Combining Heuristics and Transition Classifiers in Classical Planning

Farid Musayev[1], Dominik Drexler[1], Daniel Gnad[1,2] and Jendrik Seipp[1]

[1]Linköping University, Sweden
{farid.musayev, dominik.drexler, daniel.gnad, jendrik.seipp}@liu.se
[2]Heidelberg University, Germany

Abstract. Recent work on learning for classical planning has primarily focused on exclusively employing the learned heuristics or policies. However, no purely learning-based method has consistently outperformed state-of-the-art planners to date. To address this, we return to the research paradigm that integrates learned domain knowledge with traditional, non-learned planning techniques. We propose a novel and simple approach for learning transition classifiers, using tree-based statistical learning over description logic features. In experiments, we evaluate various strategies for integrating learned classifiers with the FF heuristic, a state-of-the-art non-learned heuristic. Our results demonstrate that augmenting classical heuristics with transition classifiers leads to substantial performance improvements. The strongest variant combines classifier-based lookahead search with learned knowledge to avoid transitions into unsolvable states, frequently outperforming state-of-the-art traditional and learning-based planners.

1 Introduction

Recent research has introduced many learning-based approaches to classical planning. Many works focus on learning heuristics or policies to guide a search process, using either traditional learning techniques [28, 4], or modern deep learning architectures [31, 29, 13, 27]. An alternative approach builds on finding general policies expressed in a formal target language by encoding the learning task as an optimization problem or a new planning task [2, 11, 8, 24]. Both approaches follow the paradigm of exclusively employing the learned domain knowledge, either by guiding a best-first search or greedily executing the learned policy. However, to date, none of these methods outperforms traditional non-learning planners across arbitrary domains. Some techniques are only applicable in domains that are optimally solvable in polynomial time, others do not consistently outperform planners based on well-established heuristics like h^{FF} [16].

In this work, we combine the strengths of both paradigms, learning and planning, instead of relying exclusively on a learned model. To achieve this, we analyze several ways of combining a satisficing heuristic with our domain knowledge in a search algorithm [32, 5, 18, 12].

On the learning side, our work combines concepts from Francès et al. [11], who use models based on description logic features, and from Ferber et al. [10], utilizing tree-based methods for learning domain knowledge. Given a set of training instances in a planning domain, we expand the state spaces and label transitions as good, bad, or unsolvable, meaning that the optimal goal distance decreases, does not decrease, or the transition leads into a state from which the goal cannot be reached, respectively. We use description logic features [1] to characterize the state transitions. The learning task is then set as a multi-label classification problem, training a decision tree to separate good, bad, and unsolvable transitions. From a learned tree, we extract a set of rules for each label and show that this can be translated into transition classifiers which are closely related to policies as defined in the framework of Francès et al. [11]. During the search, we employ these classifiers to 1) greedily follow good transitions, 2) break tie between states with identical heuristic values, 3) identify preferred operators, 4) compute a path-dependent heuristic that evaluates the taken transitions according to their label, or 5) detect unsolvable states.

In our evaluation, we demonstrate that a classifier-based lookahead search (1) that uses a traditional non-learned heuristic and our domain knowledge to identify unsolvable states (5) is an effective combination. It improves both the coverage and the number of state expansions required to solve instances across several domains when compared to other combinations, including a plain non-learned heuristic and purely learning-based methods.

2 Background

In this section, we review classical planning, heuristic search, description logics, transition classifiers and decision trees, drawing from Francès et al. [11]. If not mentioned otherwise, we assume sets to be finite.

2.1 Classical Planning

A first-order **planning domain** D is a tuple $\langle Q, ar, A\rangle$ where Q is a set of predicates, $ar : Q \to \mathbb{N}$ is a function that maps predicates to their arity, and A is a set of actions schemas of the form $\langle pre, \mathit{eff}\rangle$, where pre is the precondition, an arbitrary first-order formula, and eff is an arbitrary first-order effect.

A **planning problem** (or problem) P is a pair $\langle D, I\rangle$ where D is a planning domain and $I = \langle O, s_0, G\rangle$ is problem-specific information that consists of a set of objects O and two sets of ground atoms s_0 and G describing the initial state and the goal condition. A **ground atom** $p(\bar{o})$ over a predicate p denotes that $\bar{o} = o_1, \ldots, o_{ar(p)}$ with $o_i \in O$ for all $1 \leq i \leq n$ is in the relation with name p. A **state** s is a set of ground atoms over the predicates and objects. The set of

all possible states is denoted by S. The initial state is s_0, and the set of all goal states is $S_G = \{s \in S \mid G \subseteq s\}$. A **ground action** is an action schema after substituting all variables with objects. A ground action is applicable in a state if the ground propositional precondition formula holds in the state. Applying the effect of a ground action in a state produces a unique successor state.

A planning problem P induces a state space $S_P = \langle S, s_0, S_G, Succ\rangle$ where S, s_0 and S_G are defined as above and $Succ$ is a binary relation over states where $\langle s, s'\rangle \in Succ$ iff there exists a ground action that is applicable in s and its application in s yields the successor state s'. A trajectory seeded at a state s_1 of P is a state sequence $s_1, s_2, \ldots, s_n$ such that $\langle s_i, s_{i+1}\rangle \in Succ$ for all $1 \leq i < n$. A state s is **reachable** in P if there exists a trajectory seeded at s_0 that ends in s. A **plan** for s is a trajectory seeded at s that ends in a goal state. A plan for s is **optimal** if it is a plan for s of minimal length. A state s is **solvable** if a plan for s exists, and otherwise is **unsolvable**. The goal distance $V^*(s)$ of a state s is the length of an optimal plan for s. We set $V^*(s)$ to ∞ if no plan for s exists.

2.2 Heuristic Search

A common approach to solving classical planning tasks is heuristic search. It uses domain-independent goal distance estimators to guide a search in the state space, starting from the initial state. More formally, a **heuristic** is a function $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$ that maps states into an estimate for the plan length. In this work, we focus on greedy best-first search [6], which expands states in order of increasing heuristic value.

2.3 Features & Transition Classifiers

A **feature** is a function of the state. We consider two types of features. Boolean features map states into true or false and numeric features map states into non-negative integers.

