quality metric on the training set. **More details about the training protocol are provided in Sect. A.3.**

Experimental Results. Table 1 reports coverage of all compared methods on all domains. The methods are ordered from the losses trained on the search tree to the RL baselines and PHS (for a coverage split by difficulty of problem instances, see Tbl. A.2). The experimental results show that coverage of Tree-PPO is much better than that of Linear-PPO, which supports the view that decisions should depend on the evolving search tree rather than individual states. On average, all variants of PPO are worse than ranking losses, which we attribute to the fact that extracting the best trajectory from the search tree provides a cleaner learning signal than policy optimization of actor-critic methods like PPO.

The close average performance of L_{gbfs} and $L^{softmax}_{gbfs}$ (53.0% and 52.0% test coverage) supports the link between ranking losses and policy optimization. $L^{softmax}_{gbfs}$ adds a probabilistic interpretation but does not materially change the aggregate result, although its effect is domain-dependent.

[1] In Linear-PPO, each trajectory is clipped so that its last state is the first maximizer of o along it. Thus, for $(s_0, s_1, \ldots, s_n)$, $o(s_n) = \max_{i \in \{0,1,\ldots,n\}} o(s_i)$ and $(\forall i \in \{0, 1, \ldots, n-1\})(o(s_i) < o(s_n))$.

The proposed L_{po} has the best average performance. It delivers the best performance on six of ten domains, lagging by one percent on the other three. The single large degradation of 6% is on Childsnack. Coverage analysis on the training set across epochs (Fig. 1) suggests this may be due to the chosen bootstrap protocol, which greedily optimizes the heuristic after each solved problem instance and may forget previous solutions.

The proposed L_{po} achieves comparably higher coverage on the Floortile domain. Investigation of the search tree revealed that this domain contains many dead-ends, and L_{po} can effectively learn to block them thanks to its weighting of consistent errors, a feature missing in L_{gbfs} and $L_{gbfs}^{softmax}$.

The average coverage of L_{gbfs}, $L_{gbfs}^{softmax}$, and the proposed L_{po} is within 1% across five domains. We believe this is caused by the fact that as the heuristic improves and makes fewer errors, the states in the minibatch of all three losses become similar, therefore removing differences between them. Figure 1 shows the fraction of the plan length and the number of expanded states. If this fraction is one, then the states in minibatches of these loss functions coincide.

On average, PHS and LevinTS perform worse than the ranking losses, with test coverage of 29.0% and 15.0%. We attribute this to the fact that ranking losses directly optimize the efficiency of GBFS. We note that Chrestien et al. (2023) reached similar conclusions when comparing L_{gbfs} and PHS. GSP (Aichmüller et al., 2026) performs worse than the L_{po}, L_{gbfs}, $L_{gbfs}^{softmax}$, Tree-PPO, and even to Linear-PPO. Since all methods share the same network and budget, we believe the culprit is the training objective, as GSP relies on deep Q-learning and currently seems to underperform policy-based methods (Raileanu & Fergus, 2021; Furuta et al., 2021; Feng & Zhong, 2024; Shuhailo et al., 2025).

The domain-independent LAMA planner achieves the highest coverage in four of the ten domains and ties with L_{po} in Ferry. Its average coverage of 49.0% is below that of L_{po} 57.0%, though the methods tie on the average IPC-style score at 49.0% Tbl. A.5. LAMA expands far more states than L_{po}, Tbl. A.4, which is likely caused by the fact that LAMA does not stop when finding the first solution, but uses the remaining time budget to improve. We also point out that LAMA uses a separate implementation and requires no training.

7 Conclusion

This paper bridged the gap between optimizing heuristic functions for GBFS and RL. The Tree-MDP formalism combines search and problem dynamics, enabling principled learning of heuristic functions via reinforcement learning methods. The experiments with PPO clearly confirmed the advantages of the proposed formalism, improving average coverage by 18% over the naive approach.

Policy learning in the Tree-MDP formalism connects recently proposed ranking losses with policy optimization, showing both variants achieve similar performance. This connection facilitated the design of a new loss function, L_{po}, which combines strengths from both planning and reinforcement learning and allows optimizing the policy without explicitly designing a reward, as it is sufficient to define a total order of plans. The proposed L_{po} achieves the highest average coverage and IPC plan quality score among the learned heuristics, with the largest gains in Floortile and Transport and smaller gains in several other domains.

We hope this formalism will facilitate the transfer of ideas between planning and reinforcement learning and help identify differences, as we have shown that using the search tree structure can lead to better algorithms. In the future, we would like to investigate the relationship between Tree-MDP and Monte Carlo Tree Search, as well as methods that compute a policy from an entire search tree. Furthermore, we would like to make the training computationally cheaper by investigating whether all states in the search tree are needed in the minibatch.

Limitations. Since the goal was to compare all methods under the same settings and scenarios, all methods used the same encoding of states and architecture of the neural network, which might cause differences in the metrics reported in the original works. We also deliberately avoided comparison to imitation learning, since the focus was on optimization in bootstrap learning. The experiments were carried out on a dataset from the IPC 2023 learning track, which is now widely adopted by the community. However, the training set contains only easy problems, and method performance might improve, especially for methods that solved all instances in the training set (see Tbl. A.1).

AI USE STATEMENT

Generative AI tools were used to assist with editing the manuscript and formatting in LaTeX, including tables and cross-references. They were not used to generate experimental data or to replace the authors' validation of scientific claims. The authors reviewed and verified all AI-assisted text and artifacts and take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

The method definitions and theoretical arguments are provided in sections 3 and 4. The experimental protocol, datasets, evaluation metrics, and compared methods are described in Sect. 6. The supplementary material provides pseudocode, complete training and evaluation details, hyperparameter selection, implementation and hardware information, and per-domain results in sections A.3 to A.5. All learned-method results report the specified random seeds. The implementation and scripts needed to reproduce the experiments will be released upon acceptance.

REFERENCES

Michael Aichmüller, Yannik Hesse, and Hector Geffner. Learning to search and searching to learn for generalization in planning. *CoRR*, abs/2605.25720, 2026. doi: 10.48550/ARXIV.2605.25720. URL https://doi.org/10.48550/arXiv.2605.25720.

Hyrum S. Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. Learning to evade static PE machine learning malware models via reinforcement learning. *CoRR*, abs/1801.08917, 2018.

