Learning Admissible Heuristics via Cost Partitioning

Hugo Barral[1,2], **Quentin Cappart**[2,3], **Marie-José Huguet**[1], **Sylvie Thiébaux**[1,4]
[1]LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France
[2]Polytechnique Montréal, Montréal, Canada
[3]UCLouvain, Louvain-la-Neuve, Belgium
[4]Australian National University, Canberra, Australia
{hugo.barral,marie-jose.huguet}@laas.fr, quentin.cappart@polymtl.ca, sylvie.thiebaux@anu.edu.au

Abstract

Admissible heuristics are essential for optimal planning, yet learning them remains challenging due to the risk of overestimation. Cost partitioning combines multiple abstraction heuristics while preserving admissibility, but computing optimal partitions online is expensive. We propose a framework that learns to infer admissible cost partitions by leveraging the Lagrangian dual equivalence between cost partitioning and multiplier prediction. Planning states and patterns are encoded as labelled graphs, and an action-centric variant of the Weisfeiler–Leman algorithm extracts structural feature vectors. A deep architecture with axial self-attention and a softmax output layer maps these features to cost weights that satisfy the partition constraints by construction, ensuring admissibility. Experiments demonstrate reduced node expansions compared to suboptimal partitioning baselines while maintaining strict admissibility. To our knowledge, this is the first machine-learned heuristic guaranteed to be admissible.

1 Introduction

Recent advances in machine learning are starting to have an impact on planning, in particular through learning search guidance in the form of heuristic functions (Shen, Trevizan, and Thiébaux 2020; Chen, Trevizan, and Thiébaux 2024; Horcík et al. 2025; Corrêa, Pereira, and Seipp 2025), state rankings (Garrett, Kaelbling, and Lozano-Pérez 2016; Hao et al. 2025), policies (Toyer et al. 2020; Ståhlberg, Bonet, and Geffner 2022a,b; Silver et al. 2024; Chen et al. 2025), and action pruning (Musayev et al. 2025). However, the focus of these works has been on improving the scalability of planning, rather than the quality of the solutions. In particular, learning admissible heuristics, able to guide the search of optimal planners, remains an open problem. Some contributions present models that achieve low residual errors or relaxed probabilistic guarantees (Ernandes and Gori 2004; Ofir Marom and Rosman 2020; Futuhi and Sturtevant 2026). Yet, despite their accuracy, these learned estimators lack formal admissibility guarantees. Since a single overestimation of h^* can cause A* to return a suboptimal plan, the landscape still lacks a robust method to learn admissible heuristics for optimal planning.

To address this challenge, we explore a new research direction. Instead of directly attempting to learn admissible heuristics from scratch, we learn to combine admissible heuristics via cost partitioning. Cost partitioning is a well-established, rigorous method which distributes the cost of actions amongst multiple admissible heuristics, to ensure that adding them remains admissible, thereby producing more informed admissible heuristics (Haslum, Bonet, and Geffner 2005; Katz and Domshlak 2010; Pommerening, Röger, and Helmert 2013; Pommerening et al. 2015; Seipp, Keller, and Helmert 2020). For certain classes of admissible heuristics such as abstraction heuristics, an optimal cost partitioning (OCP) maximizing the combined heuristic value over all partitions can be computed in polynomial time via linear programming (Katz and Domshlak 2010). However the resulting linear program (LP) is too large to be solved at every node of the search, which reduces the usage of OCP in practice. This suggests that a learning-based approach could be beneficial to speed up the process.

In this paper, we develop such an approach, taking inspiration from two key results from the literature. The first is the theoretical insight that OCP is equivalent to a Lagrangian dual problem (Pommerening et al. 2019): the multipliers that maximize a Lagrangian lower bound, and provide tight bounds in branch-and-bound, directly correspond to cost partition weights. In other words, each multiplier encodes how much of an action's original cost is allocated to a particular abstraction. The second result is the recent development of machine learning models that can be trained to predict tight Lagrangian multipliers for integer linear and constraint programs (Abbas and Swoboda 2024; Parjadis et al. 2024; Bessa et al. 2025). Taken together, these results suggest it should be possible to develop similar machine learning approaches to infer valid cost partitions.