We use features to define **conditions** and **effects** as follows. The possible conditions for a Boolean feature p (resp. numeric feature n) are $p, \neg p$ (resp. $n > 0, n = 0$). A state s satisfies the condition p (resp. $\neg p$) iff p (resp. $\neg p$) is true in s, and the condition $n > 0$ (resp. $n = 0$) iff n is greater than 0 (resp. n is equal to 0). The possible effects for a Boolean feature p (resp. numeric feature n) are $p, \neg p, p?$ (resp. $n\uparrow, n\downarrow, n\updownarrow, n\downarrow\!\!\uparrow, n?$). A state pair $\langle s, s'\rangle$ satisfies the effect p iff p is true in s', $\neg p$ iff $\neg p$ is true in s', $n\uparrow$ iff $n(s) < n(s')$, $n\downarrow$ iff $n(s) > n(s')$, $n\updownarrow$ iff $n(s) \leq n(s')$, and $n\downarrow\!\!\uparrow$ iff $n(s) \geq n(s')$. The effects $p?$ and $n?$ define that there is no constraint on the value or feature change.[1]

A **transition classifier** c_Φ is a relation over state pairs. A transition classifier that characterizes good transitions is also called a *policy*. We can represent a transition classifier as a set of rules over features Φ, each of the form $C \mapsto E$, where C is a set of feature conditions and E is a set of feature effects. A state pair $\langle s, s'\rangle$ is **compatible** with a rule iff s satisfies all conditions in C and $\langle s, s'\rangle$ satisfies all effects in E. A state pair $\langle s, s'\rangle$ is in c_Φ, or c_Φ-compatible, iff $\langle s, s'\rangle$ is compatible with a rule in c_Φ. A state trajectory $s_1, s_2, \ldots, s_n$ for s_1 that ends in s_n is c_Φ-compatible if $\langle s_i, s_{i+1}\rangle$ is c_Φ-compatible for $1 \leq i < n$.

2.4 Description Logics

Description logics (DLs) are a family of knowledge representation languages [1] where **concepts** represent unary derived predicates and **roles** represent binary derived predicates over a universe Δ representing the set of objects O of a problem P over a domain D with predicates Q. We use the grammar of DLs to generate a set of features Φ. For details on the concepts and grammar rules, we refer to Francès et al. [11]. The features Φ are domain general, meaning that they are not specific to a single planning instance, but rather represent knowledge about the planning domain D. Similar to Francès et al. [11], we limit the exponential growth of the feature set by limiting the syntactic complexity and the total number of generated features. The syntactic complexity of a feature is defined as the number of grammar rules used to construct it.

Example 1. *In the Spanner domain, an agent needs to collect spanners, reach the end of the corridor and tighten loosened nuts. As an example, the following DL features $\{n, h, e\}$ (two numeric, one Boolean) can be generated:*

- $n \equiv |tightened_g \sqcap \neg tightened|$*: number of untightened nuts,*
- $h \equiv |\exists at.\top|$*: number of objects not held by the agent,*
- $e \equiv |\exists at.(\forall at^{-1}.man)|$*: Boolean indicating whether the agent location is empty, i.e., there is no spanner or nut at it.*

2.5 Decision Trees

A **decision tree** [3] over a set of features Φ is a single rooted binary tree. Each **inner node** v is a tuple $v = \langle v_l, v_r, \ell_l, \ell_r\rangle$, where v_l and v_r are the left and right **child nodes** of v, and ℓ_l and ℓ_r are edge constraints describing Boolean **feature conditions** or **effects** over a feature in Φ. A **leaf node** v_i is a tuple $\langle\langle x_1, y_1\rangle, \ldots, \langle x_{k_i}, y_{k_i}\rangle\rangle$ where each pair $\langle x_j, y_j\rangle$ for all $1 \leq j < k_i$ consists of a transition x_j and its associated label y_j. For a given transition $\langle s, s'\rangle$, the **value** of a node $v(s, s')$ is recursively defined as follows: if v is a leaf node, then the label is the one with the largest occurrence in the node, where ties are broken arbitrarily. If v is an inner node, then it is the value of the left child (resp. right child) if $\langle s, s'\rangle$ satisfies ℓ_l (resp. ℓ_r). A path in the decision tree is a sequence of nodes $v_1, v_2, \ldots, v_k$ such that v_{i+1} is a child of v_i.

3 Learning Transition Classifiers

We next describe our algorithm for learning domain knowledge and transforming it into a set of policy rules. The algorithm consists of three steps: 1) self-supervised generation of training data that consists of a set of labeled transitions and description logic features, 2) learning a decision tree over a subset of the features to classify transitions, and 3) translating the decision tree into a transition classifier for each label.

3.1 Data Generation

Given a set of training instances $\mathcal{P}$, we generate their induced state spaces. For each state s in the induced state space, we label each outgoing transition $\langle s, s'\rangle$ as *good* if the optimal goal distance decreases, i.e., $V^*(s) > V^*(s')$, *bad* if the optimal goal distance does not decrease and s' is solvable, i.e., $V^*(s) \leq V^*(s') \neq \infty$, and *unsolvable* if the transition reaches an unsolvable state, i.e., $V^*(s) < \infty$ and $V^*(s') = \infty$. We randomly sample transitions from the fully expanded state spaces of the training instances. To limit the combinatorial explosion during state space expansion, we set limits on the size of the state spaces and the time for expansion. Our aim is to learn classifiers that can generalize across D, particularly to instances of a much larger size than those seen during training. Given the qualitative

[1] We extended the language of general policies with $n\updownarrow$ and $n\downarrow\!\!\uparrow$ to capture precisely the opposite meanings of $n\downarrow$ and $n\uparrow$ respectively.

nature of our features, we assume that uniform random sampling provides us with transitions that are representative of instances of arbitrary size and sufficient for generalization to larger instances.

