An Admissible Neuro-Symbolic Landmark Cut Heuristic – Preliminary Results

Submission 6

Abstract

Much work has been done towards using machine learning to solve classical planning problems, e.g. by learning policies or heuristics. However, so far, these methods do not come with formal guarantees on solution *optimality*. In this paper, we introduce a neuro-symbolic variant of the admissible *Landmark Cut* (LM-cut) heuristic. The computation of LM-cut starts with selecting one precondition per action using the *precondition choice function* (*pcf*), then a graph is constructed following rules that guarantee that every cut is a landmark, which is then used for the actual heuristic computation. While the *pcf* impacts the *quality* of the heuristic, it does not impact *admissibility*. At the same time, computing the *perfect pcf* is hard, and it is approximated in practice. We present preliminary results showing that the *pcf* can be learned with Reinforcement Learning instead. We compare the heuristic values computed with our *pcf* with the original approximation and show that they are higher in several domains, which is advantageous since both heuristics are admissible.

1 Introduction

Much work has been done towards solving classical planning problems using methods from machine learning (ML). Approaches can be divided into three main categories:

1. Using ML to select the most promising planner for a given problem (see e.g. Sievers et al. (2019), Ma et al. (2020), or Ferber and Seipp (2022)),
2. to learn a policy, i.e., a function mapping the current state to an action to take, (see e.g. Toyer et al. (2018), Groshev et al. (2018), Ståhlberg, Bonet, and Geffner (2022; 2023), or Silver et al. (2024)), and
3. to learn a heuristic function, i.e., a function mapping the current state to its goal distance, which is commonly used to steer a systematic search towards state regions that contain a goal state (see e.g. Ferber, Helmert, and Hoffmann (2020), Shen, Trevizan, and Thiébaux (2020), Ferber et al. (2022), Chen, Thiébaux, and Trevizan (2024), or Chen, Trevizan, and Thiébaux (2024)).

We present work in the setting of *optimal planning*, i.e., finding solutions with the guarantee that there is no solution with lower costs/number of actions. So far, the only work combining planning and ML in this setting is from category 1, *planner selection* (see e.g. Sievers et al. (2019) or Ma et al. (2020)). We present the first provably admissible heuristic function. In the symbolic setting, optimal planning is a well-studied field and has its own track in the *International Planning Competition* (Taitler et al. 2024).

A standard method is to combine A^* search (Hart, Nilsson, and Raphael 1968) with an *admissible* heuristic, i.e., one that never overestimates the goal distance. A state-of-the-art example is the *Landmark Cut* heuristic (LM-cut). It approximates h^+ (the optimal delete-relaxed solution) by incrementally computing landmarks for a planning problem via cuts in a graph representing a relaxed version of it. Landmarks are then disjunctions of actions such that one of them must be contained in every solution. Cuts are computed by first selecting one precondition per action using the *precondition choice function* (*pcf*). Based on this selection, the *justification graph* is constructed in a way that guarantees that all cuts are landmarks. After selecting a cut, the planning model is modified and the process is re-started. Finding the optimal *pcf* is a hard problem, and LM-cut approximates it based on the h^{max} heuristic (Bonet and Geffner 2001) instead. This has been motivated by a proof argument (see Helmert and Domshlak (2009)), but empirical results have shown that this is also a good choice in practice. However, it is known that the overall approach can be combined with other *pcfs* (Bonet and Helmert 2010), and there has been work on improving tie-breaking (Lauer and Fickert 2020).

We introduce a neuro-symbolic LM-cut heuristic that replaces the commonly-used h^{max}-based *pcf* with a learned one. The key observation is that, while the *pcf* impacts the *quality* of the heuristic, it does not impact *admissibility*. So the resulting heuristic still guarantees optimality. We first show that finding the optimal *pcf* is NP-complete. Then we train Graph Neural Networks (GNNs) to compute the *pcf* based on a graph representing all possible justification graphs. Since we are computing an admissible heuristic, we know that a higher heuristic value is a stronger approximation of the theoretical limit h^+. With this information, we train our GNN using Reinforcement Learning.

In our empirical evaluation, we compare the values of our heuristic to those of the original LM-cut heuristic and the theoretical limit of h^+. Our preliminary results show that our heuristic values are significantly higher than those from the original LM-cut in several domains. At this stage, we do not present results on actual search performance.

2 Background

2.1 Classical Planning

A classical planning problem p is a tuple $p = (L, A, s_0, G)$, where L is a set of propositional state features describing the environment. All sets in the definition are finite. A state $s \subseteq L$ is described by a subset of these features, where a state feature l is *true* if it is contained in the state ($l \in s$) and it is *false* otherwise. $s_0 \in L$ is the initial state, and $G \subseteq L$ is the goal condition. A is a set of actions. An action $a \in A$ comes with a set of preconditions $prec(a) \subseteq L$, a set of add effects $add(a) \subseteq L$, a set of delete effects $del(a) \subseteq L$, and a cost $c(a) \in \mathbb{N}_0$.

An action a is *applicable* to a state s, if and only if $prec(a) \subseteq s$ holds. If an applicable action a is applied to a state s, the state resulting from the application is defined by the function $\gamma : A \times 2^L \to 2^L$, where $\gamma(a, s) = (s \setminus del(a)) \cup add(a)$. The solution to a planning problem is a sequence $\pi = a_1, a_2, \ldots, a_n$ such that (1) a_i is applicable in state s_{i-1}, where s_i for $i > 0$ is defined as $s_i = \gamma(a_i, s_{i-1})$, and (2) $s_n \subseteq G$. States that include the state features in G are called *goal states*. By abuse of notation, we define the costs of a solution $\pi = a_1, a_2, \ldots, a_n$ as $c(\pi) = \sum_i c(a_i)$. We call a solution π *optimal* if there is no solution π' with $c(\pi') < c(\pi)$.

i-g Normal Form. For some proofs it is helpful to assume the planning problem to have a certain form called *i-g Normal Form* (igNF) (Helmert and Röger 2023). A classical planning problem $p = (L, A, s_0, G)$ is in *i-g Normal Form* if and only if there are two dedicated state features $\boldsymbol{i}$ and $\boldsymbol{g}$, such that $s_0 = \{\boldsymbol{i}\}$ and $G = \{\boldsymbol{g}\}$. Further, every action has at least one precondition. Obviously, every classical planning problem can be compiled in that form.