Shahab Jabbari Arfaee, Sandra Zilles, and Robert C. Holte. Learning heuristic functions for large state spaces. *Artificial Intelligence*, 175(16-17):2075–2098, 2011. doi: 10.1016/J.ARTINT.2011.08.001. URL https://doi.org/10.1016/j.artint.2011.08.001.

Alex Ayoub, Kavosh Asadi, Dale Schuurmans, Csaba Szepesvári, and Karim Bouyarmane. Learning to reason efficiently with discounted reinforcement learning. *CoRR*, abs/2510.23486, 2025. doi: 10.48550/ARXIV.2510.23486. URL https://doi.org/10.48550/arXiv.2510.23486.

Julian Bernhard, Robert Gieselmann, Klemens Esterle, and Alois C. Knoll. Experience-based heuristic search: Robust motion planning with deep q-learning. In Wei-Bin Zhang, Alexandre M. Bayen, Javier J. Sánchez Medina, and Matthew J. Barth (eds.), *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*, pp. 3175–3182. IEEE, 2018. doi: 10.1109/ITSC.2018.8569436. URL https://doi.org/10.1109/ITSC.2018.8569436.

Mohak Bhardwaj, Sanjiban Choudhury, and Sebastian A. Scherer. Learning heuristic search via imitation. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, volume 78 of *Proceedings of Machine Learning Research*, pp. 271–280. PMLR, 2017. URL http://proceedings.mlr.press/v78/bhardwaj17a.html.

Dillon Z. Chen, Felipe W. Trevizan, and Sylvie Thiébaux. Return to tradition: Learning reliable heuristics with classical machine learning. In Sara Bernardini and Christian Muise (eds.), *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*, pp. 68–76. AAAI Press, 2024. doi: 10.1609/ICAPS.V34I1.31462. URL https://doi.org/10.1609/icaps.v34i1.31462.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 6278–6289, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/131f383b434fdf48079bff1e44e2d9a5-Abstract.html.

Alexander Chervov, Kirill Khoruzhii, Nikita Bukhal, Jalal Naghiyev, Vladislav Zamkovoy, Ivan Koltsov, Lyudmila Cheldieva, Arsenii Sychev, Arsenii Lenin, Mark Obozov, Egor Urvanov, and Alexey Romanov. A machine learning approach that beats rubik's cubes. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/fb82011040977c7712409fbdb5456647-Abstract-Conference.html.

Leah Chrestien, Stefan Edelkamp, Antonín Komenda, and Tomás Pevný. Optimize planning heuristics to rank, not to estimate cost-to-goal. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/50ea4dbd1cff6bd3daef939eff10c092-Abstract-Conference.html.

Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=H1dh6Ax0Z.

Guhao Feng and Han Zhong. Rethinking model-based, policy-based, and value-based reinforcement learning via the lens of representation complexity. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/0fe6a18be9491139fb759e2f645374b1-Abstract-Conference.html.

Patrick Ferber, Malte Helmert, and Jörg Hoffmann. Neural network heuristics for classical planning: A study of hyperparameter space. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang (eds.), *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pp. 2346–2353. IOS Press, 2020. doi: 10.3233/FAIA200364. URL https://doi.org/10.3233/FAIA200364.

Alan Fern, Roni Khardon, and Prasad Tadepalli. The first learning track of the international planning competition. *Mach. Learn.*, 84(1-2):81–107, 2011. doi: 10.1007/S10994-011-5234-Y. URL https://doi.org/10.1007/s10994-011-5234-y.

Hiroki Furuta, Tatsuya Matsushima, Tadashi Kozuno, Yutaka Matsuo, Sergey Levine, Ofir Nachum, and Shixiang Shane Gu. Policy information capacity: Information-theoretic measure for task complexity in deep reinforcement learning. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 3541–3552. PMLR, 2021. URL http://proceedings.mlr.press/v139/furuta21a.html.

Arthur Guez, Theophane Weber, Ioannis Antonoglou, Karen Simonyan, Oriol Vinyals, Daan Wierstra, Rémi Munos, and David Silver. Learning to search with mctsnets. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1817–1826. PMLR, 2018. URL http://proceedings.mlr.press/v80/guez18a.html.

Mingyu Hao, Felipe Trevizan, Sylvie Thiébaux, Patrick Ferber, and Jörg Hoffmann. Learned pairwise rankings for greedy best-first search. In *Proc. ICAPS Workshop on Reliable Data-Driven Planning and Scheduling*, 2024.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136. URL https://doi.org/10.1109/TSSC.1968.300136.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.*, 14:253–302, 2001. doi: 10.1613/JAIR.855. URL https://doi.org/10.1613/jair.855.

Rostislav Horcík and Gustav Sír. Expressiveness of graph neural networks in planning domains. In Sara Bernardini and Christian Muise (eds.), *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*, pp. 281–289. AAAI Press, 2024. doi: 10.1609/ICAPS.V34I1.31486. URL https://doi.org/10.1609/icaps.v34i1.31486.

Rostislav Horčík, Gustav Šír, Vítězslav Šimek, and Tomáš Pevný. State encodings for GNN-based lifted planners. In Toby Walsh, Julie Shah, and Zico Kolter (eds.), *Thirty-Ninth AAAI Conference on Artificial Intelligence, Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence, Fifteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2025, Philadelphia, PA, USA, February 25 - March 4, 2025*, pp. 26525–26533. AAAI Press, 2025. doi: 10.1609/AAAI.V39I25.34853. URL https://doi.org/10.1609/aaai.v39i25.34853.

Alexander Mattick and Christopher Mutschler. Reinforcement learning for node selection in branch-and-bound. *arXiv preprint arXiv:2310.00112*, 2023.

Nils J. Nilsson. *Problem-solving methods in artificial intelligence*. McGraw-Hill computer science series. McGraw-Hill, 1971. ISBN 0070465738. URL https://www.worldcat.org/oclc/185451766.

Laurent Orseau and Levi H. S. Lelis. Policy-guided heuristic search with guarantees. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 12382–12390. AAAI Press, 2021. doi: 10.1609/AAAI.V35I14.17469. URL https://doi.org/10.1609/aaai.v35i14.17469.

Laurent Orseau, Levi Lelis, Tor Lattimore, and Theophane Weber. Single-agent policy tree search with guarantees. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 3205–3215, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/52c5189391854c93e8a0e1326e56c14f-Abstract.html.