The set of uniformly sampled transitions $\mathcal{T}$ induces a set of states $S_{\mathcal{T}}$, i.e., $s, s' \in S_{\mathcal{T}}$ for each $\langle s, s' \rangle \in \mathcal{T}$. For feature generation, we randomly subsample state pairs from $S_{\mathcal{T}}$ that correspond to transitions in $\mathcal{T}$ and obtain a set of subsampled state pairs $SS_{\mathcal{T}}$. The aim is to generate features that can distinguish transitions in $\mathcal{T}$ with different labels. From $SS_{\mathcal{T}}$, we generate a pool of candidate features Φ over the description logic grammar as follows. First, we generate sentences over primitive concepts and roles with the syntactic complexity of one. Then, we iteratively generate sentences over composite concepts and roles and Boolean and numeric features from the previously generated ones with a syntactic complexity of one plus the syntactic complexity of the sub-sentences. A feature f makes it into the pool Φ if no previously generated feature evaluates equivalently on all states in $SS_{\mathcal{T}}$. For more details on the feature generation, we refer to Francès et al. [11] and Drexler and Seipp [7].

3.2 Learning a Decision Tree

We learn a binary decision tree on the training set of transitions $\mathcal{T}$ using the CART algorithm [3]. We associate each node in the tree with a set of transitions from $\mathcal{T}$. Starting from the root node, the algorithm recursively partitions the feature space at each inner node using a transition feature (condition or effect) over a feature in Φ that minimizes the Gini index. Notice that such a transition feature splits the transitions at each inner node into two sets. The first set corresponds to the transitions where the condition or effect holds, and the second set to the transitions where it does not. The algorithm terminates when a maximum number of leaf nodes or a maximum tree depth is reached. Each leaf node is assigned a label ("good", "bad", or "unsolvable") based on the majority voting over the labels of transitions in the leaf node.

3.3 From a Decision Tree to Transition Classifiers

A decision tree over transition features is closely related to a general policy [11] which corresponds to a transition classifier that classifies good transitions without any false positives for the whole domain. We can translate a decision tree over transition features into the language of general policies when adding the numeric feature effects $n\downarrow$ and $n\updownarrow$ that capture precisely the opposite meanings of feature effects $n\uparrow$ and $n\downarrow$, respectively. Adding the numeric feature effects ensures that the resulting size of transition classifiers is polynomial in the size of the decision tree.

Theorem 1. *For each decision tree with root node v, there exist three transition classifiers c_i for $i \in \{good, bad, uns\}$ such that the following condition holds for any given state pair $\langle s, s' \rangle$: if $v(s, s')$ is "good" (respectively "bad", or "unsolvable") then $\langle s, s' \rangle$ is in c_{good} but not in c_{bad} or c_{uns} (respectively $\langle s, s' \rangle$ in c_{bad} but not in c_{good} or c_{uns}, or $\langle s, s' \rangle$ is in c_{uns} but not in c_{good} or c_{bad}).*

Proof. The proof is constructive. Consider a decision tree with root node v_1. For all paths $v_1, v_2, \ldots, v_k$ in the decision tree from the root node v_1 to a leaf node v_k we have a rule $r = C \mapsto E$ whose feature conditions C and feature effects E consist of all the feature conditions, respectively feature effects, along the path. If $v_k(s, s')$ is "good", "bad", or "unsolvable", then r is either in c_{good}, c_{bad}, or c_{uns} respectively. Any given state pair $\langle s, s' \rangle$ is compatible with a rule in at most one of the three transition classifiers because the feature conditions and effects on all inner nodes are mutually exclusive. Hence, it immediately follows that if $v_1(s, s')$ is "good", "bad", or "unsolvable", then $\langle s, s' \rangle$ is in c_{good} but not in c_{bad} or c_{uns} (respectively $\langle s, s' \rangle$ in c_{bad} but not in c_{good} or c_{uns}, or $\langle s, s' \rangle$ is in c_{uns} but not in c_{good} or c_{bad}). □

The translation allows us to view a decision tree as three transition classifiers and apply the syntax and semantics of general policies over features Φ. By establishing the correspondence between decision trees and general policies, this translation enables us to leverage existing general policy evaluation methods for efficient assessment of learned transition classifiers during search.

4 Combining Heuristics and Transition Classifiers

In this section, we describe methods to combine transition classifiers with heuristics to combine domain-general with problem-specific knowledge. Since transition classifiers are simple relations over state pairs and heuristics are functions of the state, we take a general perspective on the combination.

We have learned three classifiers c_{good}, c_{bad}, c_{uns} to identify good, bad, and unsolvable transitions. In the best case, when c_{good} is a perfect classifier, we can strictly follow it and reach a goal state. In most cases, however, the transition classifiers will not be perfect. To support the classifiers during search, we use the satisficing h^{FF} heuristic [16]. We next discuss several combinations of transition classifiers and h^{FF} with different search methods. The first one mainly trusts the transition classifiers and uses h^{FF} only as secondary guidance. The second uses h^{FF} as the primary heuristic and the classifiers as tie-breakers. The third and fourth methods give equal weight to the classifiers and h^{FF} by using them in a dual-queue approach. Finally, the fifth method uses c_{uns} to detect unsolvable states.

4.1 Policy Lookahead

The first method is a *policy lookahead*, introduced by Yoon et al. [32], which greedily follows the transitions compatible with c_{good}. In this approach, for every state s expanded during search, all successors s' of s are generated (in random order) and the state pairs $\langle s, s' \rangle$ are evaluated using c_{good}. If one of the pairs is compatible with c_{good}, the corresponding successor state is selected for the next expansion. All states expanded along a trajectory and their successors are added to an open list that is sorted by one or more heuristics. If following c_{good} fails before reaching a goal state, i.e., some state s does not have any successor s' such that $\langle s, s' \rangle$ is compatible with c_{good}, the next state is selected from the open list. With the open list as fallback, this method is complete even with an imperfect policy.

4.2 Tie-Breaking using Transition Classifiers

The second combination is an h^{FF}-based greedy best-first search (GBFS). We use the transition classifiers as tie-breakers in case multiple states have the same h^{FF} value. This is done by defining a heuristic h^{π} that evaluates a state s' according to the transition classifier that captures the state transition from its parent s. In particular, $h^{\pi}(s') = 0$ if $\langle s, s' \rangle$ is in c_{good}, $h^{\pi}(s') = 1$ if it is c_{bad}-compatible, and $h^{\pi}(s') = 2$ if it is in c_{uns}. This definition leads to prioritizing good transitions over bad ones, and bad transitions over unsolvable ones, among transitions to successor states with the same h^{FF} values.

