Learning Heuristic Functions for HTN Planning

Daniel Höller

Saarland University, Saarland Informatics Campus, Saarbrücken, Germany.
hoeller@cs.uni-saarland.de

Abstract

During recent years, ML-based heuristic functions for automated planning have shown increasing performance. A main challenge is the level of generalization present in planning: techniques shall generalize at least across different instances of the same domain (which results in different sizes of learning input). To overcome the issue, a common approach is to use graph representations as input. While GNNs are a natural choice for learning, other methods have recently been favored because they show better runtime performance and need less training data. However, the work has been limited to non-hierarchical planning so far. We describe the first approach to learn heuristics for *hierarchical* planning. We extend the *Instance Learning Graph* – a graph structure used in non-hierarchical planning – to the new setting and show how to learn heuristic functions based on it. Since our heuristics are applicable to the lifted model, there is no need to ground it, thus we combine it with a novel lifted HTN planning system. Like recent systems in non-hierarchical planning, it grounds the search space explored so far, but not the entire model prior to search. Our evaluation shows that our approach is competitive with the lifted systems from the literature, though the ground systems reach higher coverage.

1 Introduction

During the past years, much work has been done towards exploiting methods from *Machine Learning* (ML) in automated planning: for selecting the best planning system for a given problem (Sievers et al. 2019; Ma et al. 2020; Ferber and Seipp 2022), for learning policies (Toyer et al. 2018; Groshev et al. 2018; Rivlin, Hazan, and Karpas 2020; Toyer et al. 2020; Silver et al. 2024), and for learning heuristic functions (Ferber, Helmert, and Hoffmann 2020; Shen, Trevizan, and Thiébaux 2020; Ferber et al. 2022; Heller et al. 2022; Ståhlberg, Bonet, and Geffner 2022, 2023; Chen, Thiébaux, and Trevizan 2024).

A main problem to overcome is the level of generalization present in planning: approaches need to generalize at least across different instances of the same domain, some approaches even generalize across domains, similar to symbolic planners in classical planning. Early approaches mitigated the issue by using neural network structures inspired by *Convolutional Neural Networks* (e.g. Toyer et al. 2018; 2020). Weight sharing between neurons enables scaling across instances of different size. More recent approaches use graph structures to represent the planning model, which are then used in combination with *Graph Neural Networks* (GNNs) (e.g. Shen, Trevizan, and Thiébaux 2020, Ståhlberg, Bonet, and Geffner 2022, or Chen, Thiébaux, and Trevizan 2024).

Recently, Chen, Trevizan, and Thiébaux (2024) have shown that a combination of features extracted from graphs using an algorithm based on the *Relational Weisfeiler-Leman (WL)* algorithm (Barceló et al. 2022) (originally introduced for testing whether graphs are isomorphic) and classical ML methods like *Support Vector Machines* (SVMs) and *Gaussian Processes* (GPs) yield good results in learning heuristics for classical planning, with advantages over GNN-based approaches regarding the amount of training data needed and computational runtime. The latter is especially important when learning heuristic functions, because these need to be computed very often during search.

However, all of this work has been done in *non-hierarchical* planning, mostly in classical planning and some extensions to numeric planning. In this paper, we describe the first approach to learning heuristics for hierarchical planning, more precisely for *Hierarchical Task Network (HTN)* planning, the most widely-used hierarchical planning formalism (Bercher, Alford, and Höller 2019). We base our approach on the one introduced by Chen, Trevizan, and Thiébaux (2024). We extend the graph structure to represent hierarchical models, extract input features using the WL method, and learn heuristic functions in a supervised learning setting using SVMs and GPs. We generalize over different instances of a planning domain. Second, we introduce a novel HTN planning system for lifted heuristic search. Like recent systems in classical planning (Corrêa et al. 2020), it grounds the search space explored so far, but does not ground the model prior to search.

Our empirical evaluation on the benchmarks of the track on *Totally Ordered HTN Planning* of the 2023 *International Planning Competition* (IPC) (Alford, Behnke, and Schreiber 2024) shows that (1) the quality of our learned heuristic is comparable with those used by recent HTN planning systems, and that (2) our overall approach is competitive with the current lifted HTN planning systems, while ground systems still show a better overall performance.

2 Background

Planning models are usually defined in a lifted input language like PDDL, or, in case of hierarchical planning, HDDL (Höller et al. 2020a). However, the currently best-performing HTN planning systems need a ground model and generate it before search (which might result in an exponential blowup of model size). While there are also challenges for the search (Corrêa et al. 2020), the main problem when planning directly on the lifted model are the (symbolic) heuristic functions (Lauer et al. 2021; Corrêa et al. 2022; Wichlacz, Höller, and Hoffmann 2022; Wichlacz et al. 2023). Since our learned heuristic will be computed on a lifted model, we also use a lifted *search*.

Next we introduce lifted HTN planning. Our formalism is loosely based on the one introduced in the HDDL standard.

A lifted HTN planning problem is defined based on a quantifier-free first-order predicate logic $L = (P, T, V, O)$ consisting of a set of type symbols T, a set of (typed) variables V, a set of predicate symbols P, and a set of typed constants (in the context of HDDL often referred to as *objects*) O. All these sets and also the ones introduced later in this section are finite. Each predicate symbol has an associated arity that defines its number of parameters out of V.

A HTN planning problem includes two types of tasks, *primitive* tasks (often referred to as *actions*) and *abstract* tasks (often called *compound*). Actions can be executed in the environment and cause state transition, abstract tasks cannot be executed directly, they are decomposed into other (primitive or abstract) tasks until only actions are left in a process similar to the derivation of a word from a formal grammar. We call the set of primitive and abstract tasks A and C, respectively. The structure tasks are maintained in in HTN planning is called *task network*. A task network forms a partially ordered multiset of tasks.

Definition 1 (Task Network)**.** *A task network tn over a set of task names X is a tuple $(ID, \prec, \alpha, B)$, where*

1. *ID is a set of task identifiers,*
2. *$\prec$ is a strict partial order over ID,*
3. *$\alpha : ID \to X$ maps task identifiers to task names,*
4. *B is a set of variable constraints. Each constraint can (a) bind two variables to be equal or non-equal, or (b) constrain a parameter to be/not to be of a certain type.*

The definition via identifiers (*ids*) ID makes it possible to include some task (e.g. $drive(city_A, city_B)$) multiple times in the network. α maps the ids to the actual tasks. $\prec$ defines the ordering in which tasks (or the tasks they are decomposed into) need to be present in the solution.