We introduce a novel framework that is based on this perspective. Rather than solving an optimization problem from scratch for every evaluated state, we train a domain-specific model capable of inferring valid cost partitions online. Our framework constructs a graph representation of a planning state and its abstractions, from which we extract Weisfeiler-Leman (WL) feature vectors (Chen, Trevizan, and Thiébaux 2024). A deep learning architecture then contextualizes the feature vectors via a self-attention mechanism (Vaswani et al. 2017) and produces cost partition weights. A softmax output layer enforces that the weights sum to 1, ensuring the resulting heuristic remains admissible. The model is trained on optimal cost partitions from various planning instances from the given domain, in order to estimate a function that

compute optimal cost partitions from contextualized action features. Once learned, this relationship allows the model to infer effective cost partitions even on problems larger than those encountered during training.

We empirically demonstrate through experiments on 6 domains that these inferred partitions yield a strictly admissible heuristic that can reduce search node expansions compared to suboptimal cost partitioning baselines.

The remainder of this paper is structured as follows. Section 2 provides the necessary background and notation for our learning approach to planning. Section 3 details our model architecture for inferring cost partition. Section 4 presents an empirical evaluation on 6 International Planning Competition domains, with a focus on coverage, node expansions, and search time. Section 5 concludes with a discussion of limitations and directions for future work.

2 Background and Notation

2.1 Planning

Throughout this paper we assume that planning tasks are provided in lifted form and internally grounded and transformed into a finite domain representation (FDR) by the planner prior to search (Helmert 2009). We define these two representations below.

A lifted planning task is a tuple $\Pi^{\mathsf{lifted}} = \langle \mathcal{P}, \mathcal{A}, \mathcal{C}, s_0, G \rangle$, where $\mathcal{P}$ is a set of predicates, $\mathcal{A}$ is a set of action schemas, $\mathcal{C}$ is a set of object constants, s_0 is the initial state and G is the goal condition. A predicate $p \in \mathcal{P}$ of arity $n_p \in \mathbb{N}_0$ has parameters $\{x_i \mid 1 \leq i \leq n_p\}$. An action schema $a \in \mathcal{A}$ of arity $n_a \in \mathbb{N}_0$ is defined by a tuple $\langle \Delta(a), \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a) \rangle$, where $\Delta(a)$ is a set of parameters such that $|\Delta(a)| = n_a$, and $\mathsf{pre}(a)$, $\mathsf{add}(a)$ and $\mathsf{del}(a)$ are sets of predicates from $\mathcal{P}$ instantiated with either parameter variables or objects in $\Delta(a) \cup \mathcal{C}$. A (ground) atom (resp. action) is an instance of a predictate (resp. action schema) obtained by substituting all its parameters with objects from $\mathcal{C}$. A state in the lifted representation is the set of all atoms that are true in that state. The goal G is a set of atoms and a state s is a goal state if $G \subseteq s$.

An FDR planning task is a tuple $\Pi^{\mathsf{FDR}} = \langle V, O, s_0, s_* \rangle$, where V is a set of finite-domain state variables, O is a set of actions, s_0 is the initial state and s_* is the goal condition. Each $v \in V$ has a finite domain D_v. A fact is a pair $\langle v, d \rangle$, with $v \in V, d \in D_v$, and equates a ground atom of the underlying lifted task. An assignment (resp. a partial assignment) is a set of facts where each $v \in V$ appears once (resp. at most once). Each $o \in O$ is defined by its precondition $\mathsf{pre}(o)$ and effect $\mathsf{eff}(o)$, expressed as partial assignments on V. Finally, s_0 and s_* are an assignment and partial assignment, respectively.