2.2 Heuristics in Classical Planning

Heuristic search is a predominant approach for finding optimal solutions in planning. A heuristic is a function $h : 2^L \to \mathbb{R}_0^+ \cup \{\infty\}$ that maps every state to a number, which can be interpreted as the distance from that state to the nearest goal state. The actual (minimal) goal distance of a state s is called the *perfect heuristic* and denoted $h^*(s)$.

Definition 1 (Safety)**.** *A heuristic h is called* safe *if and only if $h(s) = \infty$ only for states that are unsolved, i.e., for which there does not exist an action sequence to reach a solution.*

In practice, this means that a state s can be pruned when $h(s) = \infty$ without rendering the search incomplete.

Definition 2 (Admissibility)**.** *A heuristic h is called* admissible *if and only if $h(s) \leq h^*(s)$ for all states s.*

Combined with a search algorithm like A^*, admissible heuristics can be used to find optimal solutions.

When we have two admissible heuristics, the result of adding them up will – in general – not be admissible anymore. However, there is a way to combine such heuristics by adding them up that results in an admissible result (Katz and Domshlak 2010) called *cost partitioning*. The way LM-cut computes the costs of a cut forms a cost partitioning.

Definition 3 (Cost partitioning)**.** *Let $p = (L, A, s_0, G)$ be a planning problem. A cost partitioning is a sequence of functions $c_i : A \to \mathbb{R}_0^+$ such that for all $a \in A$ $(\sum_i c_i(a)) \leq c(a)$.*

When we compute i heuristics, each with the i^{th} cost function of a cost partitioning, we can add up the result and get an admissible overall heuristic. Intuitively, the costs of an action are "split" between the heuristic functions.

The computation of LM-cut is done by incrementally generating landmarks. Landmarks have originally been introduced in the context of problem decomposition (Hoffmann, Porteous, and Sebastia 2004). However, there is a variety of work based on landmarks to compute heuristics to guide a search (see e.g. the LAMA system (Richter and Westphal 2010), that combines it with delete-relaxation heuristics). A landmark is something that needs to be contained in every solution to a planning problem. Depending on the context, this might be a state feature that needs to be achieved during the execution of every solution, or an action that needs to be contained in every solution. In the context of LM-cut, we will encounter *disjunctive action landmarks*.

Definition 4 (Disjunctive Action Landmarks)**.** *Given a planning problem $p = (L, A, s_0, G)$, a* Disjunctive Action Landmark *is a set $D \subseteq A$ such that at least one action $a \in D$ is contained in every solution.*

Whenever we refer in the following to a landmark, we mean a disjunctive action landmark.

3 The LM-Cut Heuristic

In this section, we first discuss the original LM-cut heuristic, then we introduce a generic variant of the algorithm not relying on h^{max} and show its admissibility. Further we show that finding the best *pcf*, i.e., the one maximizing the heuristic value of the overall heuristic, is NP-complete.

3.1 The Original LM-Cut Heuristic

The original LM-cut heuristic as introduced by Helmert and Domshlak (2009) is given in Figure 1. We define it on a planning problem $p = (L, A, s_0, G)$, but it can be computed the same way on every search node, of course. To simplify the definition, we assume the problem to be in igNF. In a first step, h^{max} is computed. If it is 0 or ∞, it is returned as the heuristic value of $h^{lm\text{-}cut}$. If it is not, the actual computation of $h^{lm\text{-}cut}$ is started by initializing the return value to 0 (line 2). In the following, the costs of the actions are updated. We assume to start every $h^{lm\text{-}cut}$ computation with a fresh copy of the original costs, but that we are able to modify it within the scope of the heuristic function. To indicate this (possible) modification, we subscripted the h^{max} function with the costs c. Further, $h^{lm\text{-}cut}$ uses the h^{max} values of individual state features. Therefore its signature is now $h_c^{max} : 2^L \times L \to \mathbb{N}_0 \cup \{\infty\}$ (see e.g. line 3 in the algorithm).

While the h_c^{max} heuristic is non-zero for the overall problem, cut landmarks are generated in the following way. First, one precondition is selected for each action[1] by the *Precon-*

[1] assuming igNF, every action has at least one precondition

```
1  if h^max(s_0) ∈ {0, ∞} then return h^max(s_0)
2  h ← 0
3  while h^max_c(s, g) > 0 do
       // h^max-based precondition choice function
4      ∀a ∈ A : pcf(a) ← max arg_{l∈prec(a)} h^max_c(s, l)
       // define justification graph
5      J = (V, E) with V = L and
6      E = {(p, a, e) | ∃a ∈ A, pcf(a) = p, e ∈ add(a)}
       // determine goal zone
7      V_g = {v_0 ∈ V | ∃v_0 v_1 ... v_n, v_n = g
8            (v_{i−1}, a, v_i) ∈ E, c(a) = 0, for 1 ≤ i ≤ n}
       // compute cut landmark
9      cut = {a | (v, a, w), v ∉ V_g, w ∈ V_g,
10           v reachable from i without crossing V_g}
11     cut costs ← arg min_{a∈cut} c(a)
       // update h and costs
12     h ← h + cut costs
13     for a ∈ cut do c(a) ← c(a) − cut costs
14 return h
```