The ordering relation and the decomposition structure in general are often restricted in some way to make finding a solution (computationally) easier. A common restriction is to only allow for totally ordered networks, which has e.g. its own sub-track at the IPC.

Actions are triples $(name, prec, \mathit{eff})$. The name includes the actual name of the action, followed by a sequence of (typed) parameter variables, e.g. *drive(?a, ?b)*. *pre* is a first order formula over literals over P and *eff* is a conjunction over literals over P. We require all contained variables to be included in *name*. Given an action a, we will refer to the elements above as *name(a)*, $pre(a)$, and $\mathit{eff}(a)$. Further let $\mathit{eff}^+(a)$ and $\mathit{eff}^-(a)$ be the set of atoms contained in the effect as positive and negative literals, respectively.

The grammar-like decomposition structure is defined via abstract tasks and methods. Abstract tasks are names followed by a list of typed parameters. Methods form a rule on how an abstract task can be decomposed. A method m is a triple (c, tn, B) containing the name of an abstract (compound) task, a task network tn, and a set of variable constraints B that (co-)designate variables from c and tn.

Definition 2 (Planning Domain)**.** *An HTN planning domain is a tuple $\mathcal{D} = (L, A, C, M)$ of the underlying logic L, the set of primitive and abstract tasks A and C, and the set of (decomposition) methods M.*

All tasks in the methods need to be from A and C. The set of states S of the problem is the set of all subsets of ground predicates. We call an element (like a task network, a primitive/abstract task, or a predicate) *ground* when all its parameters have been bound to or replaced by constraints from O, e.g., a ground predicate is a predicate symbol from P followed by a sequence of constants (of the respective type) taken from O.

Definition 3 (Planning Problem)**.** *An HTN planning problem is a tuple $\mathcal{P} = (\mathcal{D}, s_I, tn_I, g)$ of a planning domain $\mathcal{D}$, an initial state s_I from S, the initial task network tn_I, and the goal description g, a formula over the predicates defined in the domain.*

A lifted planning problem forms a compact representation of a ground problem. The latter can be generated by systematically replacing variables by constants of their type. For details of the grounding process, we refer to Alford, Bercher, and Aha (2015). For an actual grounding system, we refer to Behnke et al. (2020). We define the semantics of HTN planning problems via the ground representation.

A ground action a is executable in a state $s \in S$ if and only if $s \models prec(a)$. The state transition function $\gamma : S \times A \to S$ is defined as follows. If an action a is applicable in a state s then $\gamma(s, a) = (s \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$, else $\gamma(s, a)$ is undefined. The application of a sequence of actions $\gamma^* : S \times A^* \to S$ is defined accordingly.

We call a (ground) task network $tn = (ID, \prec, \alpha)$ *executable* if and only if there is a sequence $i_1, \ldots, i_n$ of the ids $i_j \in ID$ with $|ID| = n$ in line with $\prec$ such that $\alpha(i_1), \ldots, \alpha(i_n)$ is executable.

Please notice that no constraints are given in the ground task network. This is because the (in-)equality relations will be incorporated in the grounding process, and will thus hold in the resulting model.

Next we need to define how an abstract task c in a given task network tn can be decomposed. To do so, we need a method $m = (c, tn')$. Intuitively, decomposition is then done by removing c from tn, adding the task network tn' from m to it, and maintaining the ordering constraints by introducing all ordering constraints to the new tasks that have been in the network for c.

Formally, a task network $tn_1 = (ID_1, \prec_1, \alpha_1)$ with a task id $t \in ID_1$ with $\alpha_1(t) = c$ can be decomposed by a method

$m = (c, tn)$. Let $tn' = (ID', \prec', \alpha')$ be a copy of tn that uses ids not contained in ID_1. The task network tn_2 resulting from the decomposition is defined as follows:

$$tn_2 = \big((ID_1 \setminus \{t\}) \cup ID', \prec' \cup \prec_D, (\alpha_1 \setminus \{t \mapsto c\}) \cup \alpha'\big)$$
$$\prec_D = \{(t_1, t_2) \mid (t_1, t) \in \prec_1, t_2 \in ID'\} \cup$$
$$\{(t_1, t_2) \mid (t, t_2) \in \prec_1, t_1 \in ID'\} \cup$$
$$\{(t_1, t_2) \mid (t_1, t_2) \in \prec_1, t_1 \neq t \wedge t_2 \neq t\}$$

When a task network tn can be decomposed into a task network tn' by decomposing a task t using a method m, we will write $tn \xrightarrow[t,m]{} tn'$.

Definition 4 (Solution)**.** *Let $\mathcal{P} = (\mathcal{D}, s_I, tn_I, g)$ be a planning problem, $\mathcal{D} = (L, A, C, M)$ the underlying domain, and $tn_S = (ID_S, \prec_S, \alpha_S)$ a (ground) task network. tn_S is a solution to the planning problem if and only if the following conditions hold.*

- *There is a sequence of decompositions from tn_I to a task network $tn = (ID, \prec, \alpha)$ such that $ID = ID_S$, $\prec \subseteq \prec_S$, and $\alpha = \alpha_S$.*
- *tn_S only contains primitive tasks and has an executable linearization (i.e., it is executable) that, when executed, leads to a state s with $s \models g$.*

3 Learning HTN Heuristic Functions

Next we describe our learning process, starting with the graph structure we use, followed by the feature extraction.

3.1 A Graph Representation for HTN Planning

Our graph structure builds on the *Instance Learning Graph* for classical planning, the following definition is the one given by Chen, Trevizan, and Thiébaux (2024, Def. 3.1) adapted to our notation.