An action o is applicable in state s if $\mathsf{pre}(o) \subseteq s$, and leads to the successor state $\mathsf{succ}(s, o) = s \oplus \mathsf{eff}(o)$ where $\oplus$ replaces $(v, d) \in s$ with $(v, d') \in \mathsf{eff}(o)$ if $d \neq d'$. A solution or a plan is a sequence of actions $\pi = \langle o_1, \ldots, o_n \rangle$, that can be applied to s_0 and to all the subsequent states to achieve a goal state. That is, π induces a sequence of states $s_0, \ldots, s_n$ such that $\mathsf{pre}(o_i) \subseteq s_{i-1}$ and $s_i = \mathsf{succ}(s_{i-1}, o_i)$ for $i = 1, \ldots, n$, and $s_* \subseteq s_n$. An action cost is defined by $c(o) \in \mathbb{R}_0^+$. The cost of π is $c(\pi) = \sum_{o_i \in \pi} c(o_i)$. A planning task is solvable if there exists at least one plan. In an optimal planning setting, we would like to find a plan π^* of minimum cost, i.e. $\arg\min_{\pi^*} c(\pi^*)$.

2.2 Abstraction Heuristics and Cost Partitioning

Abstraction heuristics map states to a smaller set of abstract states, treating concrete states mapped to the same abstract state as equivalent under the abstraction. This induces an abstract transition system whose size is much smaller than the original state space. In this abstract system, optimal goal distances yield an admissible heuristic for the original task and can be computed efficiently using uninformed search algorithms. Abstractions are commonly generated by projecting tasks onto a small set of variables or *pattern* $P \subseteq V$. We write $\Pi^P = \langle P, O^P, s_0^P, s_*^P \rangle$ for the pattern projection of Π^{FDR}, where for a (partial) assignment s, the projection $s|_P$ is the restriction of s to the variables in P, and for each $o \in O$, the projected action $o^P \in O^P$ keeps $\mathsf{pre}(o^P) = \mathsf{pre}(o)|_P$ and $\mathsf{eff}(o^P) = \mathsf{eff}(o)|_P$.

To focus on multiple aspects of the task and obtain a more informed heuristic estimate, multiple abstractions $P_1, \ldots, P_k$ are typically combined. However, these heuristics are generally not additive, as the same action may be applied redundantly across multiple abstractions. Cost partitioning addresses this by distributing the cost of each action among the abstractions before computing their heuristics, ensuring that the sum of the per-abstraction heuristic estimates never exceeds the true cost (Katz and Domshlak 2010). Let us note c^P, the cost function distributed to a pattern abstraction P. For the distribution to ensure the additivity of a set of patterns $\mathcal{S}$, the following constraint must be satisfied:

$$\sum_{P \in \mathcal{S}} c^P(o) \leq c(o), \forall o \in O \tag{1}$$

If each abstraction P is evaluated under its assigned cost function c^P, producing heuristic h^P, then the additive heuristic

$$h^{\mathcal{S}}(s, c) = \sum_{P \in \mathcal{S}} h^P(s, c^P) \tag{2}$$

is admissible at state s.

The informativeness of the resulting additive heuristic depends on the chosen abstractions and partition, and also on the current search state s. Optimal Cost Partitioning (OCP) is the the problem of computing the cost partitioning that maximises the heuristic value $h^{\mathcal{S}}(s, c)$. It can be can be expressed as an LP that additively combines abstraction heuristics as in (2) by enforcing the cost partition constraint (1) (Pommerening et al. 2014). However, solving this LP anew at each state is computationally prohibitive in practice. Existing approaches either settle for faster, suboptimal partitioning (Karpas and Domshlak 2009; Seipp, Keller, and Helmert 2017) and/or do not recompute the partition at every state (Seipp 2021).

Furthermore, a theoretical result by Pommerening et al. (2019) shows that the OCP problem, whose dual belongs to the operator-counting framework (Pommerening et al. 2014), can be seen as maximizing a sum of independent

Algorithm 1: WL algorithm

Input: Graph $G = \langle N, E, \kappa, l \rangle$; number of iterations L.
Output: Set of colours.