Roberta Raileanu and Rob Fergus. Decoupling value and policy for generalization in reinforcement learning. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8787–8798. PMLR, 2021. URL http://proceedings.mlr.press/v139/raileanu21a.html.

Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res.*, 39:127–177, 2010. doi: 10.1613/JAIR.2972. URL https://doi.org/10.1613/jair.2972.

Lara Scavuzzo, Feng Yang Chen, Didier Chételat, Maxime Gasse, Andrea Lodi, Neil Yorke-Smith, and Karen I. Aardal. Learning to branch with tree mdps. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9,*

2022, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/756d74cd58592849c904421e3b2ec7a4-Abstract-Conference.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

William Shen. Strips-hgn. https://github.com/williamshen-nz/STRIPS-HGN, 2020.

Oleksii Shuhailo, Karel Chvalovský, and Tomáš Pevný. Why reinforcement learning struggles with expression simplification: A reward analysis. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*, 2025.

Oleksii Shuhailo, Mohsen Ghaffari, Karel Chvalovský, and Tomáš Pevný. Understanding reward shaping in planning: A theoretical and empirical analysis. In *Proceedings of the International Symposium on Combinatorial Search*, volume 19, pp. 286–290, 2026.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In Akshat Kumar, Sylvie Thiébaux, Pradeep Varakantham, and William Yeoh (eds.), *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24, 2022*, pp. 629–637. AAAI Press, 2022. URL https://ojs.aaai.org/index.php/ICAPS/article/view/19851.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general policies with policy gradient methods. In Pierre Marquis, Tran Cao Son, and Gabriele Kern-Isberner (eds.), *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023*, pp. 647–657, 2023. doi: 10.24963/KR.2023/63. URL https://doi.org/10.24963/kr.2023/63.

Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 International Planning Competition. *AI Magazine*, 45(2):280–296, 2024. doi: 10.1002/aaai.12169.

Sam Toyer, Sylvie Thiébaux, Felipe W. Trevizan, and Lexing Xie. Asnets: Deep learning for generalised planning. *J. Artif. Intell. Res.*, 68:1–68, 2020. doi: 10.1613/JAIR.1.11633. URL https://doi.org/10.1613/jair.1.11633.

Tan Zhi-Xuan. PDDL.jl: An extensible interpreter and compiler interface for fast and flexible AI planning. Master's thesis, Massachusetts Institute of Technology, 2022.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=r1Ue8Hcxg.

A Appendix

A.1 GBFS Algorithm

Algorithm 1: Greedy Best-First Search (GBFS).

```
function GBFS(n0, heuristic)
    open_list = PriorityQueue()
    enqueue!(open_list, n0, heuristic(n0))
    close_list = Set()

    while !isempty(open_list)
        # remove node of minimum value
        n = dequeue!(open_list)

        if n in close_list
            continue # pruning
        end
        push!(close_list, n)

        if is_solution(n)
            return n
        end

        # Node expansion
        for c in children(n)
            enqueue!(open_list, c, heuristic(c))
        end
    end
    return false
end
```

A.2 Propositions

Proof of Prop. 1. Let $o^\star$ denote the maximum of o over the states reachable from s_0. For any sequence of expansions $(s_0, s_1, \ldots, s_n)$, the rewards telescope, and since $V(\mathcal{T}_0) = \{s_0\}$, the cumulative reward equals

$$G = \sum_{i=0}^{n} \tilde{R}(T_i, s_i) = \max_{s \in V(\mathcal{T}_{n+1})} o(s) - o(s_0).$$

Every state in $V(\mathcal{T}_{n+1})$ is reachable from s_0, hence $G \leq o^\star - o(s_0)$ for every sequence of expansions.

This upper bound is attained. Let s^+ be a reachable state with $o(s^+) = o^\star$ and let $(s'_0, s'_1, \ldots, s'_m)$ be a path without repeated states from $s'_0 = s_0$ to $s'_m = s^+$. Expanding $s'_0, \ldots, s'_{m-1}$ in this order is a valid sequence of actions, since each s'_{i+1} is a child of s'_i that has not been expanded before and is therefore in the open list. The resulting search tree contains s^+, so its cumulative reward is $o^\star - o(s_0)$. (If the episode terminates earlier because a best reachable state has been expanded, the search tree already contains a state with value $o^\star$.)

Consequently, the maximal cumulative reward is $o^\star - o(s_0)$, and a sequence of expansions attains it if and only if $\max_{s \in V(\mathcal{T}_{n+1})} o(s) = o^\star$, that is, if and only if $\mathcal{T}_{n+1}$ contains a state $s^\star$ with $o(s^\star) = o^\star$. □

A.3 Experiments Setup

Training protocol. Since the model does not have any prior knowledge about problems, the learning protocol follows the bootstrap process (Arfaee et al., 2011; Orseau & Lelis, 2021). For a given domain, the algorithm sequentially iterates over the training instances of that domain provided by the IPC 2023 learning track (Taitler et al., 2024) (89 to 99 instances, depending on the domain) and for each training instance performs the following steps:

1. Execute GBFS search algorithm using the current heuristic $h_\theta(s)$ until the problem is solved or the time limit of 30s is exhausted.
2. Identify the best node in the search tree from step 1, which is the node whose state has the lowest depth among the states with the highest objective function o.
3. Create the training set from the search tree from step 1 and the solution from step 2.

Algorithm 2: Bootstrap learning pipeline used in Sect. 6.

```
function TrainDomain(domain_problems, method, h_theta_init,
    hyperparams; epochs=10)
    h_theta = h_theta_init
    for epoch in 1:epochs
        for problem in domain_problems
            # GBFS: sample next nodes for expansions; time limit 30s
            T = RunGBFS(problem, h_theta; time_limit=30s, temperature
                =1.0)

            # Extract best node from the tree based on objective
                function
            n_best = BestNode(T; objective=o, tiebreak=:min_depth)
            # Build the set based on the best node and method
                specification.
            D = BuildTrainingSet(T, n_best, method)

            for k in 1:hyperparams.grad_steps
                theta = UpdateParameters(theta, D, method, hyperparams
                    )
            end
        end
    end
    return h_theta
end
```

4. Update the parameters of the model using the training set created in step 3.

Iterating over all training instances constitutes an epoch, the experiment is allowed to run for 10 epochs or a maximum of 24 hours. We use all ten domains of the learning track. The construction of the training set depends on the loss function and is described in relevant sections using the trajectory identified in step 2. The evaluation used models from the last 10th epoch.