Definition 5 (Instance Learning Graph (ILG))**.** *Let $L = (P, T, V, O)$ be a logic, $\mathcal{D} = (L, A, \cdot, \cdot)$ a planning domain, and $\mathcal{P} = (\mathcal{D}, s_I, \cdot, g)$ a planning problem*[1]*. The ILG is defined as a graph $G = (V, E, c, l)$ where V is the set of vertices, E the set of edges, c a function coloring the vertices, and l a function labeling the edges.*

- $V = O \cup s_I \cup g$
- $E = \bigcup_{p=P(o_1,\dots,o_n)\in(s_I\cup g)} \{(p, o_1), \dots, (p, o_n)\}$
- $c : V \to (\{ag, ap, ug\} \times P) \cup \{ob\}$ *with*

$$u \mapsto \begin{cases} ob, & \text{if } u \in O \\ (ag, P) & \text{if } u = P(o_1, \dots, o_n) \in s_I \cap g \\ (ap, P) & \text{if } u = P(o_1, \dots, o_n) \in s_I \setminus g \\ (ug, P) & \text{if } u = P(o_1, \dots, o_n) \in g \setminus s_I \end{cases}$$

- $l : E \to \mathbb{N}$ *with* $(p, o_i) \mapsto i$

The graph contains nodes for the objects as well as the atoms contained in the initial state and the goal of the planning model. The nodes are colored based on whether they are contained only in the goal (ug unachieved goal), in both (ag achieved goal) or only in s_I (ap achieved predicate). Atoms are connected by edges with the objects in their parameter list. Edges are label with their position in this list.

[1]Since this definition was given for classical planning, we do not need all elements and omitted the unnecessary ones.

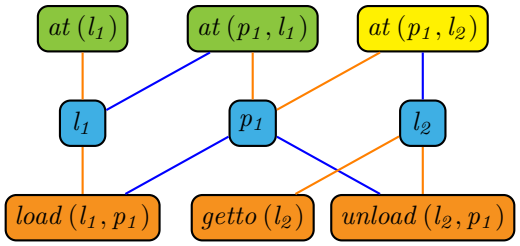

Figure 1: Example for an HILG.

Interestingly, the graph encoding does not include any information about the actions in the domain/the actual transition system. This needs to be learned implicitly by the ML-approach applied to the graph, resulting in a compact representation and (at least to us) elegant approach not relying on much graph tailoring towards the benchmark set.

We want to extend the graph in the same, simplistic way to the setting of HTN planning. Further, we see from heuristics successfully used in the IPC'23 (Olz, Höller, and Bercher 2023) that incorporating only the tasks contained in the currently task network seems to be enough information to extract good heuristic values in many domains (i.e., without incorporating e.g. ordering information). Thus we define our graph representation as follows:

Definition 6 (HTN Instance Learning Graph (HILG))**.** *Let $L = (P, T, V, O)$ be a logic, $\mathcal{D} = (L, A, C, M)$ a planning domain, and $\mathcal{P} = (\mathcal{D}, s_I, tn_I, g)$ a planning problem with $tn_I = (ID, \prec, \alpha)$. The HILG is defined as a graph $G = (V, E, c, l)$ where V is the set of vertices, E the set of edges, c a function coloring the vertices, and l a function labeling the edges.*

- $V = O \cup s_I \cup g \cup TS$, *with* $TS = \{\!\{\alpha(t) \mid t \in ID\}\!\}$
- $E = \bigcup_{p=P(o_1,\dots,o_n)\in(s_I\cup g)} \{(p, o_1), \dots, (p, o_n)\}$
  $\cup \bigcup_{p=N(o_1,\dots,o_n)\in TS} \{(p, o_1), \dots, (p, o_n)\}$
- $c : V \to (\{ag, ap, ug\} \times P) \cup \{ob\} \cup (\{t\} \times (C \cup A))$ *with*

$$u \mapsto \begin{cases} ob, & \text{if } u \in O \\ (ag, P) & \text{if } u = P(o_1, \dots, o_n) \in s_I \cap g \\ (ap, P) & \text{if } u = P(o_1, \dots, o_n) \in s_I \setminus g \\ (ug, P) & \text{if } u = P(o_1, \dots, o_n) \in g \setminus s_I \\ (t, N) & \text{if } u = N(o_1, \dots, o_n) \in TS \end{cases}$$

- $l : E \to \mathbb{N}$ *with* $(p, o_i) \mapsto i$

We added the set of tasks in the current task network as nodes to the graph, and connected them in a similar way to the objects of the model. Be aware that a task might be in the network multiple times, so that we used a multiset (which we denoted by $\{\!\{\}\!\}$).

Figure 1 shows an example for an HILG for a simple transport problem where a single truck needs to deliver packages. The green nodes at the top left encode that both the truck and package p_1 are currently at location l_1. The

goal location for the package is l_2, as encoded by the yellow goal node. The nodes in the middle represent the objects in the model. The nodes at the bottom represent the tasks from the current task network: the actions $load\,(l_1, p_1)$ and $unload\,(l_2, p_1)$, as well as the abstract task $getto\,(l_2)$ in the middle. Every node that represents an atom or a task is connected to the nodes represeting the objects in its parameter list. The color of the edges represent the respective position in the parameter list.

3.2 Training

Next we describe the training process; we build our learning framework on the *WLPlan* framework (Chen 2024). We implement a domain-specific, supervised learning setup.

Benchmark Set. We use the benchmark set of the track on *Totally Ordered HTN (TOHTN)* planning of the IPC'23. While our approach is not limited to this setting, it is the most widely-used HTN problem class, comes with a large benchmark set and more planning systems to compete with than the general track. We had to exclude the *Monroe* domains, which encode an *plan recognition as planning* approach (Höller et al. 2018a). This leads to a very diverse set of tasks included in the solutions that needs to be learned based on a much larger training set to even include all tasks in the training data.

Training Data. For each domain, we generated optimal solutions for the up to 10 smallest instances using an optimal configuration of the PANDA planning system (Höller 2023), the winner of the respective track of the IPC. However, we were not able to generate all 10 solutions for all domains, in *Minecraft (pl)* and *Snake* we train on only 3 instances, in *Assembly*, *Multiarm-BW*, and *BW-HPDDL* on 5, in *Factories* on 6, *Satellite* on 7, *Rover* and *SharpSAT* on 8, and *Transport* on 9. For *Freecell*, PANDA did not find optimal plans at all and we trained on 10 (potentially) suboptimality ones. We also evaluated weather it is helpful to include sub-optimal plans when we were not able to find all 10 optimal solutions, but this did not reflect in coverage.

Next we go through the solutions. For each intermediate step, we store pairs $(hilg(tn, s), h^*(tn, s))$ that contain the graph representation of the intermediate task network and state, together with its optimal goal distance.

The number of graphs was between a minimum of 140 and a maximum of 5245 with a median of 582 (depending on the number of solutions as well as the solution length).