1: $\kappa^0(v) \leftarrow \kappa(v), \forall v \in V$
2: **for** $j = 1, \ldots, L$ **do for** $v \in V$ **do**
3: $\quad \kappa^j(v) \leftarrow \text{hash}\big(\kappa^{j-1}(v), \{(\kappa^{j-1}(u), \iota) \mid \iota \in \Sigma_E, u \in \mathcal{N}_\iota(v)\}\big)$
4: **end for**
5: **return** $\bigcup_{j=0,\ldots,L} \{\kappa^j(v) \mid v \in V\}$

sub-problems, each parameterized by Lagrangian multipliers. Consequently, any technique capable of producing Lagrangian multipliers also yields a valid cost partition, providing an alternative avenue to tackle the online computational challenge. In particular, Parjadis et al. (2024) showed that machine learning can be used to obtain tight multipliers in the context of Lagrangian relaxation applied to constraint programming, while Abbas and Swoboda (2024); Bessa et al. (2025) demonstrated similar results for Lagrangian decomposition. Inspired by this work, we propose a related approach to learn Lagrangian multipliers that yield valid and tight cost partitions.

2.3 Graph Representation of Planning States

A suitable learning approach must start with an encoding of planning states. A graph representation is a natural choice because a planning state fundamentally consists of entities connected by structural relationships. A labelled graph is a tuple $G = \langle N, E, \kappa, l \rangle$, where N is a set of nodes, $E \subseteq \binom{N}{2}$ is a set of undirected edges, $\kappa : N \to \Sigma_N$ maps nodes to a set of colours Σ_N, and $l : E \to \Sigma_E$ maps edges to a set of colours Σ_E. The neighbourhood of a node u under edge colour ι is $\mathcal{N}_\iota(u) = \{v \mid (u, v) \in E \cap l(u, v) = \iota\}$. The neighbourhood of a node u in a graph is $\mathcal{N}(u) = \bigcup_{\iota \in \Sigma_E} \mathcal{N}_\iota(u)$.

We rely on the modified Weisfeiler–Leman (WL) algorithm (Chen, Trevizan, and Thiébaux 2024) to extract structural features. The original WL algorithm is an incomplete test to decide graph non-isomorphism. It has since evolved into a reliable method for extracting a vector representation of a graph's structure (Shervashidze et al. 2011). Because the WL procedure is entirely deterministic, it provide a consistent and reproducible representation of each state-pattern pair, suitable as input to a learned predictor.

Alg. 1 shows the WL algorithm that takes as input a labelled graph G and a number of iteration L. After initializing the nodes original colour, Line 3 iteratively updates the colours of each node v. It aggregates the unions of node and edge colours of its neighbours in a set and hash them with v's current colour into a colour for the next iteration. The algorithm returns a set of colours seen over all iterations. Those set of colours can be represented as histogram vector v with a size d equal to the number of observed colours on a training set of graphs. Let $v[c]$ count how many times the WL algorithm has encountered colour c throughout its iterations. As there is no guarantee that all possible colours can be observed from a limited training set of a given planning domain, non-observed colours encountered after training are purely ignored.

3 Learning to Infer Cost Partitions

In this section, we describe our methodology to avoid solving the optimal cost partitioning problem from scratch for every evaluated state. We define a framework to learn a domain-specific model that predicts a valid cost partition for the current state and a set of abstractions. Figure 1 schematizes the full pipeline. First, the planner state and each pattern are jointly encoded as a labelled graph. From there, we leverage a variant of the WL algorithm to extract action-centric feature vectors; these are shaped into a three-dimensional array whose axes correspond to abstraction, ground action, and feature dimension. Finally, a neural network equipped with self-attention contextualizes the feature array and, via a softmax output layer, produces cost partition weights that satisfy the sum constraint. Section 3.1 details the graph representations and our method to project graph to abstraction patterns. Section 3.2 presents our WL variant to extract action-centric features. Section 3.3 describes the attention-based predictor and how this model enforces admissibility.