4.3 Transition Classifiers for Preferring Operators

The concept of *preferred operators* is commonly used in satisficing heuristic search to focus the search for goal states on promising regions of the state space [22]. Usually, preferred operators are a byproduct of the heuristic evaluation: to estimate the goal distance of a state s, h^{FF} computes a relaxed plan and marks all operators that are part of the relaxed plan as preferred [16]. For brevity, we call states that are reached via preferred operators *preferred states* and the transitions they were reached on *preferred transitions*. Preferred states are maintained in a separate open list (also sorted by h^{FF}), and the search alternates between expanding a state from the main open list that contains all states and expanding a state from the list containing only preferred states.

Preferring c_{good}-Compatible Transitions. Our third combination of transition classifiers and heuristics marks the transitions that are compatible with c_{good} as *preferred*, and thereby biases search towards executing good transitions. Pruning all non-preferred states would render the search incomplete. Hence, using preferred operators in a dual-queue approach, which prioritizes preferred states but does not prune any non-preferred states, offers a good trade-off, as preferred states are more likely to be expanded early.

Discrepancy Search. The fourth combination uses the concept of *discrepancy* [14, 17] to compute a path-dependent state ranking based on transition classifiers. The rank $r(s')$ of a state s' is defined as the rank $r(s)$ of its predecessor s plus the value assigned to the transition from s to s', defined as above: good $\mapsto 0$, bad $\mapsto 1$, unsolvable $\mapsto 2$. So the rank of a state s' is the sum of all transition values on the first path found from the initial state to s'. The intuition behind this approach is that states with a low rank have a low discrepancy, that is, they were reached by paths that seldom deviated from c_{good}. We use r in a dual-queue approach by alternatingly expanding states from the open list ordered by h^{FF} and the one ordered by r.

4.4 Unsolvability Heuristic

We use c_{uns} to compute a heuristic h^{uns} for detecting unsolvable states. Here, states s' generated by unsolvable transitions are assigned a value of $h^{uns}(s') = 1$, all other states evaluate to 0. Employed in a tie-breaking open list, states marked as unsolvable by h^{uns} will only be considered once all other states have been expanded. The resulting heuristic is similar in spirit to the work by Ståhlberg et al. [28], who learned compact per-domain characterizations of unsolvable states in the form of description logic formulas. However, the work did not evaluate learned classifiers within search.

5 Experiments

We next describe our learning pipeline and show in detail how the learned transition classifiers are used during planning.

5.1 Common Setup

We use the benchmark set from the Learning Track of the International Planning Competition (IPC) 2023 [30], which consists of ten domains with two instance sets each: around 100 instances for training and 90 for testing. We reserve the test instances exclusively for our final experiment that evaluates different planner configurations. We allocate the 20 largest training instances per domain for validating our search configurations and use the remaining instances $\sim$80 for learning. For running the experiments, we use the Lab toolkit [25] on a compute cluster with Intel Xeon Gold CPUs and Ubuntu 20.04 LTS 64-bit. All code, benchmarks and experiment data are available online [19].

5.2 Learning Pipeline

We follow the IPC setup and use 1 CPU core, 24 hours, and 32 GiB for training per domain. For generating the description logic features, we use the DLPlan library [7], setting their maximum syntactic feature complexity to 10 and limiting the total number of Boolean and numeric features to 10,000. In total, we sample 5,000 states and 100,000 transitions from the state spaces that can be fully expanded within 60 seconds and have no more than 10,000,000 states. We reserve 80% of the instances as training data, and 20% as validation data. We learn decision trees using the CART algorithm as implemented in the Scikit-learn machine learning library [20]. During learning, two hyperparameters of a decision tree—a maximum depth in the set {10, 20, 30} and a maximum number of leaf nodes in the set {100, 200, 300}—are tuned using F1 score-based grid search with 5-fold cross-validation. After evaluating the models on the validation set, we merge both datasets and retrain the final model using the best hyperparameters obtained from the grid search.

5.3 Learning Results

Table 1 summarizes results obtained with our learning pipeline for the ten domains. We see that Miconic, Spanner and Sokoban are the only domains where the state spaces of almost all available instances can be fully expanded. Despite a high imbalance (between bad and good transitions, approximately 80%/20%) in the training data, the model learned for Miconic achieves the best results across all metrics on the validation data. In Spanner, there is also a pronounced imbalance (between bad, good and unsolvable transitions) in the training data: 0.39%/99.24%/0.37%. Consequently, the model demonstrates perfect precision and recall for good transitions but low precision and recall for bad and unsolvable transitions. In contrast, Sokoban (a PSPACE-complete domain) yields the worst F1 score among all domains. However, it achieves a high recall for good transitions, correctly identifying 90% of them, while its precision for these transitions is only 50%. Sokoban is also one of the four domains (along with Childsnack, Floortile, and Spanner) that include unsolvable transitions. However, the model struggles to learn anything useful for correctly classifying such transitions in the validation data. When examining the data, we observe that the features generated for Sokoban are not discriminative enough to distinguish between transitions. This leads to poor performance on both training and validation data.

Approximately half of the available instances can be expanded for Ferry and Rovers. Despite resulting in a very large tree, the model for Ferry achieves high precision and recall, which is also reflected in its F1 score. In contrast, a similarly sized model does not perform as well in Rovers. The data in Rovers is more balanced (52.77%/47.23%) compared to other domains; however, the model struggles to classify good transitions accurately. One limitation is that in Rovers, the generated features have a maximum complexity of 8 (due to the limit on the number of features), the lowest among all domains.

For Blocksworld and Childsnack, the models achieve high precision, recall, and F1 scores for all transition types.