Figure 1: Original LM-cut algorithm

```
1  h ← 0
2  while true do
       // arbitrary precondition choice function
3      ∀a ∈ A : pcf(a) ← l such that l ∈ prec(a)
       // define justification graph
4      J = (V, E) with V = L and
5      E = {(p, a, e) | ∃a ∈ A, pcf(a) = p, e ∈ add(a)}
       // determine goal zone
6      V_g = {v_0 ∈ V | ∃v_0 v_1 ... v_n, v_n = g
7            (v_{i−1}, a, v_i) ∈ E, c(a) = 0, for 1 ≤ i ≤ n}
9      if no path from s_0 to g then return ∞
10     if i ∈ V_g then break
       // compute cut landmark
11     cut = {a | (v, a, w), v ∉ V_g, w ∈ V_g,
12           v reachable from i without crossing V_g}
13     cut costs ← arg min_{a∈cut} c(a)
       // update h and costs
14     h ← h + cut costs
15     for a ∈ cut do c(a) ← c(a) − cut costs
16 return h
```

Figure 2: Generic LM-cut algorithm

dition Choice Function (*pcf*). This is done by selecting one of the preconditions that determine the h^{max} value of the overall action (this choice might be ambiguous, see Lauer and Fickert (2020) for a study on tie-breaking strategies). Be aware that this is a relaxation: every solution for the original problem is a solution for the relaxed problem. This gives an intuition why landmarks computed on this problem are landmarks for the original problem.

Then the *justification graph* is built (line 5-6). The facts from the problem form its nodes. For each add effect of each action one edge is introduced. It connects the precondition selected by the *pcf* with the fact added and is labeled by the action name. The *goal-zone* (line 6-7) contains all nodes from which $\boldsymbol{g}$ is reachable via a path in the justification graph only containing edges labeled with zero-cost actions (using the updated action costs). Then a *cut* through the graph is made that separates $\boldsymbol{i}$ and $\boldsymbol{g}$ (line 9-10). Every such cut forms a landmark for the planning problem. LM-cut uses the one along the border of the goal-zone. Be aware that the condition given in line 10 is an optimization that creates a *smaller* landmark (see Lauer and Fickert (2020) for an empirical perspective on this optimization). It is not essential for the landmark property, because every superset of a landmark is also a landmark. Since at least one of the contained actions needs to be included in every solution, each landmark adds the costs of the cheapest action included to the heuristic value (line 11-12). Then, the costs of all actions included in the cut are decreased by the value of the cut. Consider an action that is contained in two landmarks: then its second occurrence will only be included with its reduced costs, i.e., this way a cost partitioning is created. We refer to the costs that are left for some action in an iteration as its *residual costs*.

It has been shown that the resulting heuristic is admissible (Helmert and Domshlak 2009).

3.2 A Generic LM-Cut Heuristic

The original LM-cut algorithm was designed as a proof argument for showing dominance relations between heuristic families (Helmert and Domshlak 2009). Therefore its design makes use of the h^{max} heuristic several times in the algorithm and the proofs of its properties. However, it is known that the overall approach can be combined with other *pcf*s (Bonet and Helmert 2010). Since we want an overall heuristic independent of h^{max}, we now present a generic variant of the algorithm.

In the following example, we show that using a *pcf* based on h^{max} is not always a good choice.

Example 1. *Let $p = (L, A, s_0, G)$ be a planning problem with:*

- $L = \{1, \ldots, n, z, g\}$
- $s_0 = \emptyset$, $G = \{g\}$
- $A = \{a_i \mid i \in 1 \ldots n\} \cup \{add_g\}$
- $prec(a_i) \mapsto \emptyset$, $add(a_i) \mapsto \{i, z\}$, $del(a_i) \mapsto \emptyset$
- $prec(add_g) \mapsto \{z\}$, $add(add_g) \mapsto \{g\}$, $del(add_g) \mapsto \emptyset$
- $c(a_i) = 1$, $c(add_g) = 0$

In this problem, all facts have an h^{max} value of 1. When we select z as precondition of add_g, we end up with a single cut of cost 1. If we iteratively choose state features from $\{1, \ldots, n\}$, we end up with an overall heuristic value of n.

The generic algorithm is given in Figure 2. In the original algorithm, LM-cut is computed in the initial step and LM-cut returns ∞ when h^{max} does. We replace this statement with a criterion based on graph reachability. Since it is based on the justification graph, we will come to it later on (line 9).

After initializing the heuristic value to 0, the algorithm starts with the main loop, which has no stop criterion in its condition. Then a precondition choice function is defined as a function mapping actions to one of their preconditions. We assume the problem to be in igNF, be aware that spe-

cial treatment is necessary when there are actions without preconditions. Formally, a *pcf* is defined as follows.

Definition 5 (Precondition Choice Function)**.** *Let $p = (L, A, s_0, G)$ be a planning problem in igNF. A* Precondition Choice Function *(*pcf*) is a function $f : A \to L$ such that, when $f(a) = l$, then $l \in prec(a)$.*

Next the *justification graph* and the *goal-zone* are defined as before. In line 9, the first stop condition is reached, returning ∞ when the goal state feature $\boldsymbol{g}$ is not reachable from $\boldsymbol{i}$. Below in Thm. 1 we show this does not render the heuristic unsafe, but before we show the theoretical properties, we want to go through the rest of the algorithm. The next stop condition breaks the loop when $\boldsymbol{i}$ is part of the goal-zone. The cut is defined as the subset of actions that span from the non-goal-zone of the graph to the goal-zone (line 11-12). As before, the costs of the cut are defined as the costs of the minimum cost action contained in it, and the residual costs of the actions in the cut are decreased by this value before entering the next loop of the algorithm.

Next we show *safety*, *termination*, and *admissibility* of the algorithm.