3.1 Graph Representation of Abstractions

As described in Section 2.3, a planning state s must be encoded as a labelled graph $G(s)$ that captures the entities and structural relationships of the planning task. We adapt two graph representations $G(s)$ from the literature, the Action-Object-Atom graph (AOAG) (Wang and Trevizan 2025) and the FDR learning graph (FLG) (Chen, Thiébaux, and Trevizan 2024), to capture the current task state and the connections between actions and state components. The former focuses on the lifted structure of objects and atoms, whereas the latter encapsulates a grounded (FDR) view of the task. We assume our original lifted task is grounded through a translation process that allows us to keep information from both representations. Let $\mathcal{D}_\Pi = \bigcup_{v \in V} D_v$ be the set of all domain values of a task Π. During grounding we retain the mappings $\rho : \mathcal{D}_\Pi \to \mathcal{P}$ and $sch : O \to \mathcal{A}$ that link grounded values and actions to their lifted counterparts. We write an example planning task Π_X with lifted and grounded elements in Table 1 to illustrate the graph representations.

Action-Object-Atom graph We use a particular case of AOAG from Wang and Trevizan (2025) where the graph always represent all grounded actions in O. The graph's nodes represent the lifted task's objects, actions, and atoms that are true in the current state or in the goal. The action nodes are labeled by their corresponding action schema, and the atom nodes are labeled by their "goal status" which encodes whether the atom is an achieved goal (`ag`), an unachieved goal (`ug`) or a non-goal atom (`ap` for "achieved proposition" since it must then be true in the current state). The graph edges connect object nodes to the nodes representing the proposition and actions that include this object in their argument list at some position i, and are labeled by

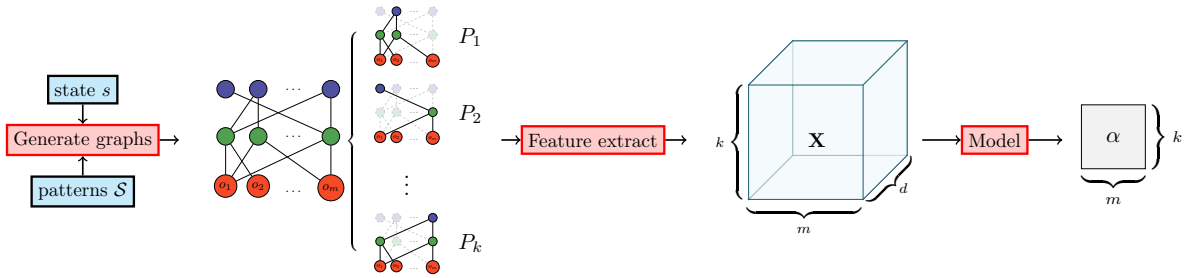


Figure 1: Three-stage pipeline for learning cost partitions. (Left) a state and its patterns are encoded as labelled graphs. (Center) WL algorithm yields action-centric embeddings arranged in a 3D array. (Right) Model with self-attention mechanism predicts admissible partition weights. k denotes the number of patterns, m the number of actions and d the number of features.