Satellite, Transport and Floortile are the domains with the smallest number of expanded instances. In Satellite and Transport, the trees are large and almost identical in terms of depth and the number of leaf nodes. While it may seem that the F1 score for Satellite is high, a closer look at the precision for good transitions reveals that the model

Domain	$\|EI\|$	Φ_{DT}	Depth	Leaves	Comp	Pre$(-)$	Pre$(+)$	Pre(u)	Rec$(-)$	Rec$(+)$	Rec(u)	F1-train	F1-valid
Blocksworld	31	231	25	300	9	0.98	0.97	–	0.97	0.98	–	1.00	0.97
Childsnack	34	114	18	300	10	0.94	0.99	1.00	0.96	0.98	1.00	0.99	0.98
Ferry	38	70	19	200	10	0.98	0.97	–	0.99	0.94	–	0.99	0.97
Floortile	17	87	19	100	10	0.69	0.81	0.93	0.84	0.59	0.99	0.90	0.80
Miconic	79	34	11	52	10	1.00	1.00	–	1.00	0.99	–	1.00	1.00
Rovers	38	235	25	300	8	0.75	0.77	–	0.79	0.72	–	0.90	0.76
Satellite	23	220	21	300	10	0.98	0.83	–	0.97	0.87	–	0.96	0.91
Sokoban	72	19	10	52	10	0.50	0.49	0.34	0.10	0.90	0.05	0.32	0.30
Spanner	69	44	14	194	10	0.77	1.00	0.65	0.45	1.00	0.55	0.85	0.72
Transport	23	167	19	300	10	0.90	0.76	–	0.91	0.74	–	0.91	0.83

Table 1. Learning results across all domains. $|EI|$ is the number of fully expanded instances per domain, Φ_{DT} is the number of features used by the decision tree, *Depth* is the tree depth, *Leaves* is the number of leaf nodes, *Comp* is the maximum complexity of generated features. *Pre* is precision, *Rec* is recall, both on validation data, $(-)/(+)/(u)$ indicates good/bad/unsolvable transitions. For the domains without unsolvable transitions, we cannot report precision and recall.

is precise on only 83% of these transitions. The situation is even worse for Transport, where recall for good transitions is also low. For Floortile, the model has the smallest number of expanded instances, with only 38,000 transitions in total. This domain also exhibits the highest precision and recall for unsolvable transitions. However, for good and bad transitions, the metrics are worse, which results in a smaller overall F1 score.

5.4 Planning with Transition Classifiers

We use the 20 largest IPC training instances to validate our algorithms. The baseline configurations include three traditional heuristic planners: h^{FF} [16], $h^{\text{FF-pref}}$ (the variant that uses h^{FF} with preferred operators [22]), the first iteration of LAMA [23], and a state-of-the-art learning-based planner, WL-GOOSE with heuristic $h^{\text{WLF}}_{\text{GPR}}$ [4]. This variant of WL-GOOSE uses Gaussian Process Regression (GPR) [21] over features derived with the Weisfeiler-Leman algorithm [26] to learn domain-dependent heuristics, and is considered the state-of-the-art in classical planning for heuristic learning. $h^{\text{WLF}}_{\text{GPR}}$ is implemented on top of the Fast Downward planning system [15].

We evaluate the following search configurations:

- $h^{\langle \text{FF},\pi \rangle}$ denotes heuristic search with h^{FF} where h^{π} is used as a tie-breaker,
- $h^{\langle uns,\text{FF},\pi \rangle}$ denotes heuristic search with h^{uns} and h^{FF} where h^{π} is used as a tie-breaker,
- $Alt^{h^{\text{FF}}}_{c_{good}\text{-pref}}$ denotes alternating between h^{FF}-sorted states and c_{good}-preferred states,
- $Alt^{\langle h^{uns},h^{\text{FF}} \rangle}_{c_{good}\text{-pref}}$ denotes alternating between h^{FF}-sorted states enhanced with unsolvability and c_{good}-preferred states,
- $Alt^{h^{\text{FF}}}_{r}$ denotes alternating between h^{FF}-sorted states and r-ranked states,
- $Alt^{\langle h^{uns},h^{\text{FF}} \rangle}_{r}$ denotes alternating between h^{FF}-sorted states enhanced with unsolvability and r-ranked states,
- $L^{c_{good}}_{\text{bfs}}$ denotes a heuristic-free baseline that does policy lookahead in BFS manner, sorting states by the shortest distance from the initial state,
- $L^{c_{good}}_{\text{dfs}}$ denotes a heuristic-free baseline, doing policy lookahead in DFS manner and sorting states by the longest distance from the initial state,
- $L^{c_{good}}_{h^{\text{FF}}}$ denotes policy lookahead with h^{FF}-sorted states,
- $L^{c_{good}}_{\langle h^{uns},h^{\text{FF}} \rangle}$ denotes policy lookahead with h^{FF}-sorted states enhanced with unsolvability.

As done for the IPC Learning Track, we limit runtime and memory to 30 minutes and 8 GiB. All configurations are part of or were implemented by us within the Fast Downward planning system [15].

5.5 How to Use Transition Classifiers in Search?

Table 2 shows per-domain coverage results for our different algorithms (and the baselines). When analyzing plain classifier-first algorithms, $L^{c_{good}}_{\text{bfs}}$ and $L^{c_{good}}_{\text{dfs}}$, it becomes evident that using the learned knowledge alone is not sufficient. Although $L^{c_{good}}_{\text{dfs}}$ solves more tasks in total than some of the other configurations, it usually requires a significantly higher number of expansions. Adding a heuristic to the classifier-first configurations, $L^{c_{good}}_{h^{\text{FF}}}$ and $L^{c_{good}}_{\langle h^{uns},h^{\text{FF}} \rangle}$, improves the coverage in five domains: Blocksworld, Childsnack, Rovers, Spanner and Transport, compared to other algorithms.

The $h^{\langle \text{FF},\pi \rangle}$ approach relies too heavily on h^{FF}, and using transition classifiers as tie-breakers does not provide much benefit. Alternating between transition classifiers and heuristics in $Alt^{h^{\text{FF}}}_{c_{good}\text{-pref}}$ improves coverage in four domains, Floortile, Rovers, Satellite and Transport, compared to h^{FF}. However, this performance is still worse than $h^{\text{FF-pref}}$. The results are even worse for $Alt^{h^{\text{FF}}}_{r}$, where the coverage decreases in Miconic, one of the easiest domains.