Feature Generation. Based on the graph representation we generate the learning input. We used the *Relational Weisfeiler-Leman (WL)* method (Barceló et al. 2022) from the WLPlan framework. The algorithm was originally introduced to test graph isomorphism. Based on their initial coloring, it updates the colors of nodes by incorporating the colors of their neighbors for several iterations. The result can be represented as histogram of the colors, which has been used as input features for machine learning. The method needs a hyperparameter that specifies the number of iterations of the algorithm, which we set to $1 - 3$.

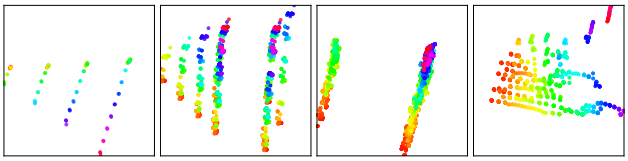

Figure 2: PCA-based visualization of the embeddings of the task networks. Colors indicate the heuristic values.

As proposed by Chen, we visualize the feature vectors by using *Principal Component Analysis (PCA)* to project them to two-dimensional space. The result can be seen in Figure 2, where colors represent the corresponding heuristic values. As can be seen, in several domains there seems to be a quite good structure in the features, indicating that they could enable the prediction of the respective heuristic values. We included *Freecell* as negative example with less structure.

Training. Like Chen, Trevizan, and Thiébaux, we train two kinds of regression models based on the resulting features: support vector regression with the dot product kernel (SVR) and Gaussian process regression (Rasmussen and Williams 2006) with the dot product kernel (GPR).

4 The LION Planning System

Our search is based on the progression algorithm introduced by Höller et al. (2020b). A progression search only processes tasks from a task network that have no predecessor in their ordering relation. This makes it possible to maintain the current state during search, which enables the effective application of heuristics. While this search was used by several systems at the IPC'23, those systems required a preground model because they used heuristic functions based on a ground model. Since our heuristic is applicable to the lifted model, we do not need to ground the model and implemented a lifted variant, which is outlined in Algorithm 1. We call our system LION, **li**fted progressi**on**.

Search nodes are triples containing the current state $s \in S$, the current task network, and the sequence of actions applied so far. Initially (Line 1), the fringe contains a single search node consisting of the initial state s_I, the initial task network tn_I, and the empty sequence ε of actions. Wlog, we make the following assumption:

Assumption 1. *All search nodes initially contained in the fringe are ground.*

If the assumption does not hold, we compile the (then lifted) initial task network tn_I away by introducing a new task t^n and a method decomposing t^n into tn_I, and set the new initial task network to the network only containing t^n.

The main loop of the algorithm starts by removing a node $n = (s, (ID, \prec, \alpha), \pi)$ from the fringe (Line 3). If n is a solution, it is returned. A node is a solution if and only if $ID = \emptyset$ and $s \models g$. From the contained task network, the unconstrained tasks $U = \{t \mid t \in ID, \neg\exists(t', t) \in \prec\}$ are split into abstract U_C and primitive U_A tasks. When

```
1  fringe ← {(s_I, tn_I, ε)}
2  while fringe ≠ ∅ do
3      n ← fringe.pop()
4      if isGoal(n) then return n'
5      (U_C, U_A) ← n.unconstrainedNodes()
6      if U_C = ∅ then
7          for t ∈ U_A do
8              if isApplicable(t) then
9                  n' ← n.apply(t)
10                 fringe.add(n')
11     else
12         t ← selectAbstractTask(U_C)
13         for m ∈ {m | m ∈ M decomposes t} do
14             GM ← n.ground(t, m)
15             for gm ∈ GM do
16                 n' ← decompose(t, gm)
17                 fringe.add(n')
```

Algorithm 1: Lifted progression algorithm.

there are unconstrained abstract tasks, one of them is selected[2] (Line 12), and all applicable methods are systematically grounded and applied (Lines 13, 14).

Assume that the task network in n is ground (which is shown later in Theorem 1), then we can apply decomposition as defined by Höller et al. (2020b, Def. 4):

Definition 7. *Given a search node $(s, (ID, \prec, \alpha), \pi)$, an unconstrained task id $t \in ID$ with $\alpha(t) = c$, and a method (c, tn), the search node n' that results from decomposing t is defined as $n' = (s, tn', \pi)$ with $tn \xrightarrow[t,m]{} tn'$.*

Since we ground methods, the following trivially holds:

Lemma 1. *Given that the task network in n is ground, all task networks contained in nodes inserted into the fringe in Line 17 are ground.*

In case all tasks in U are primitive, we remove them in turn and apply them (starting in Line 6). Assume n is ground, then we can apply progression as defined by Höller et al. (2020b, Def. 3):

Definition 8. *Given a search node $(s, (ID, \prec, \alpha), \pi)$ and a task $t \in U_A$ that is mapped to an applicable action $a = \alpha(t)$, the search node resulting from progressing t is defined as $n' = (\gamma(a, s), tn', \pi \circ a)$ with $tn' = (ID \setminus \{t\}, \prec \setminus \{(t, t') \mid t' \in ID\}, \alpha \setminus \{t \mapsto a\})$.*

In this case, we need to apply all actions in turn, because by the application of an action, we commit do its position in the solution and do not know which is the right one.

Lemma 2. *Given that n is ground, all task networks in nodes inserted into the fringe in Line 10 are ground.*

Now we know that we start with a ground fringe (Assumption 1) and that both operations leading to new nodes added to the fringe return ground search nodes when applied to a ground search node. Thus the following holds:

[2]This is no non-deterministic choice, we can choose randomly.

Theorem 1. *All search nodes contained in the fringe during search are ground.*

This means that, like recent systems in classical planning, we apply a lazy-grounding approach, i.e., while the model is not pre-grounded before search, all search nodes that have been explored so far are fully ground. Further we do not need to adapt the definitions of goal test, unconstrained tasks, action applicability and application, and decomposition known from the ground case (and given above).

The only change concerns method decomposition (Lines 13 and 14). While in the ground setting, the model contains a set of ground methods applicable to a given abstract task, we now have a set of lifted methods that we systematically ground during search.

We call a grounding function complete if and only if it returns all ground methods that may be part of a solution. Assuming a complete grounding function, the following holds:

Theorem 2. *Given a complete grounding function is used, Algorithm 1 is sound and complete.*

Proof. Whether the algorithm is sound depends on the definition of action application, decomposition, and goal test, which are unchanged compared to the ground case. Since we assume that the grounding function is complete, completeness also follows from the ground case. □

5 Empirical Evaluation

We evaluate our approach against two groups of planning systems from the 2020 and 2023 IPCs: systems that ground the HTN model before search, and those that solve the problems based on the lifted model. Conceptually, our system belongs to the second group.