Lifted	FDR
$\mathcal{P} = \{p_1, p_2\}$ $\mathcal{C} = \{c_1, c_2\}$	$V = \{v_1, v_2\}$ $D_{v_1} = \{F_1, T_1\}$ $D_{v_2} = \{F_2, T_2\}$
$\mathcal{A} = \{a_1(c_1, c_2), a_2(c_1)\}$ $\mathsf{pre}(a_1) = \{\neg p_1(c_1), p_2(c_2)\}$ $\mathsf{pre}(a_2) = \{\neg p_2(c_2)\}$ $\mathsf{add}(a_1) = \{p_1(c_1)\}$ $\mathsf{add}(a_2) = \{p_2(c_2)\}$ $\mathsf{del}(a_1) = \{\neg p_2(c_2)\}$ $\mathsf{del}(a_2) = \{\}$	$O = \{o_1, o_2\}$ $\mathsf{pre}(o_1) = \{\langle v_1, F_1\rangle,$ $\langle v_2, T_2\rangle\}$ $\mathsf{pre}(o_2) = \{\langle v_2, F_2\rangle\}$ $\mathsf{eff}(o_1) = \{\langle v_1, T_1\rangle,$ $\langle v_2, F_2\rangle\}$ $\mathsf{eff}(o_2) = \{\langle v_2, T_2\rangle\}$
$G = \{p_1(c_1), p_2(c_2)\}$	$s_* = \{\langle v_1, T_1\rangle,$ $\langle v_2, T_2\rangle\}$
$s = \{p_1(c_1)\}$	$s = \{\langle v_1, T_1\rangle,$ $\langle v_2, F_2\rangle\}$

Table 1: Example task Π_X in lifted and FDR formalisms. The facts $\langle v_1, T_1\rangle$, $\langle v_2, T_2\rangle$, $\langle v_1, F_1\rangle$, $\langle v_2, F_2\rangle$ in the FDR representation encode the atoms $p_1(c_1)$, $p_2(c_2)$ and their respective negation in the lifted representation.

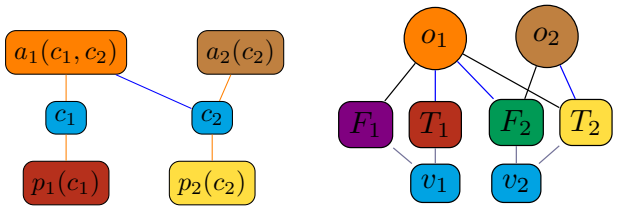
the index i. Formally, the AOAG of a task Π is the graph $G_{\text{AOAG}}(s) = \langle N, E, \kappa, l\rangle$ with:

- $N = \mathcal{C} \cup s \cup G \cup O$
- $E = \bigcup_{p(x_1,\ldots,x_{n_p}) \in s_0 \cup G} \{\langle p, x_1\rangle, \ldots, \langle p, x_{n_p}\rangle\} \cup \bigcup_{\Delta(a)=(x_1,\ldots,x_{n_a}) \in \mathcal{A}} \{\langle a, x_1\rangle, \ldots, \langle a, x_{n_a}\rangle\}$
- $\kappa : V \to \{\mathtt{ob}\} \cup (\{\mathtt{ap}, \mathtt{ag}, \mathtt{ug}\} \times \mathcal{P}) \cup \mathcal{A}$ defined by

$$u \to \begin{cases} \mathtt{ob}, & \text{if } u \in \mathcal{C}. \\ (\mathtt{ap}, p), & \text{if } u = p \in s \setminus G. \\ (\mathtt{ag}, p), & \text{if } u = p \in s \cap G. \\ (\mathtt{ug}, p), & \text{if } u = p \in G \setminus s. \\ sch(u), & \text{if } u \in O. \end{cases}$$

- $l : E \to \mathbb{N}$ with $l(e) = i$ for $e = \langle \_, x_i\rangle$

Fig. 2a illustrates the AOAG graph of the example task defined in Table 1.



(a) AOAG graph. (b) FLG graph.

Figure 2: Graph representations for Π_X in Table 1.

FDR learning graph We provide an alternative to the original FLG from Chen, Thiébaux, and Trevizan (2024) that is made compatible with the WL algorithm by integrating predicate and action schema information as node colours. The graph's nodes represent the FDR task variables, values, and actions. Action nodes are labeled by their corresponding action schema, whereas the label of a value node encodes the predicate it stems from (via the function ρ) and its "goal status", similarly as for the AOAG. Edges connect variables with their possible values, and these values to the actions that use them as preconditions or effects. Edges are labelled with the type of edge they represent (i.e. variable-value, precondition, or effect edge). Formally, the FLG of a task Π is the graph $G_{\text{FLG}}(s) = \langle N, E, \kappa, l\rangle$ with:

- $N = V \cup \mathcal{D}_\Pi \cup O$
- $E = E_{\mathtt{var:val}} \cup E_{\mathtt{pre}} \cup E_{\mathtt{eff}}$ with
  $E_{\mathtt{var:val}} = \bigcup_{v \in V} \{\langle v, d\rangle_{\mathtt{var:val}} \mid d \in D_v\}$,
  $E_{\mathtt{pre}} = \bigcup_{o \in O} \{\langle o, d\rangle_{\mathtt{pre}} \mid \exists v \langle v, d\rangle \in \mathsf{pre}(o)\}$ and
  $E_{\mathtt{eff}} = \bigcup_{o \in O} \{\langle o, d\rangle_{\mathtt{eff}} \mid \exists v \langle v, d\rangle \in \mathsf{eff}(o)\}$
- $\kappa : V \to \{\mathtt{var}\} \cup (\{\mathtt{av}, \mathtt{uv}, \mathtt{ag}, \mathtt{ug}\} \times \mathcal{P}) \cup \mathcal{A}$ defined by $u \to$

$$\begin{cases} \mathtt{var}, & \text{if } u \in V. \\ (\mathtt{av}, \rho(d)), & \text{if } u = d,\ \exists v\ \langle v, d\rangle \in s \setminus s_*. \\ (\mathtt{uv}, \rho(d)), & \text{if } u = d,\ \exists v\ d \in D_v, \langle v, d\rangle \notin (s \cup s_*). \\ (\mathtt{ag}, \rho(d)), & \text{if } u = d,\ \exists v\ \langle v, d\rangle \in s \cap s_*. \\ (\mathtt{ug}, \rho(d)), & \text{if } u = d,\ \exists v\ \langle v, d\rangle \in s_* \setminus s. \\ sch(u), & \text{if } u \in O. \end{cases}$$

Algorithm 2: Action-centric WL algorithm

Input: Projected graph $G(s)^P = \langle N, E, \kappa, l \rangle$; set of action O; number of iterations L.
Output: List of set of colours.

1: $\kappa^0(v) \leftarrow \kappa(v), \forall v \in V$
2: **for** $o \in O$ **do**
3: $\quad H^o \leftarrow \{o\} \cup \{u \mid u \in \mathcal{N}(o)\}$
4: **end for**
5: **for** $j = 1, \ldots, L$ **do for** $v \in V$ **do**
6: $\quad \kappa^j(v) \leftarrow \text{hash}\big(\kappa^{j-1}(v), \{(\kappa^{j-1}(u), \iota) \mid \iota \in \Sigma_E, u \in \mathcal{N}_\iota(v)\}\big)$
7: **end for**
8: **return** $\big\langle \bigcup_{j=0,\ldots,L} \{\kappa^j(u) \mid u \in H^o\} \mid o \in O \big\rangle$

- $l : E \to \{\texttt{var:varl}, \texttt{pre}, \texttt{eff}\}$ with $l(e) = \alpha$ for $e \in E_\alpha$

Fig. 2b illustrates the FLG graph of the example task defined in Table 1.

Project to pattern abstraction Now that a planning state can be encoded as a labelled graph $G(s)$, we can adapt this same graph to a given pattern P to capture the information specific to the state s in the projected task Π^P. We propose a generic masking method that projects $G(s)$ onto any chosen pattern P. Starting from the graph construction for either AOAG or FLG, we obtain the projected graph $G(s)^P$ by masking (i.e., removing) all vertices and edges that are not relevant to the variables in P. For FLG, masking is applied to nodes that represent a variable v and their domain D_v such that $v \notin P$. Since AOAG is a lifted representation, we first ground the task using the SAS$^+$ translation (Helmert 2009), which assigns each object and atom node to the corresponding SAS$^+$ variables. We then mask all nodes associated with variables $v \notin P$. Edges are also masked from the graph whenever at least one of their incident nodes is masked. However, action nodes are never masked, even when they become disconnected. This is because the feature extraction step must produce an embedding for every ground action of the pattern abstraction, regardless of its connectivity in the projected subproblem. Consequently, $G(s)^P$ may contain isolated action nodes.