To improve performance in domains with unsolvable states—Childsnack, Floortile, Spanner, and Sokoban—we extend all algorithms with the unsolvability heuristic. This results in improvements for Floortile for all algorithms. Although the coverage improves in Childsnack, we observe that the overall performance is still worse than in other domains. This can also be attributed to the quality of c_{good}, which we elaborate on further in the next section. In Spanner, we observe that the quality of c_{uns} is insufficient, as the coverage decreases between $L^{c_{good}}_{h^{\text{FF}}}$ and $L^{c_{good}}_{\langle h^{uns},h^{\text{FF}} \rangle}$. Despite this, for both of these algorithms the coverage remains higher than in other methods. In Sokoban, the coverage does not improve at all when compared to the baseline configurations. This is expected given the poor quality of the learned model.

In summary, our evaluation demonstrates that a classifier-based lookahead search using a heuristic for guidance and an unsolvability heuristic offers the best combination of the learned knowledge and the heuristic. Therefore, we proceed with $L^{c_{good}}_{\langle h^{uns},h^{\text{FF}} \rangle}$ as the best configuration and analyze it more closely on a per-domain basis in the next section.

5.6 Are Transition Classifiers Helpful?

When considering improvements to h^{FF}, $h^{\text{FF-pref}}$ usually offers a computationally cheap enhancement. Thus, we evaluate whether learning

Domain	Baselines				Heuristic-first		Alternating between heuristics and classifiers				Classifier-first			
	h^{FF}	$h^{\text{FF-pref}}$	LAMA	$h^{\text{WLF}}_{\text{GPR}}$	$h^{\langle \text{FF},\pi\rangle}$	$h^{\langle uns,\text{FF},\pi\rangle}$	$Alt^{h^{\text{FF}}}_{c_{good}\text{-pref}}$	$Alt^{\langle h^{uns},h^{\text{FF}}\rangle}_{c_{good}\text{-pref}}$	$Alt^{h^{\text{FF}}}_{r}$	$Alt^{\langle h^{uns},h^{\text{FF}}\rangle}_{r}$	$L^{c_{good}}_{\text{bfs}}$	$L^{c_{good}}_{\text{dfs}}$	$L^{c_{good}}_{h^{\text{FF}}}$	$L^{c_{good}}_{\langle h^{uns},h^{\text{FF}}\rangle}$
Blocksworld	28	31	60	**72**	30	30	28	28	26	26	18	36	39	39
Childsnack	22	33	**35**	31	25	26	20	23	15	20	22	29	33	34
Ferry	69	69	66	**76**	67	67	65	65	61	61	65	74	68	69
Floortile	11	15	11	2	12	31	22	32	22	30	10	7	17	**33**
Miconic	**90**	**90**	**90**	**90**	88	89	89	88	83	83	73	**90**	**90**	**90**
Rovers	32	57	**68**	37	31	31	38	38	30	29	17	45	47	47
Satellite	64	67	**89**	48	67	66	66	65	60	60	31	54	67	67
Sokoban	36	37	**40**	38	36	34	34	34	34	34	26	18	36	35
Spanner	30	30	30	73	30	30	30	30	30	30	**90**	68	56	49
Transport	38	58	**66**	28	45	44	52	53	37	36	33	62	57	57
Total (900)	420	487	**555**	495	431	448	444	456	398	409	385	483	510	520

Table 2. Number of solved tasks per domain and algorithm on the test set.

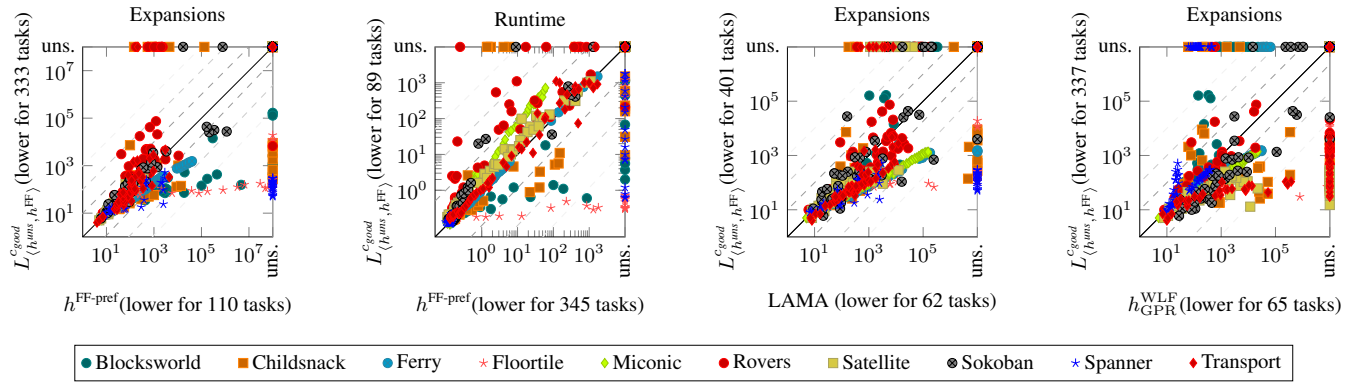

Figure 1. Number of state expansions and runtime of our best algorithm, $L^{c_{good}}_{\langle h^{uns},h^{\text{FF}}\rangle}$, compared to traditional ($h^{\text{FF-pref}}$, LAMA) and learning-based ($h^{\text{WLF}}_{\text{GPR}}$) planners.

domain knowledge in the form of transition classifiers is worth the extra effort. A comparison of coverage alone is insufficient here, which is why we also inspect the number of expansions and planner runtime, shown in Figure 1. Based on these metrics, we identify several groups of domains when comparing the performance of the best configuration $L^{c_{good}}_{\langle h^{uns},h^{\text{FF}}\rangle}$ to $h^{\text{FF-pref}}$.