Bootstrap learning pipeline details. Algorithm 2 summarizes the end-to-end training loop. The pseudocode uses the following helper routines and parameters:

- `RunGBFS(problem, h_theta; time_limit, temperature)`: executes GBFS guided by heuristic h_θ. Node expansions are sampled from the policy (softmax over $-h_\theta$) with temperature `temperature` rather than selecting the node from the queue; execution stops when the problem is solved or `time_limit` is reached.
- `BestNode(T; objective=o, tiebreak=:min_depth)`: returns the node in the search tree T with maximum objective value o; ties are broken by minimal depth. In our experiments, the objective function was the number of achieved subgoals.
- `BuildTrainingSet(T, n_best, method)`: constructs training data (supervised targets or RL rollouts) from the search tree T and the trajectory to `n_best`. In our experiments, we use the following constructions:
  - $\mathrm{L}_{\mathbf{gbfs}}$: extract the shortest path $\xi^\star$ from the root to `n_best`. For each step i on $\xi^\star$, store the open list of that configuration together with a label that the next node $s_i^\star$ on $\xi^\star$ should be preferred over all other open-list nodes (pairwise ranking objective; Eq. (8)).
  - $\mathrm{L}_{\mathbf{gbfs}}^{\mathbf{softmax}}$: uses the same extracted data as $\mathrm{L}_{\mathrm{gbfs}}$ (open lists along $\xi^\star$ with $s_i^\star$ as the preferred node), but optimizes a log-softmax / negative log-likelihood over the open list instead of pairwise comparisons (equations (5) and (9)).
  - $\mathrm{L}_{\mathbf{po}}$: uses the open lists actually observed during the GBFS run. We still extract $\xi^\star$ to `n_best` and, for each observed open list at time step t, label as the "correct" choice the on-trajectory node from $\xi^\star$ that is present in that open list (Eq. (7)).
  - **Tree-PPO**: extract a per-expansion training sequence from the Tree-MDP run: for each time step t, record the current search configuration (in particular the open list), the expanded node (action), and the step reward defined for Tree-MDP (Sect. 3). From this sequence, compute the per-step returns and PPO targets (advantages, value targets, policy targets).

  - **Linear-PPO**: extract a standard state–action training sequence from the Linear-MDP run: for each time step t, record the planning state, the chosen domain operator (action), and the step reward defined for Linear-MDP (Sect. 2). From this sequence, compute the per-step returns and PPO targets.
  - **LevinTS**: no training set is constructed. LevinTS reuses a policy trained by Linear-PPO and replaces GBFS only with Levin tree search, which orders states by the ratio of their depth to the probability of the path leading to them.
  - **PHS**: when a solution is found, extract the solution trajectory and build supervised targets for (i) the cost-to-goal estimator and (ii) the policy network as in Orseau & Lelis (2021).

- `UpdateParameters(theta, D, method, hyperparams)`: performs one optimization update on parameters `theta` using dataset `D` according to the specified method. The integer `hyperparams.grad_steps` is fixed for an experimental run (i.e., the same for all problem instances) and controls how many such updates are applied after processing each instance.

Evaluation and metric. The learning track provides, for each domain, the training instances described above and a disjoint set of 90 test instances, split by the organizers into 30 easy, 30 medium, and 30 hard ones. The easy test instances have roughly as many objects as the training instances, whereas the medium and hard ones are considerably larger (Tbl. A.6), so the test set measures generalization to larger problems. We follow this protocol. All hyperparameter configurations listed in Sect. A.4 are evaluated on both the training and test instances, and their scores are retained. For each method and domain, configuration selection is performed independently for each seed. We select the configuration with the highest IPC plan quality metric on the training instances for that seed and report its corresponding test results. Test-set scores do not influence this selection. Evaluation uses the same search as during training, a time limit of 30 s per instance, and no parameter updates. We report *coverage*, the percentage of test instances solved within this limit, averaged over three runs with seeds 1, 2 and 3, together with the standard deviation. Plan lengths and numbers of expanded states on the solved instances are reported in Sect. A.5.

A.4 Hyperparameter selection

For each method and domain, all configurations listed above were evaluated on the training and test instances for each seed, and both sets of scores were retained. Configuration selection is performed independently for each seed using the highest IPC plan quality metric on the training instances. We report the corresponding test results and aggregate them across seeds; test-set scores are not used for selection.

Hyper-parameters. Since the configuration of the ObjectBinary neural networks realizing the heuristic function was fixed according to results of Horčík et al. (2025), there are a few hyper-parameters: The learning rate of the ADAM optimizer, where we have experimented with following values $\{5 \cdot 10^{-4}, 10^{-4}\}$, and the number of gradient descent steps after solving the problem instance (Step 4 of bootstrap) which was varied in $\{10, 20\}$.

The Linear-PPO and Tree-PPO algorithms used clip ratio 0.2, GAE lambda 0.95, no entropy regularization, $\beta = 0$. We have tested two values of discount coefficients $\gamma \in \{0.99, 1\}$, since the first promotes shorter trajectories whereas the latter is more appropriate for an episodic setting. Policy and critic networks were implemented as two separate networks. The minibatch contained 100 actions, and the update in Step 4 of bootstrap performed one epoch of actor and 5 epochs of critic. While experiments involving GBFS were allowed to perform a maximum of 10 epochs, the Linear-PPO has tried the following combinations of epoch and time limit $\{(10/30s), (20/15s), (30/10s), (60/5s)\}$ treated as a hyperparameter.

Implementation. Experiments were implemented in Julia 1.13.0 using `PDDL.jl` and `SymbolicPlanners.jl` (Zhi-Xuan, 2022). ObjectBinary encoding implementation was taken from the repository of Horčík et al. (2025). All experiments ran on Intel Xeon Gold 6146 processors with 8 cores and 64GB memory per job, with a maximum total runtime of 24 hours.