Theorem 1 (Safety)**.** *Let $p = (L, A, s_0, G)$ be a planning problem and g the justification graph as defined in line 5-6 in Alg. 2 for some* pcf*, when there is no path from the initial state to the goal-zone in the justification graph, then p is unsolvable.*

Proof. When we go back from the goal-zone, following edges in a backward manner, we are going from an effect of an action to one of its preconditions (by edge definition). We know that the goal is necessary to achieve (by the solution definition in planning), from there we go back the effect/precondition relations. When there is no path from the initial state to the goal-zone in a justification graph, then – on every path – there exists a state feature that is required to achieve an effect that is necessary for solving the planning problem, but, since there is no connection, it is not achieved by any action. Thus the problem is unsolvable, and it is safe to return ∞. □

To show termination, we need the following proposition.

Proposition 1 (Cut Costs)**.** *The costs of cuts produced by the algorithm are always greater than* 0*.*

Proof. A cut has 0-costs if and only if it includes an edge belonging to an action of costs 0. Assume a cut produced by the algorithm includes such an action. Then, there must be such an edge *between* the goal-zone and the rest of the graph, which is a contradiction to the cut definition. □

Theorem 2 (Termination)**.** *Given a planning problem $p = (L, A, s_0, G)$, Alg. 2 terminates after a maximum of $|A|$ iterations.*

Proof. We know from Prop. 1 that the costs of every cut are greater than 0. Since it is set to the action with minimum costs contained in the cut, in every iteration, the residual costs of at least one action are set to 0 in line 14. After at most $|A|$ iterations, all actions have residual costs of 0 and since the graph is connected, $\boldsymbol{i}$ is part of the goal-zone and the algorithm stops (line 10). □

Theorem 3 (Cuts are landmarks (Bonet and Helmert 2010))**.** *Let $p = (L, A, s_0, G)$ be a planning problem and g the justification graph for some* pcf*. Every cut in the justification graph is disjunctive action landmark for p.*

Now we can show admissibility of Alg. 2.

Theorem 4 (Admissibility)**.** *The heuristic values returned by Alg. 2 are admissible.*

Proof. Let $h(s)$ by the values returned by the algorithm for some state s. When the algorithm returns ∞, we know from Thm. 1 that the problem is unsolvable. What is left to show is that $h(s) \leq h^*(s)$ for all states s. We know that $h(s)$ is the sum of the generated cuts (by definition). We know that at least one action from each cut must be contained in every solution (from Thm. 3). Further, the costs of all actions contained in a cut are decreased by the costs of the cut, i.e., when considering some action a from the problem, its overall costs is always greater or equal to the sum of the costs of the cuts in which it is contained, which makes it a cost partitioning. □

Next we want to further investigate the problem of finding a good precondition choice function. Since the overall heuristic is admissible independent from the *pcf*, *good* means maximizing the LM-cut value. In the following, we assume to have a sequence of *pcfs*, one for each iteration of the main loop of the algorithm.

Problem 1 (BoundedPCF)**.** *Given a planning problem $p = (L, A, s_0, G)$ and a natural number k. Determine whether there is a sequence of precondition choice functions pcf_i so that computing the generic LM-Cut heuristic with these* pcfs *yields a heuristic value of at least k, i.e., it finds at least k landmarks.*

Theorem 5. *BoundedPCF is $\mathbb{NP}$-complete.*

Proof. Membership can be easily obtained by guessing k *pcfs* and executing the generic LM-Cut heuristic.
For hardness, we reduce from the problem of independent set (Garey and Johnson 1979, GT20). It asks given an undirected graph $G = (V, E)$ and a natural number k where there is a subset of the vertices $I \subseteq V$ with $|I| = m$ such that no two $u, v \in I$ with $u \neq v$ are adjacent in G, i.e., $\{u, v\} \notin E$. We construct a planning problem in igNF. Let $L = \{i, g\} \cup V$ (wlog. $i, g \notin V$) and $A = \{end\} \cup \{e_{u,v} \mid \{u, v\} \in E\}$. Then $s_0 = \{i\}$, and $G = \{G\}$. For the actions $e_{u,v}$, we set $prec(e_{u,v}) = \{i\}$ and $add(e_{u,v}) = \{u, v\}$. Lastly for end we set $prec(end) = V$ and $add(end) = \{g\}$. We select k for the BoundedPCF problem to be $m + 1$ The only action for which a *pcf* can actually choose a precondition is end.

Let $I = \{v_1, \ldots, v_m\}$ be an independent set. We construct $m+1$ *pcfs* where $pcf_{i+1}(end) = v_i$ and $pcf_1(end) = v_1$, i.e., the i+1th *pcf* chooses the ith node of the independent set. In the first round of the generic LM-Cut heuristic, the goal-zone consists of the goal only and the landmark that is found is end. In the $i + 1$th round of the generic LM-Cut heuristic, the goal-zone consists of the goal, end, and any

action $e_{u,v}$ for which either u or v is already contained in $\{v_1, \dots v_{i-1}\}$. By the choice of $pcf_{i+1}(end)$, any $e_{u,v}$ with $u = v_i$ or $v = v_i$ is connected with end. Thus the set of all such actions forms a landmark. In this step all such actions still have cost 1 – otherwise the other node that is not v_i would have been in $\{v_1, \dots v_{i-1}\}$ which contradicts the fact that I is an independent set. Thus we obtain $m+1$ landmarks and a heuristic value of $m + 1$.

Conversely, if the generic LM-Cut heuristic obtains $m+1$ landmarks, the first is end alone. Any further landmark is induced by the choice of that step's *pcf* for the selected precondition for end. Let this choice be v_i. v_i cannot be connected by an edge to any previously chosen precondition of end – as otherwise that edge's action would already have cost zero, leading to cost zero for that landmark. Thus the set of vertices selected by the *pcf* is independent and contains m many vertices. □

4 A Neuro-Symbolic LM-Cut Heuristic

In this section we introduce a method to learn the pcf based on GNNs instead of using a symbolic approximation. We aim to generalize over different instances of a domain, i.e., to learn one GNN *per domain*. However, there is no technical restriction that prevents us from extending this to a domain-independent approach in future work.