Miconic, Satellite, and Transport. When comparing the coverage and the number of expansions of $h^{\text{FF-pref}}$ and $L^{c_{good}}_{\langle h^{uns},h^{\text{FF}}\rangle}$, there is no significant improvement from using transition classifiers. Although there is a clear improvement in coverage for Transport when combining transition classifiers with h^{FF}, $h^{\text{FF-pref}}$ solves more tasks by avoiding the time for classifier evaluation, as reflected in the planner runtime.

Ferry, Floortile, and Spanner. For Ferry, while we do not observe any increase in coverage, using transition classifiers helps to reduce the number of expansions as the task size increases. In Floortile, we observe a clear benefit from using the unsolvability-enhanced open list, as our configuration solves the highest number of tasks among all approaches. For Spanner, both coverage and the number of expansions improve significantly as the task size increases. However, the quality of the learned c_{uns} for Spanner is not sufficient to improve the performance of $L^{c_{good}}_{\langle h^{uns},h^{\text{FF}}\rangle}$ over other lookahead variants. A less accurate c_{uns} can misclassify good transitions as unsolvable. Consequently, these transitions are explored later in the search, leading to increased runtime and a reduced number of solved tasks.

Blocksworld and Childsnack. Both domains benefit from using transition classifiers. This is further supported by inspecting the planner runtime and the number of expansions for the instances solved by $h^{\text{FF-pref}}$. Despite the benefits of transition classifiers and strong learning performance, the overall coverage in these domains remains low. Several factors contribute to this behavior. First, the instances that can be fully expanded in these domains lack diversity in the number of objects. As a result, instances with the same number of objects but slightly different goal states appear in both training and validation sets. This leads to misleadingly good performance on the validation data. More importantly, we cannot properly evaluate the generalization of the learned models to larger instances. Another potential issue is the sampling of states. Although we assume that the uniform random sampling should provide us with a diverse set of transitions, the informativeness of state spaces, both across and within instances, can vary considerably. For example in Childsnack, when sampling transitions from a given instance, it is possible to sample symmetric transitions [9], which can reduce the model's ability to generalize to larger instances. Finally, the learned decision trees for both Childsnack and Blocksworld are large, which can lead to overfitting. We leave further investigation of these issues for future work.

Rovers and Sokoban. As discussed in Section 5.3, the learned classifiers for these domains demonstrate the weakest performance. Given this, no improvement is observed in either Rovers or Sokoban when compared to $h^{\text{FF-pref}}$; in fact, the number of expansions increases for some instances in Sokoban.

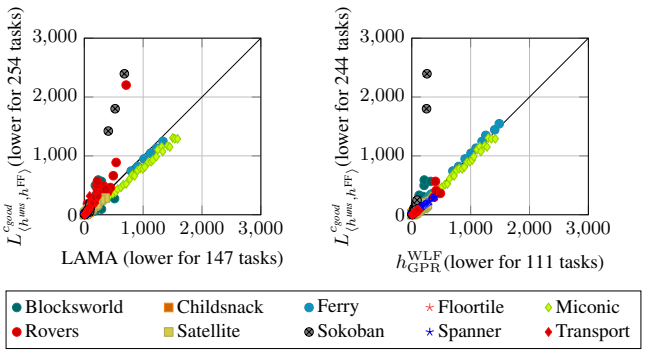

Figure 2. Plan-length comparison of our best algorithm, $L^{c_{good}}_{\langle h^{uns},h^{FF}\rangle}$, to LAMA (left) and h^{WLF}_{GPR} (right) on commonly solved instances.

5.7 Comparison to State of the Art

As mentioned in the previous section, the results demonstrate that the $L^{c_{good}}_{\langle h^{uns},h^{FF}\rangle}$ algorithm outperforms $h^{FF\text{-pref}}$ in five domains: Blocksworld, Childsnack, Ferry, Floortile, and Spanner. In our final analysis, we inspect how $L^{c_{good}}_{\langle h^{uns},h^{FF}\rangle}$ compares to two state-of-the-art methods: 1) the first iteration of LAMA, a traditional heuristic planner that has been one of the most prominent methods for satisficing classical planning since many years [23]; and 2) WL-GOOSE with heuristic h^{WLF}_{GPR}, the first learning-based planner that comes close to the performance of LAMA on several domains [4].

When looking at coverage, $L^{c_{good}}_{\langle h^{uns},h^{FF}\rangle}$ performs better than LAMA in three domains: Ferry, Floortile, and Spanner. When analyzing the number of expansions in Figure 1, we observe that despite the overall lower coverage, $L^{c_{good}}_{\langle h^{uns},h^{FF}\rangle}$ requires a lower number of expansions to solve tasks compared to LAMA. In Figure 2, we also report that $L^{c_{good}}_{\langle h^{uns},h^{FF}\rangle}$ tends to yield shorter plans than LAMA, except for a few outliers in Rovers and Sokoban. An important point here is that $L^{c_{good}}_{\langle h^{uns},h^{FF}\rangle}$ combines h^{FF} with transition classifiers. As future work, it would be interesting to explore the potential improvement gained by adding a landmark heuristic to the current configuration.

In contrast to our approach based on transition classifiers, h^{WLF}_{GPR} is a learned heuristic function. It utilizes only states visited by optimal plans on training instances to learn this heuristic. This enables the authors to extract optimal plans from IPC training instances using a state-of-the-art classical planner. However, it also means that obtained models leverage information from more training instances than our transition classifiers do. The coverage-based comparison with h^{WLF}_{GPR} reveals that $L^{c_{good}}_{\langle h^{uns},h^{FF}\rangle}$ performs better in five domains: Childsnack, Floortile, Rovers, Satellite, and Transport. The same trend is observed in the number of expansions shown in Figure 1. As previously discussed, for Childsnack, Floortile and Transport, transition classifiers strongly contribute to improved performance, while for Rovers and Satellite, h^{FF} plays a more significant role. This highlights the strength of our approach, where both types of knowledge reinforce each other. When comparing the number of expansions as seen in Figure 1, we observe that $L^{c_{good}}_{\langle h^{uns},h^{FF}\rangle}$ performs well even in domains where the coverage is lower, such as Ferry, Miconic, and Sokoban. Finally, Figure 2 shows that $L^{c_{good}}_{\langle h^{uns},h^{FF}\rangle}$ tends to produce shorter plans than h^{WLF}_{GPR}, with some outliers in Sokoban.