	L_{gbfs}	$L_{gbfs}^{softmax}$	L_{po}	Tree-PPO	LPPO	PHS	LevinTS	GSP	FF	LAMA
Blocks	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	89.0 ± 11.0	63.0 ± 16.0	93.0 ± 9.0	80.0 ± 9.0	25.0 ± 0.0	66.0	99.0
Childsnack	**100.0 ± 0.0**	**100.0 ± 1.0**	95.0 ± 1.0	39.0 ± 15.0	23.0 ± 8.0	26.0 ± 2.0	25.0 ± 4.0	24.5 ± 0.0	12.0	81.0
Ferry	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	6.0 ± 2.0	**100.0 ± 0.0**	33.0 ± 0.0	70.0 ± 25.0	**100.0**	**100.0**
Floortile	30.0 ± 11.0	33.0 ± 6.0	**87.0 ± 8.0**	24.0 ± 0.0	2.0 ± 1.0	16.0 ± 0.0	16.0 ± 0.0	16.0 ± 0.0	18.0	35.0
Miconic	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	9.0 ± 0.0	**100.0 ± 0.0**	75.0 ± 0.0	79.0 ± 1.0	**100.0**	**100.0**
Rovers	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	94.0 ± 4.0	66.0 ± 16.0	90.0 ± 9.0	28.0 ± 4.0	21.0 ± 0.0	83.0	**100.0**
Satellite	**100.0 ± 0.0**	99.0 ± 1.0	**100.0 ± 0.0**	90.0 ± 12.0	96.0 ± 3.0	24.0 ± 1.0	43.0 ± 5.0	30.0 ± 2.0	**100.0**	**100.0**
Sokoban	93.0 ± 2.0	93.0 ± 1.0	96.0 ± **2.0**	76.0 ± 17.0	4.0 ± 1.0	60.0 ± 2.0	58.0 ± 1.0	59.0 ± 1.0	57.0	**100.0**
Spanner	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	97.0 ± 5.0	**100.0**	**100.0**
Transport	99.0 ± 2.0	**100.0 ± 0.0**	**100.0 ± 0.0**	99.0 ± 1.0	99.0 ± 2.0	18.0 ± 1.0	41.0 ± 19.0	18.0 ± 0.0	80.0	**100.0**
Average	92.0 ± 1.0	92.0 ± 0.0	**98.0 ± 1.0**	83.0 ± 0.0	47.0 ± 1.0	63.0 ± 1.0	50.0 ± 3.0	43.9 ± 0.0	71.0	91.0

Table A.1: Coverage (%) on the training instances, reported as mean ± standard deviation over three runs for all learned methods. FF and LAMA are deterministic, so we do not report a standard deviation for them. The best mean of a learned heuristic in each row is shown in bold.

A.5 Detailed results

This section provides additional metrics assessing solution quality (found plans) and solver efficiency.

Table A.4 shows the average number of expanded states on the test instances, as mean and standard deviation over three runs of the learned methods; LAMA is deterministic and is reported as a single value.

Table A.1 shows average coverage on problem instances in the training set. Table A.2 reports test coverage separately for the easy, medium, and hard instance subsets.

Table A.3 shows search efficiency on training instances, measured as the plan length divided by the number of expanded states, higher is better, and optimal behavior equals one. For Linear-PPO (LPPO) this quantity is most of the time one, because it does not use tree search. The purpose of this table is to show that during training, L_{po} achieves a nearly optimal heuristic in many domains in the sense that it expands only nodes on the trajectory of the returned plan, in which case its training set is equal (or very close if the fraction is smaller than one) to L_{gbfs} and $L_{gbfs}^{softmax}$, which explains why all these methods behave similarly.

Table A.5 compares the quality of returned plans. To compare the quality of returned plans, we use the IPC plan quality metric (Fern et al., 2011) instead of plan length, because each method solves a different subset of the test instances. The plan quality metric is equal to l_{best}/l, where l is the length of its plan (0 if the instance is not solved) and l_{best} is the shortest plan found by any of the methods compared in this paper. We average the scores over all test instances. While the reference length l_{best} is not necessarily optimal, the score allows a relative comparison that accounts for both coverage and plan quality. As shown in Tbl. A.5, L_{po} obtains the highest average among the learned heuristics and ties LAMA at 49.0% overall.

A.6 Sizes of the training and test instances

Table A.6 lists, for each domain of the IPC 2023 learning track, the number of training instances and the range of objects in the training instances and in the easy, medium, and hard test instances (30 each).