We first give the basic graph definition, then we describe the RL setting, and the training procedure.

4.1 Fact-Action Graph

We represent a planning instance by a directed bipartite fact-action graph that contains all possible justification graphs induced by different *pcf* choices.

Definition 6 (Fact-Action Graph)**.** *Let $p = (L, A, s_0, G)$ be a planning problem in igNF. The Fact-Action Graph is defined as $G = (V, E)$ with vertices $V = V_L \cup V_A$, where $V_L = \{n_f \mid f \in L\}$ contains one node for each fact and $V_A = \{n_a \mid a \in A\}$ contains one node for each action. The edge set is $E = E_p \cup E_e$, with*

$$E_p = \{(n_f, n_a) \mid f \in prec(a)\},$$
$$E_e = \{(n_a, n_f) \mid f \in add(a)\}.$$

A *pcf* corresponds to selecting one incoming precondition edge (n_f, n_a) for each action a.

The topology of G is fixed for a given instance. In the following we introduce node features that encode the current state of the LM-cut computation. At iteration t, each node $v \in V$ is labelled by a feature vector $x_t(v)$. For fact nodes, x_t contains whether the fact is true in the current planning state, whether it is a goal fact, its current h^{max} value, and an indicator for infinite h^{max} values. For action nodes, x_t contains the current residual cost $c_t(a)$ and features of the standard h^{max}-based choice for a, including the resulting relaxed cost estimate. Numerical h^{max} and residual-cost features are normalized by the maximum finite h^{max} value in the current iteration, with a minimum normalization scale of one. We additionally include baseline structural features derived from the standard h^{max}-based *pcf*, namely whether a fact is in the corresponding goal-zone and whether an action is in the corresponding possible cut.

Thus, G represents the fixed space of possible *pcf* choices, while x_t represents the current state of the LM-cut computation. Choosing pcf_t at iteration t induces a justification graph, i.e., a pruned version of G, from which the symbolic LM-cut step extracts a cut and updates residual costs. The updated residual costs and recomputed derived quantities define the next feature function x_{t+1} on the same fixed fact-action graph.

4.2 LM-Cut as a Reinforcement Learning Problem

Reinforcement learning (RL) is commonly applied to sequential decision problems modeled as Markov decision processes (MDPs), where an agent observes the current state, selects an action, receives a reward, and aims to maximize the accumulated return (Sutton and Barto 1998). This naturally fits generic LM-cut: for a fixed planning state s, i.e., the search node for which the heuristic is evaluated, one run of the algorithm can be viewed as a finite-horizon deterministic MDP whose decisions correspond to *pcf* choices across iterations. The planning state s parameterizes the episode and remains fixed, while the MDP state at iteration t is the current residual action-cost function c_t. At iteration t, the MDP action is a factorized choice of one precondition for each action, represented by a *pcf* pcf_t with $pcf_t(a) \in prec(a)$ for every action $a \in A$ (we assume igNF, so every action has at least one precondition). Given pcf_t, the symbolic LM-cut procedure constructs the corresponding justification graph, computes the goal-zone, and extracts a cut L_t. The reward for the iteration is the value of this cut,

$$r_t = \min_{a \in L_t} c_t(a),$$

and residual costs are updated by subtracting r_t from all actions in L_t. The transition to the next MDP state is deterministic: the updated residual costs define c_{t+1}. The episode terminates when the LM-cut stopping criterion is satisfied. The return of the episode is therefore $R(s) = \sum_t r_t$, which is exactly the heuristic value computed by this LM-cut run. Thus, maximizing return directly corresponds to maximizing the heuristic value.

The learned policy only replaces the choice of the *pcf*. Since it is restricted to choices $pcf_t(a) \in prec(a)$, every policy action defines a valid justification graph. Cut extraction and cost updates remain symbolic, so the admissibility argument of generic LM-cut still applies.

4.3 Training

We denote by o_t^s the observation at iteration t of the LM-cut computation for the fixed planning state s. It is represented by the annotated fact-action graph from Def. 6. We employ a Graph Neural Network (GNN) based on the Graph Attention Network (GAT) architecture (Velickovic et al. 2018) to process the graph structure from Def. 6 at each iteration t. A GNN computes node embeddings h_i by iteratively aggregating information from neighboring nodes (Gilmer et al. 2017). For a node i, this aggregation is performed over its

neighborhood $\mathcal{N}(i)$, and stacking multiple layers increases the effective range of this message passing. In GAT, the contribution of each neighbor is modulated by learned attention coefficients $\alpha_{ij}^{(k)}$, computed from the connected node representations and used to weight neighbors differently at each layer. Formally, node embeddings are updated as

$$h_i^{(k+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(k)} W^{(k)} h_j^{(k)}\right),$$

where $\alpha_{ij}^{(k)}$ are learned attention weights, $W^{(k)}$ are trainable parameters, and σ is a non-linear activation function.

We apply the GNN to obtain node embeddings, which are used to parameterize the decision over precondition choice functions.

For each action $a \in A$, we consider its embedding h_a together with the embeddings of its candidate preconditions h_p for $p \in prec(a)$. Each candidate pair is represented by concatenating the action and precondition embeddings, $z_{a,p} = [h_a \,\|\, h_p]$.

A multi-layer perceptron (MLP) is applied to each pair representation $z_{a,p}$. For each action a, the resulting values are normalized over $prec(a)$ using a softmax, yielding a local categorical factor:

$$q_\theta(p \mid a, o_t^s) = \mathrm{softmax}_{p\in prec(a)}\big(\mathrm{MLP}_\theta(z_{a,p})\big).$$

The policy over *pcf*s factorizes as

$$\pi_\theta(pcf_t \mid o_t^s) = \prod_{a\in A} q_\theta(pcf_t(a) \mid a, o_t^s).$$

All decisions share the same GNN encoder and MLP parameters, which are optimized jointly.