6 Conclusions

We present a novel and simple approach to learning transition classifiers using description logic features and decision trees. Including several approaches from the literature, we analyze different ways to combine such transition classifiers with traditional heuristic-search methods. Our analysis shows that, among the different combinations, we obtain the strongest algorithm by using the transition classifier for good transitions in a lookahead search, and a mix of learned and non-learned heuristics to provide further guidance. Our results highlight the value of unsolvable transition classifiers during search, which can significantly enhance overall performance.

Compared to state-of-the-art methods based on traditional methods, i.e., the LAMA planner, and learning-based methods, i.e., WL-GOOSE, we underline the strengths of our approach, which successfully combines learned domain knowledge with traditional heuristics. This leads to superior performance in domains where transition classifiers and heuristics effectively complement each other. Here, our approach balances the contributions of learned domain knowledge and traditional heuristics, achieving improvements in coverage and reducing the number of expansions in several domains.

For future work, we suggest studying state spaces of the instances to identify which transitions are more likely to be useful for learning transition classifiers. We also believe that the optimization of the decision trees should be further explored to avoid overfitting and improve the generalization. Additionally, exploring approaches that integrate landmark heuristics, successfully employed in LAMA, into the current framework could further improve performance.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[2] B. Bonet and H. Geffner. General policies, representations, and planning width. In K. Leyton-Brown and Mausam, editors, *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, pages 11764–11773. AAAI Press, 2021.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[4] D. Z. Chen, F. Trevizan, and S. Thiébaux. Return to tradition: Learning reliable heuristics with classical machine learning. In S. Bernardini and C. Muise, editors, *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, pages 68–76. AAAI Press, 2024.

[5] T. de la Rosa, S. Jiménez, R. Fuentetaja, and D. Borrajo. Scaling up heuristic planning with relational decision trees. *Journal of Artificial Intelligence Research*, 40:767–813, 2011.

[6] J. E. Doran and D. Michie. Experiments with the graph traverser program. *Proceedings of the Royal Society A*, 294:235–259, 1966.

[7] D. Drexler and J. Seipp. DLPlan: Description logics state features for planning. In *ICAPS 2023 System Demonstrations and Exhibits*, 2023.

[8] D. Drexler, J. Seipp, and H. Geffner. Expressing and exploiting subgoal structure in classical planning using sketches. *Journal of Artificial Intelligence Research*, 80:171–208, 2024.

[9] D. Drexler, S. Ståhlberg, B. Bonet, and H. Geffner. Symmetries and expressive requirements for learning general policies. In *Proceedings of the Twenty-First International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)*. IJCAI Organization, 2024.

[10] P. Ferber, L. Cohen, J. Seipp, and T. Keller. Learning and exploiting progress states in greedy best-first search. In L. De Raedt, editor, *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI 2022)*, pages 4740–4746. IJCAI, 2022.

[11] G. Francès, B. Bonet, and H. Geffner. Learning general planning policies from small examples without supervision. In K. Leyton-Brown and Mausam, editors, *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, pages 11801–11808. AAAI Press, 2021.

[12] M. Greco, Á. Torralba, J. Baier, and H. Palacios. Scaling up ML-based black-box planning with partial STRIPS models. In *ICAPS 2022 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*, 2022.

[13] M. Hao, F. Trevizan, S. Thiébaux, P. Ferber, and J. Hoffmann. Guiding GBFS through learned pairwise rankings. In K. Larson, editor, *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)*, pages 6724–6732. IJCAI, 2024.

[14] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995)*, pages 607–615. Morgan Kaufmann, 1995.

[15] M. Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[16] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14: 253–302, 2001.

[17] W. Karoui, M.-J. Huguet, P. Lopez, and W. Naanaa. YIELDS: A yet improved limited discrepancy search for CSPs. In P. Hentenryck and L. Wolsey, editors, *Proceedings of the 4th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2007)*, pages 99–111. Springer-Verlag, 2007.

[18] M. Krajňanský, J. Hoffmann, O. Buffet, and A. Fern. Learning pruning rules for heuristic search planning. In T. Schaub, G. Friedrich, and B. O'Sullivan, editors, *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, pages 483–488. IOS Press, 2014.

[19] F. Musayev, D. Drexler, D. Gnad, and J. Seipp. Code and data for the ECAI 2025 paper "Combining Heuristics and Transition Classifiers in Classical Planning". https://doi.org/10.5281/zenodo.16895349, 2025.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] C. E. Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[22] S. Richter and M. Helmert. Preferred operators and deferred evaluation in satisficing planning. In A. Gerevini, A. Howe, A. Cesta, and I. Refanidis, editors, *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 273–280. AAAI Press, 2009.

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

[24] J. Segovia-Aguas, S. Jiménez, and A. Jonsson. Generalized planning as heuristic search: A new planning search-space that leverages pointers over objects. *Artificial Intelligence*, 330:104097, 2024.

[25] J. Seipp, F. Pommerening, S. Sievers, and M. Helmert. Downward Lab. https://doi.org/10.5281/zenodo.790461, 2017.

[26] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[27] T. Silver, S. Dan, K. Srinivas, J. Tenenbaum, L. Pack Kaelbling, and M. Katz. Generalized planning in PDDL domains with pretrained large language models. In J. Dy and S. Natarajan, editors, *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*, pages 20256–20264. AAAI Press, 2024.

[28] S. Ståhlberg, G. Francès, and J. Seipp. Learning generalized unsolvability heuristics for classical planning. In Z.-H. Zhou, editor, *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*, pages 4175–4181. IJCAI, 2021.

[29] S. Ståhlberg, B. Bonet, and H. Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In S. Thiébaux and W. Yeoh, editors, *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, pages 629–637. AAAI Press, 2022.

[30] A. Taitler, R. Alford, J. Espasa, G. Behnke, D. Fišer, M. Gimelfarb, F. Pommerening, S. Sanner, E. Scala, D. Schreiber, J. Segovia-Aguas, and J. Seipp. The 2023 International Planning Competition. *AI Magazine*, 45(2):1–17, 2024.

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

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