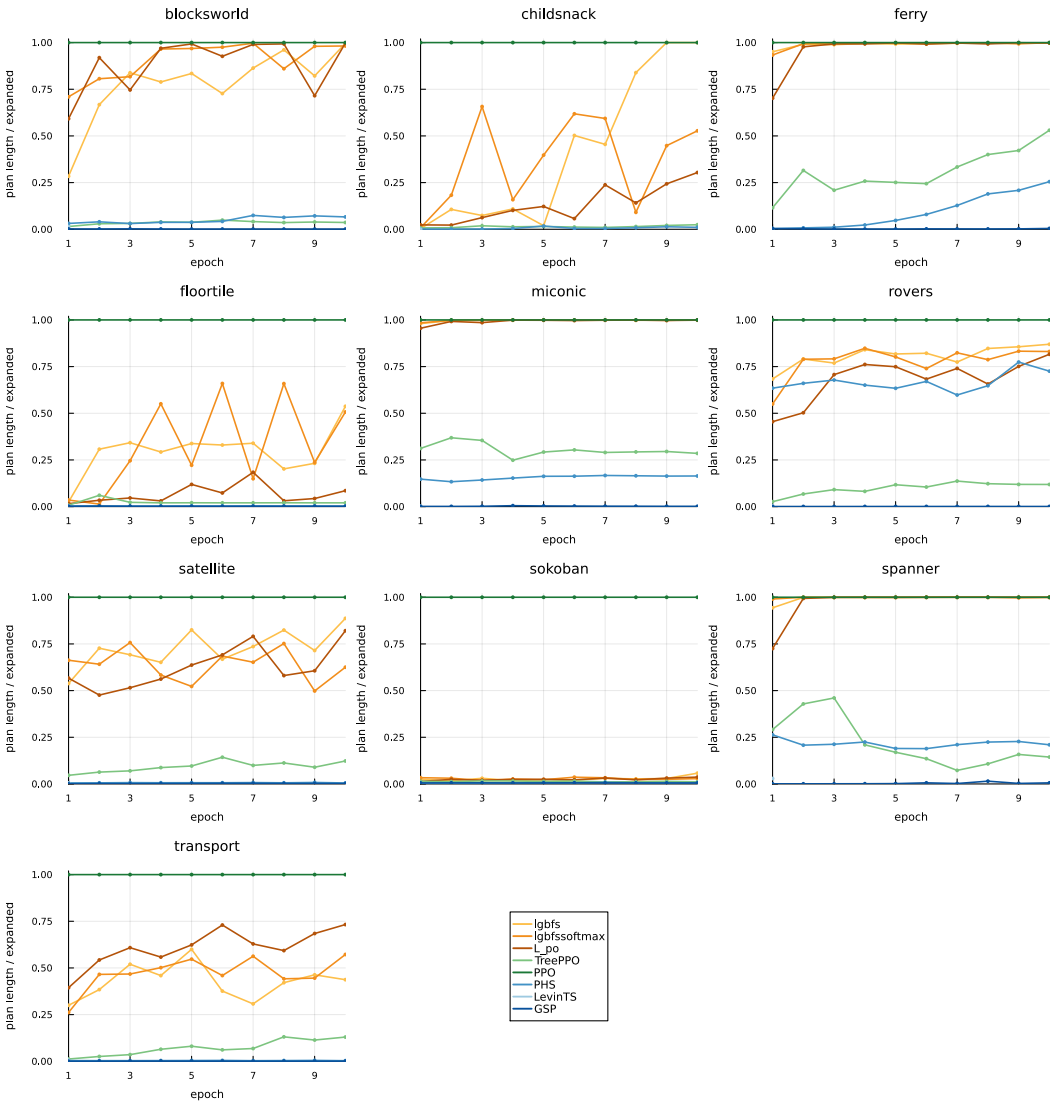


Figure 1: Plan length divided by the number of expanded nodes on solved training instances over all training epochs, shown separately for every method and domain. Higher values indicate that fewer off-plan nodes are expanded relative to the returned plan length.

Domain	Difficulty	L_{gbfs}	$L_{gbfs}^{softmax}$	L_{po}	Tree-PPO	LPPO	PHS	LevinTS	GSP	FF	LAMA
Blocks	easy	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	93.3 ± 11.5	68.9 ± 7.7	**100.0 ± 0.0**	75.6 ± 5.1	13.3 ± 0.0	73.3	**100.0**
Blocks	medium	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	6.7 ± 8.8	1.1 ± 1.9	8.9 ± 6.9	0.0 ± 0.0	0.0 ± 0.0	0.0	13.3
Blocks	hard	17.8 ± 17.1	25.6 ± 5.1	**50.0 ± 14.5**	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0	0.0
Childsnack	easy	98.9 ± 1.9	**100.0 ± 0.0**	96.7 ± 5.8	40.0 ± 9.4	12.2 ± 21.2	34.4 ± 20.4	16.7 ± 0.0	16.7 ± 0.0	0.0	86.7
Childsnack	medium	**55.6 ± 13.5**	54.4 ± 17.1	37.8 ± 10.7	0.0 ± 0.0	1.1 ± 1.9	4.4 ± 5.1	0.0 ± 0.0	0.0 ± 0.0	0.0	13.3
Childsnack	hard	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0	0.0
Ferry	easy	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	2.2 ± 3.8	**100.0 ± 0.0**	23.3 ± 0.0	**100.0 ± 0.0**	**100.0**	**100.0**
Ferry	medium	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	78.9 ± 5.1	0.0 ± 0.0	60.0 ± 17.6	0.0 ± 0.0	53.3 ± 0.0	63.3	**100.0**
Ferry	hard	**3.3 ± 5.8**	0.0 ± 0.0	1.1 ± 1.9	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0	0.0
Floortile	easy	17.8 ± 8.4	8.9 ± 1.9	**96.7 ± 3.3**	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	3.3 ± 3.3	0.0 ± 0.0	3.3	36.7
Floortile	medium	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0	0.0
Floortile	hard	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0	0.0
Miconic	easy	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	2.2 ± 1.9	**100.0 ± 0.0**	70.0 ± 0.0	83.3 ± 5.8	**100.0**	**100.0**
Miconic	medium	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	97.8 ± 1.9	0.0 ± 0.0	66.7 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	76.7	96.7
Miconic	hard	21.1 ± 1.9	20.0 ± 0.0	20.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0	**70.0**
Rovers	easy	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	98.3 ± 2.4	88.9 ± 3.8	94.4 ± 5.1	26.7 ± 3.3	13.3 ± 0.0	76.7	96.7
Rovers	medium	15.6 ± 1.9	14.4 ± 3.8	14.4 ± 5.1	3.3 ± 4.7	1.1 ± 1.9	1.1 ± 1.9	0.0 ± 0.0	0.0 ± 0.0	0.0	**56.7**
Rovers	hard	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0	0.0
Satellite	easy	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	97.8 ± 1.9	**100.0 ± 0.0**	55.6 ± 43.4	36.7 ± 0.0	43.3 ± 0.0	**100.0**	96.7
Satellite	medium	78.9 ± 13.5	77.8 ± 3.8	82.2 ± 11.7	38.9 ± 15.4	18.9 ± 13.9	4.4 ± 7.7	0.0 ± 0.0	0.0 ± 0.0	23.3	**100.0**
Satellite	hard	1.1 ± 1.9	1.1 ± 1.9	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0	**23.3**
Sokoban	easy	92.2 ± 8.4	93.3 ± 5.8	**100.0 ± 0.0**	98.9 ± 1.9	0.0 ± 0.0	57.8 ± 1.9	55.6 ± 1.9	57.8 ± 1.9	53.3	**100.0**
Sokoban	medium	2.2 ± 3.8	5.6 ± 5.1	2.2 ± 3.8	2.2 ± 1.9	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0	**13.3**
Sokoban	hard	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0	0.0
Spanner	easy	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0**	**100.0**
Spanner	medium	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	84.4 ± 19.0	**100.0 ± 0.0**	56.7 ± 25.2	0.0 ± 0.0	65.6 ± 56.8	0.0	0.0
Spanner	hard	15.6 ± 5.1	16.7 ± 0.0	**17.8 ± 1.9**	8.9 ± 7.7	7.8 ± 7.7	10.0 ± 8.8	0.0 ± 0.0	0.0 ± 0.0	0.0	0.0
Transport	easy	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	97.8 ± 1.9	**100.0 ± 0.0**	20.0 ± 0.0	43.3 ± 18.6	20.0 ± 0.0	86.7	**100.0**
Transport	medium	55.6 ± 8.4	45.6 ± 5.1	**82.2 ± 6.9**	25.6 ± 5.1	34.4 ± 5.1	0.0 ± 0.0	4.4 ± 7.7	0.0 ± 0.0	0.0	73.3
Transport	hard	0.0 ± 0.0	0.0 ± 0.0	**2.2 ± 1.9**	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0	0.0
Average	easy	90.9 ± 1.0	90.2 ± 0.8	**99.3 ± 0.9**	82.3 ± 0.0	47.4 ± 2.8	66.2 ± 3.0	45.1 ± 2.5	44.8 ± 0.0	69.3	91.7
Average	medium	60.8 ± 1.0	59.8 ± 1.1	**61.9 ± 1.6**	35.0 ± 0.0	15.7 ± 1.5	20.2 ± 4.7	0.4 ± 0.8	11.9 ± 0.0	16.3	46.7
Average	hard	5.9 ± 1.7	6.3 ± 0.7	9.1 ± 1.3	0.0 ± 0.0	0.8 ± 0.8	1.0 ± 0.9	0.0 ± 0.0	0.0 ± 0.0	0.0	**9.3**