We train the model within a RL framework using Proximal Policy Optimization (PPO) (Schulman et al. 2017). The GNN serves as a shared encoder for both precondition scoring and the value function $V_\phi(o_t^s)$. Optimization follows the standard PPO objective with value loss and entropy regularization.

5 Experiments

We implement the symbolic LM-cut computation in Python using `pymimir`, and expose it as a `Gymnasium` environment in which each episode corresponds to one LM-cut computation for a sampled planning state. The actor-critic training is implemented in `PyTorch`, with the GNN encoder implemented in `PyTorch Geometric`. For each domain, we train a separate PPO agent with a GAT-based actor-critic network using five message-passing layers and four attention heads. The remaining PPO hyperparameters are tuned separately per domain with `Optuna`, using 30 trials and validation improvement over the h^{max}-based *pcf* as the selection criterion. Final models are trained on an RTX 3090 GPU for 50,000 PPO updates with four rollouts per update.

5.1 Data

We follow the domain selection of Müller et al. (2026) and use the IPC'23 learning-track domains (Taitler et al. 2024) excluding Miconic, Spanner, and Sokoban, together with Gripper, Logistics, and Visitall from the FF domain collection[2]. Instances are generated with publicly available generators for these domains.

For training, we generate instances of varying sizes and solve them with an optimal planner. Details are given in the appendix. We then collect states along the resulting solution trajectories, excluding the initial state of each training instance. This avoids potential exact overlap with our reference evaluation, which is performed only on initial states of independently generated instances. During training, states are sampled from this pool and rewards are computed online by the symbolic LM-cut computation. Model selection is performed on a fixed held-out validation set. In future work, we plan to investigate more exploratory state-sampling strategies.

For evaluation, we independently generate a reference set using the same generators and evaluate only the initial state of each instance. We generate up to 50 unique instances per size. The actual number of evaluated instances is reported in Table 1. For each reference instance, we compute the standard h^{max}-based LM-cut value using Fast Downward (Helmert 2006) and, where feasible, the exact h^+ value as an upper bound. The latter was computed using the method by Imai and Fukunaga (2015). Our current evaluation compares heuristic values only. Evaluating the learned heuristic inside search, e.g., in terms of coverage and state expansions, remains future work.

5.2 Results

Our results are given in Table 1. First consider $\text{LM-cut}_{h^{max}}$, LM-cut_{π}, and Gain, giving the standard LM-cut value, our learned value, and their difference, respectively.

In half of the domains, the values are close to 0, while in the other domains it is positive, meaning that our heuristic values are larger on average; these are *Floortile*, *Rovers*, *Satellite*, *Transport*, and *Visitall*. The highest gain is present in *Visitall*, followed by *Transport*. We applied the *Paired Two-Sided Exact Sign Test* over per-instance heuristic value differences and found that this gain is statistically significant in the 5 domains given above (column *p*). The column *B/E/W* gives in how many instances our heuristic values were better, equal, and worse than the original one. Interestingly, there are three domains where the values are identical in all cases (*Childsnack*, *Ferry*, and *Gripper*), while in *Blocksworld* and *Logistics* they differ, but are (nearly) equal on average.

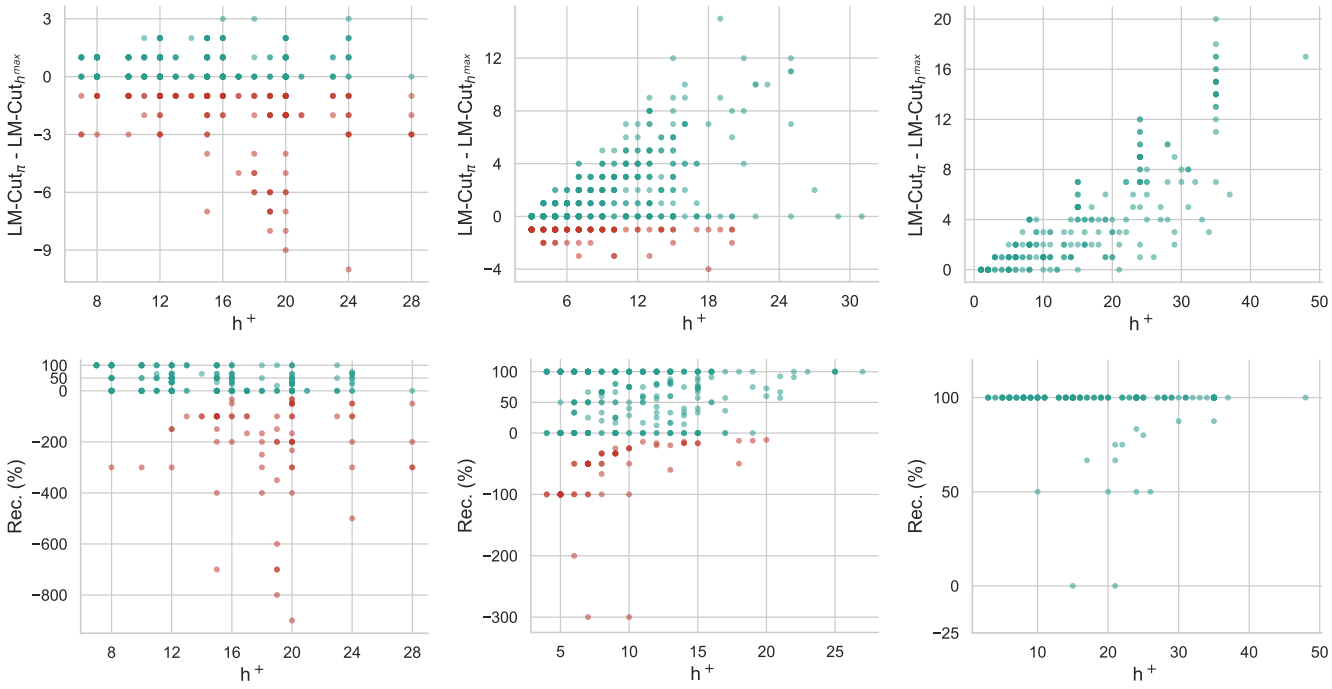
To compare the results to the theoretical limit, we computed the h^+ values, which was not feasible in all states. For *Childsnack*, *Ferry*, and *Gripper*, we have results for 100.0%, 92.7%, and 74.1% of the states, respectively. In these states, the LM-cut values and h^+ are identical, meaning that we cannot gain anything from a better *pcf*.