Ground Systems. PANDA (Höller et al. 2018b, 2020b), which is a heuristic search-based system, in its original configuration and in the version that won the IPC'23 (Olz, Höller, and Bercher 2023). The TOAD (Höller 2021, 2024) system and the HTN2SAS (Behnke et al. 2022) system, which are different translations to classical planning.

Lifted Systems. The winner and runner-up of the IPC'20, HyperTensioN (Magnaguagno, Meneguzzi, and de Silva 2021) and LiloTane (Schreiber 2021a,b). HyperTensioN uses problem transformations and a lifted search; LiloTane a translation to propositional logic that does not rely on a preground model. Further we included SIADEX (Fernández-Olivares, Vellido, and Castillo 2021), which is also based on a lifted search, and Lifted Tree Path (Quenard, Pellier, and Fiorino 2023), a SAT-based planner building on LiloTane.

All experiments ran on Intel Xeon E5-2650 CPUs with 2.30 GHz (one core per job, no GPU). Coverage tests ran with a time limit of 30 minutes and 8 GB memory. Training ran on the same machines (one core, no GPU), but with more time and memory; its resource consumption is discussed in Section 5.2.

5.1 LION Configurations

Table 1 compares the coverage of different LION configurations. In general, *Greedy Best First Search* (GBFS) performs best, which is in line with results of other systems

	search:	gbfs						weighted A*						A*					
	model:	svr			gpr			svr			gpr			svr			gpr		
	WL iterations:	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
Assembly	30	7	5	6	5	5	**8**	4	5	4	4	4	4	3	3	3	3	3	3
Barman	20	**20**	**20**	19	**20**	**20**	19	16	10	10	16	10	10	7	7	7	7	7	7
Blocksworld-GTOHP	30	28	28	28	**29**	28	**29**	28	28	28	**29**	28	**29**	24	26	24	25	26	24
Blocksworld-HPDDL	30	20	**21**	**21**	20	20	**21**	20	20	20	20	20	20	7	7	7	7	6	7
Depots	30	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	16	16	17	16	16	17
Factories	20	5	**8**	**8**	4	**8**	**8**	4	5	5	4	5	5	4	4	4	4	4	4
Freecell-Learned	60	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Hiking	30	11	10	10	**16**	15	11	5	8	9	7	8	10	5	5	5	6	6	6
Lamps	30	16	17	17	17	**18**	**18**	16	17	17	17	16	16	16	16	16	16	16	16
Logistics-Learned	80	38	37	38	35	35	**39**	35	31	31	28	28	31	28	28	28	28	28	28
Minecraft-Player	20	**5**	**5**	**5**	4	4	4	**5**	4	4	4	4	4	4	4	4	4	4	4
Minecraft-Regular	59	44	**46**	45	36	35	35	41	42	41	35	35	34	35	33	34	35	34	35
Multiarm-BW	74	26	**74**	**74**	9	71	**74**	72	61	59	**74**	59	61	23	22	20	19	21	19
Robot	20	**20**	**20**	**20**	19	**20**	**20**	12	13	12	12	12	12	11	11	11	11	11	11
Rover	30	**30**	**30**	27	**30**	**30**	28	20	18	18	21	18	18	8	8	8	8	8	8
Satellite	20	**15**	**15**	14	**15**	**15**	14	14	12	13	14	12	13	7	8	8	7	8	8
SharpSAT	21	**13**	12	11	10	12	11	9	9	9	9	9	9	9	9	9	9	9	9
Snake	27	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Towers	20	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13
Transport	40	8	13	16	–	15	14	23	**29**	**29**	7	28	28	15	14	14	10	14	15
Woodworking	30	**7**	**7**	6	**7**	**7**	6	**7**	**7**	**7**	**7**	**7**	6	6	6	6	6	6	6
	721	350	**405**	402	313	395	396	368	356	353	345	340	347	243	242	240	236	242	242

Table 1: Coverage of different LION configurations. Domains in which all systems reached the same coverage are given in gray, the best coverage per domain is given bold. In this table, all instances are included (also those used for training).

Domain	#inst	*lion*		$panda_{ipc}$		$panda_{rc}^{gbfs}$		*toad*		*htn2sas*	
Assembly	30	5	0.17	**30**	**1.00**	**30**	**1.00**	**30**	**1.00**	**30**	**1.00**
Barman	20	**20**	**1.00**	16	0.80	15	0.75	15	0.75	14	0.70
BW-GTOHP	30	28	0.93	29	0.97	**30**	**1.00**	20	0.67	26	0.87
BW-HPDDL	30	21	0.70	**30**	**1.00**	27	0.90	21	0.70	20	0.67
Depots	30	22	0.73	22	0.73	22	0.73	**24**	**0.80**	22	0.73
Factories	20	8	0.40	**10**	**0.50**	8	0.40	5	0.25	6	0.30
Freecell	60	–	–	11	0.18	**14**	**0.23**	–	–	–	–
Hiking	30	10	0.33	**25**	**0.83**	**25**	**0.83**	12	0.40	22	0.73
Lamps	30	17	0.57	**18**	**0.60**	16	0.53	**18**	**0.60**	16	0.53
Logistics	80	37	0.46	**80**	**1.00**	47	0.59	47	0.59	72	0.90
Minec.-Pl	20	**5**	**0.25**	1	0.05	1	0.05	1	0.05	1	0.05
Minec.-Reg	59	**46**	**0.78**	41	0.69	41	0.69	39	0.66	41	0.69
Multia.-BW	74	**74**	**1.00**	**74**	**1.00**	**74**	**1.00**	**74**	**1.00**	71	0.96
Robot	20	20	1.00	20	1.00	20	1.00	20	1.00	20	1.00
Rover	30	**30**	**1.00**	27	0.90	29	0.97	9	0.30	17	0.57
Satellite	20	15	0.75	16	0.80	16	0.80	**19**	**0.95**	17	0.85
SharpSAT	21	**12**	**0.57**	10	0.48	10	0.48	1	0.05	9	0.43
Snake	27	2	0.07	**4**	**0.15**	3	0.11	**4**	**0.15**	3	0.11
Towers	20	13	0.65	13	0.65	13	0.65	12	0.60	**15**	**0.75**
Transport	40	13	0.33	22	0.55	25	0.62	8	0.20	**31**	**0.78**
Woodw.	30	7	0.23	28	0.93	23	0.77	**30**	**1.00**	24	0.80
	721	405	11.93	**527**	**14.82**	489	14.11	409	11.71	477	13.42