Table A.2: Coverage (%) on the test instances, split by difficulty. Each difficulty subset contains 30 instances per domain. Learned methods are reported as mean ± standard deviation over three runs. FF and LAMA are deterministic, so we report no standard deviation for them. The best reported value in each row is shown in bold.

	L_{gbfs}	$L_{gbfs}^{softmax}$	L_{po}	Tree-PPO	LPPO	PHS	LevinTS	GSP	FF	LAMA
Blocks	0.4584 ± 0.2574	0.6392 ± 0.0938	0.8744 ± 0.1854	0.0346 ± 0.0143	**1.0000 ± 0.0000**	0.1357 ± 0.0678	0.0146 ± 0.0030	0.0009 ± 0.0000	0.0233	0.0005
Childsnack	0.7557 ± 0.1219	**0.7804 ± 0.3600**	0.4268 ± 0.4562	0.0299 ± 0.0043	0.6667 ± 0.5774	0.2333 ± 0.2516	0.0005 ± 0.0002	0.0002 ± 0.0001	0.0000	0.0002
Ferry	0.6482 ± 0.3248	0.3850 ± 0.2666	**0.7418 ± 0.2364**	0.0515 ± 0.0101	0.3333 ± 0.5774	0.0858 ± 0.0428	0.0005 ± 0.0001	0.0135 ± 0.0000	0.3244	0.0016
Floortile	0.3353 ± 0.5712	0.3239 ± 0.5485	**0.3980 ± 0.2448**	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0004 ± 0.0003	0.0000 ± 0.0000	0.0034	0.0002
Miconic	0.9822 ± 0.0214	0.9999 ± 0.0002	**1.0000 ± 0.0000**	0.2269 ± 0.0564	0.6667 ± 0.5774	0.0577 ± 0.0034	0.0022 ± 0.0000	0.0026 ± 0.0018	0.6975	0.0072
Rovers	0.5044 ± 0.0427	0.6688 ± 0.1030	0.7228 ± 0.0326	0.1125 ± 0.0103	**1.0000 ± 0.0000**	0.8463 ± 0.0446	0.0044 ± 0.0036	0.0016 ± 0.0001	0.1910	0.0008
Satellite	0.5219 ± 0.0752	0.6675 ± 0.1341	0.6540 ± 0.1054	0.1926 ± 0.1235	**1.0000 ± 0.0000**	0.4594 ± 0.4092	0.0051 ± 0.0032	0.0069 ± 0.0000	0.6628	0.0012
Sokoban	**0.0498 ± 0.0466**	0.0357 ± 0.0194	0.0430 ± 0.0197	0.0128 ± 0.0053	0.0000 ± 0.0000	0.0055 ± 0.0012	0.0032 ± 0.0006	0.0064 ± 0.0003	0.0071	0.0010
Spanner	0.7536 ± 0.4267	0.9994 ± 0.0011	**1.0000 ± 0.0000**	0.3539 ± 0.4384	**1.0000 ± 0.0000**	0.7509 ± 0.2732	0.0222 ± 0.0105	0.6482 ± 0.5602	0.0527	0.0001
Transport	0.3406 ± 0.0502	0.2253 ± 0.0661	0.4145 ± 0.1766	0.0740 ± 0.0230	**1.0000 ± 0.0000**	0.0025 ± 0.0004	0.0122 ± 0.0153	0.0006 ± 0.0000	0.0727	0.0012
Average	0.5350 ± 0.0465	0.5725 ± 0.0717	0.6275 ± 0.0216	0.0822 ± 0.0000	**0.6667 ± 0.1528**	0.2577 ± 0.0931	0.0065 ± 0.0013	0.0681 ± 0.0000	0.2035	0.0014

Table A.3: Plan length divided by the number of expanded nodes on solved training instances, reported as mean ± standard deviation over three runs for all learned methods. FF and LAMA are deterministic, so we report no standard deviation for them. Larger values indicate that fewer off-plan nodes are expanded relative to the returned plan length.