The last two columns give the average computation time per heuristic value in seconds. Unsurprisingly, there is a

[2]https://fai.cs.uni-saarland.de/hoffmann/ff-domains.html

Domain	N	LM-cut$_{h^{max}}$	LM-cut$_{\pi}$	Gain	B/E/W	p	$\bar{t}_{h^{max}}$	$\bar{t}_{\pi}$
Blocksworld	650	26.37	26.34	-0.03	50/544/56	0.63	0.13	0.710
Childsnack	548	4.88	4.88	0.00	0/548/0	–	0.09	0.066
Ferry	700	17.67	17.67	0.00	0/700/0	–	0.09	0.275
Floortile	590	15.42	15.82	0.40	261/244/85	< 0.01	0.10	0.254
Gripper	700	15.38	15.38	0.00	0/700/0	–	0.10	0.228
Logistics	700	7.09	7.11	0.02	30/646/24	0.49	0.10	0.06
Rovers	700	14.36	14.72	0.36	186/485/29	< 0.01	0.09	0.186
Satellite	700	13.21	13.58	0.37	268/375/57	< 0.01	0.09	0.177
Transport	700	6.02	7.30	1.28	326/329/45	< 0.01	0.10	0.150
Visitall	346	18.86	28.56	9.70	303/42/1	< 0.01	0.09	0.475
Avg./sum	6334	13.74	14.53	0.79	1424/4613/297	< 0.01	0.10	0.247

Table 1: Reference dataset evaluation. LM-cut$_{h^{max}}$ is the average heuristic value obtained with the standard h^{max}-based *pcf*, LM-cut$_{\pi}$ is the average value obtained with the learned *pcf* policy, and Gain is LM-cut$_{\pi}$ minus LM-cut$_{h^{max}}$. B/E/W counts instances where the learned value is better/equal/worse than LM-cut$_{h^{max}}$. The reported p is the two-sided exact sign-test p-value for LM-cut$_{\pi}$ − LM-cut$_{h^{max}}$ over paired instances, excluding zero differences. $\bar{t}_{h^{max}}$ and $\bar{t}_{\pi}$ are average per-state runtimes in seconds for the h^{max}-based and learned LM-cut computations, respectively.

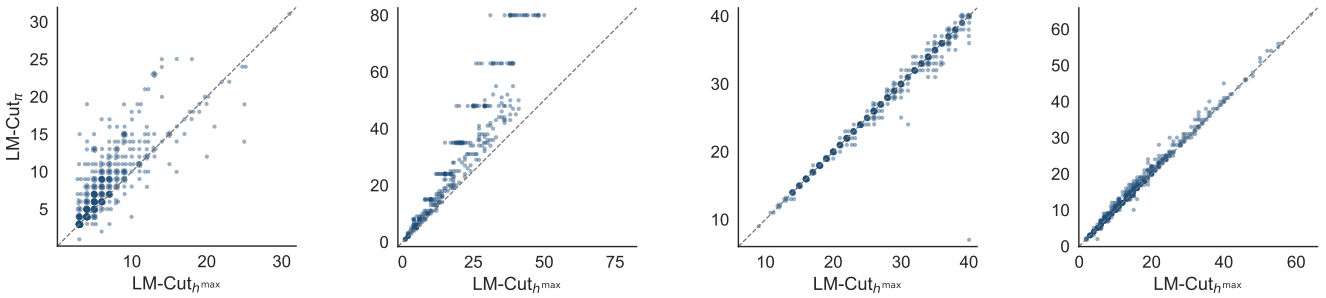


(a) Floortile (b) Transport (c) Visitall

Figure 3: Comparison to h^+ for selected domains. The top shows the gain of our heuristic compared with the baseline against the h^+ value. The bottom the percentage of the gap between the original LM-cut and h^+ that has been closed by our heuristic.

large difference between the heuristics. This is partly caused by the underlying approaches, because we need to evaluate the GNN several times during heuristic computation. However, it will also be caused by the different implementations. The original LM-cut uses the *Fast Downward* implementation, while our implementation is based on Python and not optimized towards runtime. For future work, we plan to integrate our heuristic into *Fast Downward*. We observed that one bottleneck is the construction of the graph. Since the graph structure is static, we plan to use an incremental computation of the GNN input. We hope to speed up the computation with these measures. If it is still too costly afterwards, we see two directions for further optimization. First, the parallelization by having more than one process computing the heuristic values while having one that does the search. Second, replacing GNNs by other learning methods as e.g. done by Chen, Trevizan, and Thiébaux (2024).

Figure 3 gives a comparison of the two LM-cut heuristics to h^+ for selected domains. Be aware that we only have data for 78.0% (*Floortile*), 97.4% (*Transport*), and 66.5%



(a) Transport (b) Visitall (c) Blocksworld (d) Rovers

Figure 4: Per-instance comparison of LM-cut$_{h^{max}}$ and LM-cut$_\pi$ for selected domains. Each point corresponds to one reference state. Points above the diagonal indicate higher heuristic values for LM-cut$_\pi$.