#inst	*lion*		$panda_{ipc}$		$panda_{rc}^{gbfs}$		*toad*		*htn2sas*	
25	–	–	**25**	**1.00**	**25**	**1.00**	**25**	**1.00**	**25**	**1.00**
10	**10**	**1.00**	6	0.60	5	0.50	5	0.50	4	0.40
20	18	0.90	19	0.95	**20**	**1.00**	10	0.50	16	0.80
25	16	0.64	**25**	**1.00**	22	0.88	16	0.64	15	0.60
20	12	0.60	12	0.60	12	0.60	**14**	**0.70**	12	0.60
14	3	0.21	**4**	**0.29**	2	0.14	–	–	–	–
50	–	–	2	0.04	**4**	**0.08**	–	–	–	–
20	2	0.10	**15**	**0.75**	**15**	**0.75**	5	0.25	12	0.60
20	7	0.35	**8**	**0.40**	6	0.30	**8**	**0.40**	6	0.30
70	27	0.39	**70**	**1.00**	37	0.53	37	0.53	62	0.89
17	**2**	**0.12**	–	–	–	–	–	–	–	–
49	**36**	**0.73**	31	0.63	31	0.63	29	0.59	31	0.63
69	**69**	**1.00**	**69**	**1.00**	**69**	**1.00**	**69**	**1.00**	66	0.96
10	10	1.00	10	1.00	10	1.00	10	1.00	10	1.00
22	**22**	**1.00**	19	0.86	21	0.95	1	0.05	9	0.41
13	8	0.62	9	0.69	9	0.69	**12**	**0.92**	10	0.77
12	**3**	**0.25**	1	0.08	1	0.08	–	–	–	–
24	–	–	**1**	**0.04**	–	–	**1**	**0.04**	–	–
10	3	0.30	3	0.30	3	0.30	2	0.20	**5**	**0.50**
31	7	0.23	13	0.42	16	0.52	3	0.10	**22**	**0.71**
20	–	–	18	0.90	13	0.65	**20**	**1.00**	14	0.70
560	264	9.75	**369**	**12.95**	330	12.00	268	9.46	328	11.29

Table 2: Coverage of LION as well as several ground systems. On the full benchmark set (left), and on the benchmark set without training instances (right). For each domain and system, the absolute as well as the normalized coverage are given.

Domain	#inst	*lion*		*hypert.*		*LiloTane*		*ltp*		*siadex*	
Assembly	30	**5**	**0.17**	3	0.10	**5**	**0.17**	2	0.07	–	–
Barman	20	**20**	**1.00**	**20**	**1.00**	16	0.80	17	0.85	**20**	**1.00**
BW-GTOHP	30	**28**	**0.93**	15	0.50	21	0.70	22	0.73	11	0.37
BW-HPDDL	30	21	0.70	**30**	**1.00**	1	0.03	–	–	–	–
Depots	30	22	0.73	**24**	**0.80**	22	0.73	22	0.73	22	0.73
Factories	20	**8**	**0.40**	3	0.15	4	0.20	4	0.20	–	–
Freecell	60	–	–	3	0.05	**7**	**0.12**	–	–	–	–
Hiking	30	10	0.33	**25**	**0.83**	20	0.67	15	0.50	–	–
Lamps	30	17	0.57	1	0.03	**19**	**0.63**	–	–	–	–
Logistics	80	37	0.46	22	0.28	**41**	**0.51**	1	0.01	–	–
Minec.-Pl	20	**5**	**0.25**	**5**	**0.25**	1	0.05	1	0.05	3	0.15
Minec.-Reg	59	46	0.78	**56**	**0.95**	22	0.37	28	0.47	33	0.56
Multia.-BW	74	**74**	**1.00**	8	0.11	3	0.04	–	–	1	0.01
Robot	20	**20**	**1.00**	**20**	**1.00**	11	0.55	5	0.25	–	–
Rover	30	**30**	**1.00**	**30**	**1.00**	19	0.63	19	0.63	**30**	**1.00**
Satellite	20	15	0.75	**20**	**1.00**	14	0.70	12	0.60	–	–
SharpSAT	21	12	0.57	**15**	**0.71**	8	0.38	4	0.19	1	0.05
Snake	27	2	0.07	**3**	**0.11**	–	–	1	0.04	–	–
Towers	20	13	0.65	**20**	**1.00**	9	0.45	4	0.20	10	0.50
Transport	40	13	0.33	**40**	**1.00**	34	0.85	32	0.80	1	0.03
Woodw.	30	7	0.23	7	0.23	**30**	**1.00**	**30**	**1.00**	3	0.10
	721	**405**	11.93	370	**12.11**	307	9.59	219	7.33	135	4.50

#inst	*lion*		*hypert.*		*LiloTane*		*ltp*		*siadex*	
25	–	–	–	–	–	–	–	–	–	–
10	**10**	**1.00**	**10**	**1.00**	6	0.60	7	0.70	**10**	**1.00**
20	**18**	**0.90**	5	0.25	11	0.55	12	0.60	2	0.10
25	16	0.64	**25**	**1.00**	–	–	–	–	–	–
20	12	0.60	**14**	**0.70**	12	0.60	12	0.60	13	0.65
14	**3**	**0.21**	–	–	–	–	–	–	–	–
50	–	–	**3**	**0.06**	–	–	–	–	–	–
20	2	0.10	**15**	**0.75**	10	0.50	8	0.40	–	–
20	7	0.35	–	–	**9**	**0.45**	–	–	–	–
70	27	0.39	12	0.17	**31**	**0.44**	–	–	–	–
17	**2**	**0.12**	**2**	**0.12**	–	–	–	–	–	–
49	36	0.73	**46**	**0.94**	12	0.24	18	0.37	23	0.47
69	**69**	**1.00**	4	0.06	1	0.01	–	–	1	0.01
10	**10**	**1.00**	**10**	**1.00**	1	0.10	–	–	–	–
22	**22**	**1.00**	**22**	**1.00**	11	0.50	11	0.50	**22**	**1.00**
13	8	0.62	**13**	**1.00**	7	0.54	5	0.38	–	–
12	3	0.25	**6**	**0.50**	–	–	–	–	–	–
24	–	–	–	–	–	–	–	–	–	–
10	3	0.30	**10**	**1.00**	–	–	–	–	–	–
31	7	0.23	**31**	**1.00**	25	0.81	23	0.74	–	–
20	–	–	–	–	**20**	**1.00**	**20**	**1.00**	–	–
560	**264**	9.75	237	**10.76**	164	6.73	120	5.48	72	3.28

Table 3: Coverage of LION as well as several lifted systems. On the full benchmark set (left), and on the benchmark set without training instances (right). For each domain and system, the absolute as well as the normalized coverage are given.