	L_{gbfs}	$L_{gbfs}^{softmax}$	L_{po}	Tree-PPO	LPPO	PHS	LevinTS	GSP	FF	LAMA
Blocks	554 ± 201	376 ± 42	$\mathbf{357 \pm 28}$	4112 ± 2853	1200 ± 255	2797 ± 1417	5454 ± 515	$17{,}272 \pm 0$	3232	191,532
Childsnack	55 ± 7	71 ± 34	194 ± 131	825 ± 224	$\mathbf{44 \pm 45}$	605 ± 914	$37{,}412 \pm 14{,}880$	$84{,}713 \pm 0$	0	193,416
Ferry	248 ± 164	621 ± 710	169 ± 53	1919 ± 459	$\mathbf{3 \pm 5}$	1415 ± 534	$27{,}593 \pm 4192$	4876 ± 0	232	75,008
Floortile	$9482 \pm 13{,}191$	$11{,}342 \pm 17{,}046$	214 ± 94	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$37{,}185 \pm 32{,}211$	$\mathbf{0 \pm 0}$	8000	191,398
Miconic	105 ± 6	100 ± 3	100 ± 0	405 ± 119	$\mathbf{3 \pm 2}$	1062 ± 39	6280 ± 26	7926 ± 3462	100	30,159
Rovers	256 ± 21	199 ± 5	$\mathbf{150 \pm 4}$	643 ± 56	206 ± 27	392 ± 152	6684 ± 6623	6893 ± 316	250	149,884
Satellite	156 ± 32	117 ± 26	118 ± 7	348 ± 135	$\mathbf{90 \pm 49}$	1081 ± 1102	2903 ± 1835	1787 ± 0	35	56,214
Sokoban	1268 ± 1145	1391 ± 651	940 ± 368	2842 ± 921	$\mathbf{0 \pm 0}$	3363 ± 755	4528 ± 779	2604 ± 232	1880	59,225
Spanner	145 ± 105	82 ± 1	83 ± 2	439 ± 354	$\mathbf{74 \pm 7}$	82 ± 45	769 ± 351	4071 ± 6931	264	130,087
Transport	$\mathbf{233 \pm 57}$	368 ± 112	289 ± 184	943 ± 501	294 ± 114	5004 ± 790	3140 ± 1627	$12{,}268 \pm 0$	550	44,626
Average	1250 ± 1268	1467 ± 1644	261 ± 37	1716 ± 0	$\mathbf{191 \pm 34}$	1580 ± 428	$13{,}195 \pm 1952$	$14{,}241 \pm 0$	1454	112,155

Table A.4: Average number of expanded states on the test instances solved by the respective method, reported as mean $\pm$ standard deviation over three runs for all learned methods. FF and LAMA are deterministic and are reported as single values. LAMA continues searching after finding a solution in order to improve its plan, whereas the learned GBFS methods stop at the first solution. The averages are taken over different sets of instances when coverage differs and over the available domains.

	L_{gbfs}	$L_{gbfs}^{softmax}$	L_{po}	Tree-PPO	LPPO	PHS	LevinTS	GSP	FF	LAMA
Blocks	70.0 ± 5.0	73.0 ± 2.0	$\mathbf{81.0 \pm 5.0}$	19.0 ± 5.0	7.0 ± 1.0	11.0 ± 3.0	15.0 ± 1.0	4.0 ± 0.0	15.0	38.0
Childsnack	$\mathbf{50.0 \pm 6.0}$	44.0 ± 2.0	34.0 ± 3.0	10.0 ± 2.0	3.0 ± 4.0	11.0 ± 7.0	5.0 ± 1.0	6.0 ± 0.0	0.0	33.0
Ferry	$\mathbf{67.0 \pm 3.0}$	66.0 ± 0.0	66.0 ± 0.0	56.0 ± 2.0	1.0 ± 1.0	39.0 ± 5.0	8.0 ± 0.0	50.0 ± 0.0	53.0	**67.0**
Floortile	5.0 ± 2.0	3.0 ± 1.0	$\mathbf{29.0 \pm 3.0}$	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.0 ± 1.0	0.0 ± 0.0	1.0	12.0
Miconic	$\mathbf{71.0 \pm 0.0}$	$\mathbf{71.0 \pm 2.0}$	$\mathbf{71.0 \pm 0.0}$	61.0 ± 0.0	1.0 ± 1.0	53.0 ± 2.0	23.0 ± 0.0	27.0 ± 1.0	56.0	89.0
Rovers	24.0 ± 2.0	23.0 ± 1.0	$\mathbf{28.0 \pm 4.0}$	27.0 ± 2.0	12.0 ± 2.0	11.0 ± 2.0	8.0 ± 0.0	4.0 ± 0.0	24.0	51.0
Satellite	41.0 ± 5.0	40.0 ± 3.0	$\mathbf{42.0 \pm 2.0}$	27.0 ± 4.0	16.0 ± 1.0	4.0 ± 2.0	10.0 ± 2.0	10.0 ± 0.0	40.0	73.0
Sokoban	27.0 ± 3.0	28.0 ± 2.0	30.0 ± 1.0	$\mathbf{31.0 \pm 0.0}$	0.0 ± 0.0	17.0 ± 0.0	18.0 ± 1.0	17.0 ± 1.0	18.0	38.0
Spanner	$\mathbf{60.0 \pm 1.0}$	$\mathbf{60.0 \pm 1.0}$	$\mathbf{60.0 \pm 1.0}$	58.0 ± 9.0	57.0 ± 2.0	51.0 ± 8.0	32.0 ± 0.0	46.0 ± 15.0	33.0	33.0
Transport	38.0 ± 3.0	35.0 ± 1.0	$\mathbf{52.0 \pm 1.0}$	27.0 ± 1.0	18.0 ± 1.0	4.0 ± 0.0	12.0 ± 5.0	7.0 ± 0.0	24.0	58.0
Average	45.0 ± 0.0	44.0 ± 0.0	$\mathbf{49.0 \pm 0.0}$	31.0 ± 0.0	11.0 ± 1.0	20.0 ± 0.0	13.0 ± 1.0	17.1 ± 0.0	26.0	**49.0**

Table A.5: IPC plan quality metric (Fern et al., 2011) in percents, reported as mean $\pm$ standard deviation over three runs for all learned methods. FF and LAMA are deterministic, so we do not report a standard deviation for them. For each test instance, the plan quality metric is computed as l_{best}/l, where l is the length of its plan (0 if the instance is not solved) and l_{best} is the shortest plan found by any compared method; the scores are averaged over all test instances. The best reported value in each row is shown in bold.

Domain	# train	train	test easy	test medium	test hard
Blocksworld	99	2–29	5–29	35–146	160–488
Childsnack	98	6–51	20–51	65–187	207–1326
Ferry	99	3–35	7–35	30–146	300–1461
Floortile	97	5–45	15–37	116–283	667–1022
Miconic	99	3–30	5–30	50–137	130–681
Rovers	99	10–36	12–36	49–249	294–596
Satellite	99	5–43	11–43	48–153	145–402
Sokoban	99	50–173	65–173	405–2435	3640–9880
Spanner	89	6–28	9–28	63–184	203–833
Transport	99	6–45	12–39	40–109	111–354

Table A.6: Number of training instances and range of objects per instance in the training set and in the easy, medium, and hard test sets.