(*Visitall*) of the states. Since computation time will increase with problem size, these results will be biased towards small states. Each dot visualizes the heuristic values of one state. The upper row of diagrams shows the difference between the LM-cut values (on the y-axis) and the h^+ value (on the x-axis). The lower row shows the percentage of the gap between the original LM-cut and h^+ that has been closed by our work (on the y-axis) against $h+$. In *Floortile*, the gap has been entirely closed for some states. But interestingly, there is a number of states where our heuristic performs much worse. In *Transport*, our heuristic performs much better in most states already, but there is still a gap, i.e., room for improvement. In *Visitall*, our heuristic performs optimally already for a large number of states.

Figure 4 shows a direct comparison of the two heuristics for selected domains. It plots the heuristic values against each other, i.e., a dot above the diagonal means that our heuristic value is higher. From left to right, it can be seen that in the *Transport* domain, while our heuristic values are better on average, there are still states that result in worse values. This is different in the *Visitall* domain, where nearly all (all except one) dots are above the diagonal. The results in *Blocksworld* are also interesting: while the average is nearly 0, there is still a large number of states where the heuristic values diverge. In *Rovers* it is the other way around, here the average is significantly better, but the values are nevertheless scattered around the diagonal.

6 Discussion

While our results show that there are domains in which we achieve higher performance than the original LM-cut heuristic, our investigation was only based on a comparison of the heuristic values. While this direct comparison is meaningful due to admissibility, the actual impact on search is not entirely clear. The next step for future work is to evaluate our approach in a search, first by comparing the number of nodes that have to be expanded until finding a solution. Further, we want to extend the evaluation set to include more domains.

Even when this evaluation shows a reduction in search space, the runtime of the heuristic computation will be an issue for practical application. There are several ways to mitigate this problem. One way could be to predict the *pcf* with different ML methods, as demonstrated in other work on heuristic computation (Chen, Trevizan, and Thiébaux 2024). Given that the *pcf* is learnable with GNNs, it might also be learnable with *Weisfeiler-Leman*-based features and ML methods that need less computation time. A second – more technical – way could be to parallelize the computation of multiple search nodes during search.

So far, we annotated our graph with several elements that are based on h^{max}. Intuitively, these features provide a global view on the location of a node in the graph. For future work, we want to evaluate whether these are actually necessary and test other features.

To speed up the learning process, we are further working on a variant based on supervised learning. We generate optimal *pcfs* as training data using SAT- and ILP-based solvers.

While we trained our models *per domain*, there is no technical limitation in our approach that prevents domain-independent learning. We are also interested in analysing the performance of our approach in this setting.

7 Conclusion

In this work we presented a neuro-symbolic variant of the LM-cut heuristic. We first showed that finding the optimal *pcf* (maximizing the overall heuristic value) is NP-complete. Then we introduced an approach to learn the *pcf* based on GNNs and Reinforcement Learning. Our work is the first neuro-symbolic approach to compute a provably admissible heuristic function. While it is still preliminary, our empirical results show that there are domains for which it is possible to learn *pcfs* that perform statistically significantly better than the commonly used approximation based on h^{max}. For future work, we want to test the approach on a wider benchmark set, and evaluate it in an actual search. For the latter, it will be necessary to speed up the computation, either based on parallelization or by applying an alternative machine learning method.

Acknowledgments

The authors want to thank Sylvie Thiébaux for the fruitful discussions on the topic.

References

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

Bonet, B.; and Helmert, M. 2010. Strengthening Landmark Heuristics via Hitting Sets. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI)*, 329–334. IOS Press.

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

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.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 1263–1272. PMLR.

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.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

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

Helmert, M.; and Röger, G. 2023. Lecture "Planning and Optimization", Chapter "Landmarks: Cut Landmarks & LM-Cut Heuristic".

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research*, 22: 215–278.

Imai, T.; and Fukunaga, A. 2015. On a Practical, Integer-Linear Programming Model for Delete-Free Tasks and its Use as a Heuristic for Cost-Optimal Planning. *Journal of Artificial Intelligence Research*, 54: 631–677.

Katz, M.; and Domshlak, C. 2010. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, 174(12-13): 767–798.

Lauer, P.; and Fickert, M. 2020. Beating LM-cut with LM-cut: Quick Cutting and Practical Tie Breaking for the Precondition Choice Function. In *Proceedings of the 12th ICAPS workshop on Heuristics and Search for Domain-Independent Planning (HSDIP)*.

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.

Müller, N. J.; Oster, M.; Valera, I.; Hoffmann, J.; and Gros, T. P. 2026. Per-Domain Generalizing Policies: On Learning Efficient and Robust Q-Value Functions (Extended Version with Technical Appendix). *arXiv preprint arXiv:2603.17544*.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

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

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 38th 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 32nd 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.

Sutton, R. S.; and Barto, A. G. 1998. *Reinforcement learning – An introduction*. Adaptive computation and machine learning. MIT Press. ISBN 978-0-262-19398-6.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Magazine*, 45(2): 280–296.

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.

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*. OpenReview.

Appendix

	Train/Val		Eval	
Domain	Range	N	Range	N
Blocksworld	8–17	763	8–20	650
Childsnack	8–18	357	8–21	548
Ferry	8–18	895	8–21	700
Floortile	7–16	482	7–19	590
Gripper	8–18	773	8–21	700
Logistics	8–18	753	8–21	700
Rovers	10–20	839	10–23	700
Satellite	8–18	885	8–21	700
Transport	8–18	881	8–21	700
Visitall	4–81	376	4–81	346

Table 2: Per-domain object-count ranges and instance counts for the training/validation and evaluation sets. Range denotes the minimum and maximum number of objects, and N the number of instances.