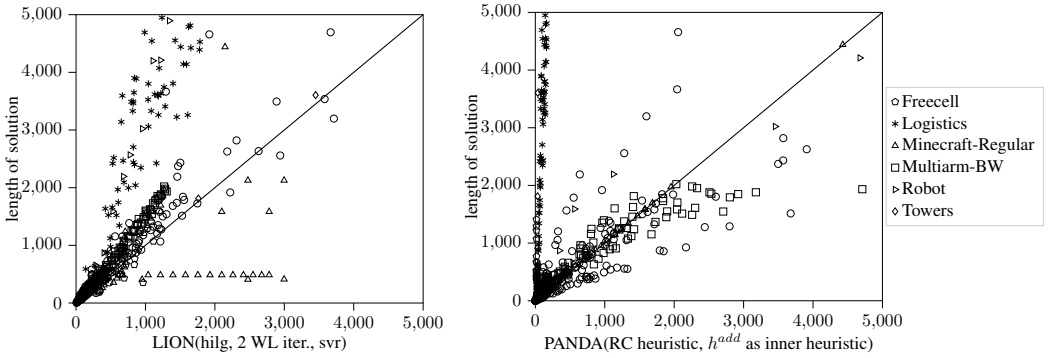

Figure 3: Length of shortest solution found by any system/configuration compared to the initial heuristic value.

like PANDA. An exception is the *Transport* domain, where *Weighted* A^* (WA*) search (we used a weight of 2) performs better. This domain has a left-recursive decomposition structure (see Höller 2024), which leads to cycles of decompositions in which all unconstrained tasks are abstract. It seems that LION is less good in escaping such structures. WA* seems to mitigate this problem. Overall, A* search reaches the lowest coverage.

Configurations using SVR perform better than those using GPR, though there are exceptions like the *Assembly*, *Lamps*, *Logistics* and especially the *Hiking* domain. Configurations with 2 WL iterations reach the highest coverage, though there are exceptions where either more seem to be helpful (e.g. *Assembly*) or only one is needed (*SharpSAT*).

In the following, we use LION with GBFS, 2 WL iterations, and SVR model.

5.2 Resource Consumption During Training

For the used configuration (2 WL iterations, SVR model), the training process took between a minimum of 5.3 seconds and a maximum of 19 minutes and 36.5 seconds, with a median of 16.3 seconds. Memory usage was between 180 MB and 64.7 GB with a median of 574 MB.

5.3 Comparison to State of the Art

Next we investigate the quality of the learned heuristics. We compare it with the RC heuristic of the PANDA system, which won the respective track in the IPC'23. We computed the heuristic values of both heuristics on all initial states. We compare them with the best (i.e., shorted) solution found by any system or configuration in our evaluation. We plot the heuristic value against the best solution found (see Figure 3). All points being on the diagonal would mean that the estimate was perfect, points below the diagonal mean that solution length was overestimated, points above mean they are underestimated. We highlighted the more interesting domains, all others are represented by dots (○). In some domains, our heuristic is closer to solution length (e.g. in *Towers* and especially in *Logistics*). However, in others (especially *Minecraft Reg.*), the RC heuristic is closer.

Table 2 gives the coverage for LION as well as several ground systems from the literature. On the left, you see the results for the entire benchmark set (including the training instances), on the right, you see the results without training instances. The best ground systems have the highest coverage. We assume this is due to the presence of dead ends in the benchmarks, which can be detected by techniques like the RC heuristics. We have no dead end detection and thus cannot prune those leafs during search. However, there are also domains where LION reaches the highest coverage like *Barman* the two *Minecraft* domains, or *SharpSAT*.

Table 3 compares LION with several lifted systems. It performs similar to the best lifted system, HyperTensioN, which is the winner of the IPC'20. Compared to the lifted systems, LION performs especially well in *Blocksworld-GTOHP*, *Factories*, and *Multiarm-Blocksworld*. Interestingly, there are some domains (e.g. *Lamps*, *Logistics*) where it performs more similar to the SAT-based LiloTane system than the search-based HyperTensioN.

Overall, we conclude that our learned heuristic is similar informed as the RC heuristic used by the winner of the last IPC, and that our overall system is on par with the best lifted HTN planning systems. However, it outperforms both the grounded as well as the lifted systems in certain domains, adding value to the overall planner portfolio.

6 Conclusion

While learned heuristic functions have gained performance in classical planning during the last years, there are no such approaches in hierarchical planning yet. In this paper, we introduced the first approach towards this end. We introduced a graph structure to represent planning models, extracted WL-based feature vectors, and learned heuristic functions in a supervised learning setting. Further, we introduced a lifted variant of the progression search algorithm underlying the winner of the IPC'23. Our combined system does not yet reach the overall performance of the best ground systems, but it is competitive to the best lifted systems like the winner and runner-up of the IPC'20; and it outperforms both the ground and the lifted systems in certain domains.

Acknowledgements. This work was funded by the European Union's Horizon 2020 research and innovation programme, grant agreement No 101070149, project TUPLES.

References

Alford, R.; Behnke, G.; and Schreiber, D., eds. 2024. *Proceedings of the Hierarchical Task Network (HTN) Track of the 11th International Planning Competition (IPC): Planner and Domain Abstracts*.

Alford, R.; Bercher, P.; and Aha, D. W. 2015. Tight Bounds for HTN Planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 7–15. AAAI Press.

Barceló, P.; Galkin, M.; Morris, C.; and Orth, M. A. R. 2022. Weisfeiler and Leman Go Relational. In *Proceedings of the First Conference on Learning on Graphs (LoG)*, 46. PMLR.

Behnke, G.; Höller, D.; Schmid, A.; Bercher, P.; and Biundo, S. 2020. On Succinct Groundings of HTN Planning Problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, 9775–9784. AAAI Press.

Behnke, G.; Pollitt, F.; Höller, D.; Bercher, P.; and Alford, R. 2022. Making Translations to Classical Planning Competitive with Other HTN Planners. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*, 9687–9697. AAAI Press.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 6267–6275. IJCAI organization.

Chen, D. Z. 2024. WLPlan: Relational Features for Symbolic Planning. arXiv:2411.00577.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. W. 2024. Learning Domain-Independent Heuristics for Grounded and Lifted Planning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, 20078–20086. AAAI Press.

Chen, D. Z.; Trevizan, F. W.; and Thiébaux, S. 2024. Return to Tradition: Learning Reliable Heuristics with Classical Machine Learning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 68–76. AAAI Press.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation Using Query Optimization Techniques. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, 80–89. AAAI Press.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2022. The FF Heuristic for Lifted Classical Planning. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*, 9716–9723. AAAI Press.

Ferber, P.; Geißer, F.; Trevizan, F. W.; Helmert, M.; and Hoffmann, J. 2022. Neural Network Heuristic Functions for Classical Planning: Bootstrapping and Comparison to Other Methods. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*, 583–587. AAAI Press.

Ferber, P.; Helmert, M.; and Hoffmann, J. 2020. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI)*, 2346–2353. IOS Press.

Ferber, P.; and Seipp, J. 2022. Explainable Planner Selection for Classical Planning. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*, 9741–9749. AAAI Press.

Fernández-Olivares, J.; Vellido, I.; and Castillo, L. 2021. Addressing HTN Planning with Blind Depth First Search. In *Proceedings of the 10th International Planning Competition: Planner and Domain Abstracts – Hierarchical Task Network (HTN) Planning Track (IPC)*, 1–4.

Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2018. Learning Generalized Reactive Policies Using Deep Neural Networks. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS)*, 408–416. AAAI Press.

Heller, D.; Ferber, P.; Bitterwolf, J.; Hein, M.; and Hoffmann, J. 2022. Neural Network Heuristic Functions: Taking Confidence into Account. In *Proceedings of the 15th International Symposium on Combinatorial Search (SoCS)*, 223–228. AAAI Press.

Höller, D. 2021. Translating Totally Ordered HTN Planning Problems to Classical Planning Problems Using Regular Approximation of Context-Free Languages. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS)*, 159–167. AAAI Press.

Höller, D. 2023. The PANDA λ System for HTN Planning in the 2023 IPC. In *Proceedings of the Hierarchical Task Network (HTN) Track of the 11th International Planning Competition: Planner and Domain Abstracts (IPC)*.

Höller, D. 2024. The TOAD System for Totally Ordered HTN Planning. *Journal of Artificial Intelligence Research (JAIR)*, 80: 613–663.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2018a. Plan and Goal Recognition as HTN Planning. In *Proceedings of the 30th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 466–473. IEEE.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020a. HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, 9883–9891. AAAI Press.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2018b. A Generic Method to Guide HTN Progression Search with Classical Heuristics. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS)*, 114–122. AAAI Press.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2020b. HTN Planning as Heuristic Progression Search. *Journal of Artificial Intelligence Research (JAIR)*, 67: 835–880.

Lauer, P.; Torralba, Á.; Fiser, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, 4119–4126. IJCAI organization.

Ma, T.; Ferber, P.; Huo, S.; Chen, J.; and Katz, M. 2020. Online Planner Selection with Graph Neural Networks and Adaptive Scheduling. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, 5077–5084. AAAI Press.

Magnaguagno, M. C.; Meneguzzi, F.; and de Silva, L. 2021. HyperTensioN: A three-stage compiler for planning. In *Proceedings of the 10th International Planning Competition: Planner and Domain Abstracts – Hierarchical Task Network (HTN) Planning Track (IPC)*, 5–8.

Olz, C.; Höller, D.; and Bercher, P. 2023. The PANDA Dealer System for Totally Ordered HTN Planning in the 2023 IPC. In *Proceedings of the Hierarchical Task Network (HTN) Track of the 11th International Planning Competition: Planner and Domain Abstracts (IPC)*.

Quenard, G.; Pellier, D.; and Fiorino, H. 2023. LTP: Lifted Tree Path. In *Proceedings of the Hierarchical Task Network (HTN) Track of the 11th International Planning Competition: Planner and Domain Abstracts (IPC)*.

Rasmussen, C. E.; and Williams, C. K. I. 2006. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press. ISBN 026218253X.

Rivlin, O.; Hazan, T.; and Karpas, E. 2020. Generalized Planning With Deep Reinforcement Learning. *CoRR*, abs/2005.02305.

Schreiber, D. 2021a. Lifted Logic for Task Networks: TOHTN Planner Lilotane in the IPC 2020. In *Proceedings of the 10th International Planning Competition: Planner and Domain Abstracts – Hierarchical Task Network (HTN) Planning Track (IPC)*, 9–12.

Schreiber, D. 2021b. Lilotane: A Lifted SAT-based Approach to Hierarchical Planning. *Journal of Artificial Intelligence Research (JAIR)*, 70: 1117–1181.

Shen, W.; Trevizan, F. W.; and Thiébaux, S. 2020. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, 574–584. AAAI Press.

Sievers, S.; Katz, M.; Sohrabi, S.; Samulowitz, H.; and Ferber, P. 2019. Deep Learning for Cost-Optimal Planning: Task-Dependent Planner Selection. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 7715–7723. AAAI Press.

Silver, T.; Dan, S.; Srinivas, K.; Tenenbaum, J. B.; Kaelbling, L. P.; and Katz, M. 2024. Generalized Planning in PDDL Domains with Pretrained Large Language Models. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, 20256–20264. AAAI Press.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, 629–637. AAAI Press.

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

Toyer, S.; Thiébaux, S.; Trevizan, F. W.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *Journal of Artificial Intelligence Research (JAIR)*, 68: 1–68.

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

Wichlacz, J.; Höller, D.; Fiser, D.; and Hoffmann, J. 2023. A Landmark-Cut Heuristic for Lifted Optimal Planning. In *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI)*, 2623–2630. IOS Press.

Wichlacz, J.; Höller, D.; and Hoffmann, J. 2022. Landmark Heuristics for Lifted Classical Planning. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI)*, 4665–4671. IJCAI organization.