3.2 Action-centric Feature Extraction

To predict a cost partition for a set of patterns $\mathcal{S}$ and a state s, our framework must extract informative features for every action $o \in O$ and every pattern $P \in \mathcal{S}$ from the $G(s)^P$ projection. Cost partitioning distributes the original cost of each action across the abstractions, subject to the constraint (1) and the quality of the resulting additive heuristic critically depends on how the distribution is performed. An effective partition allocates a larger share of an action's cost to those abstractions where the action is structurally important, i.e., where it contributes to a strong heuristic estimate. Therefore, the features must capture the structural role of o within the pattern projection Π^P, for $P \in \mathcal{S}$ at state s, enabling the subsequent model to infer high-quality cost partitions.

To compute per-node embeddings, we introduce a new Action-centric WL (AWL) in Alg. 2 based on WL procedure from Section 2.3. A projected graph $G(s)^P$ and the set of action O are taken as input along with the number of iterations L. Line 3 collects a hop set H^o for each action node $o \in O$ that contains o together with its immediate neighbours. This hop set helps to capture the local structural context of an action. After the standard WL update on every node at Line 6, the algorithm returns a set of colours that appears in the hop set H^o of each action o across all L iterations. We can represent each of those sets as a histogram feature vector described in Section 2.3.

For a state s, we apply this process to each $P \in \mathcal{S}$ and arrange the resulting vectors into a three-dimensional feature array. Let m the number of actions such that $m = |O|$, and k the number of patterns such that $k = |\mathcal{S}|$. The array $\mathbf{X}(s) \in \mathbb{R}^{k \times m \times d}$ stacks the per-action feature vector for every pattern-action pair in the order of the action and pattern enumeration (see Fig. 1). We use the same set of actions O for every P, thus the array has a consistent shape across patterns. The first dimension indexes the abstraction, the second ranges over the actions, and the third contains the AWL feature vector of the action node. This array is the input to the learned partition predictor described next.

3.3 Axial Self-Attention to predict Cost Partitions

Now that we have defined how to extract action-centric features into the tensor $\mathbf{X}(s)$, we present the model that transforms these features into admissible cost partitions.

The feature array $\mathbf{X}(s) \in \mathbb{R}^{k \times m \times d}$ produced by the AWL-based extraction encodes, for each abstraction and each ground action, a structural signature of how that action appears within the pattern projection at state s. To turn this information into a cost partition, we need a model that can simultaneously reason about two types of context: the relative importance of actions inside a single abstraction, and the role an action plays across different abstractions. Once the reasoning is done, the model needs to compute a cost weight for each action that reflects its significance across all abstractions. To enforce admissibility, those weights must respect the cost partition constraint (1), i.e., for every action the weights across abstractions must sum to at most its original cost. We propose to achieve this with an axial self-attention block (Ho et al. 2019) followed by a multi-layer perceptron (MLP) and a softmax output layer.

Axial self-attention Self-attention (Vaswani et al. 2017) computes, for each input embedding, a weighted sum over all embeddings, where the weights are derived via query–key similarity (e.g., softmax of scaled dot-products). This operation is inherently agnostic to sequence length and order because the attention pattern depends only on pairwise relationships, not on absolute positions or a fixed receptive field. This design allows a model trained on short sequences to be deployed on longer ones without architectural modification, enabling length extrapolation. Axial self-attention applies self-attention along distinct axes of the input array, allowing the model to contextualize the action features from two complementary perspectives.

1. **Action self-attention.** Along the action axis, for each abstraction $i \in \{1, \dots, k\}$ we treat the m action-node embeddings $\mathbf{X}(s)_{i,\cdot,\cdot} \in \mathbb{R}^{m \times d}$ as a sequence, and apply a standard self-attention mechanism. This allows the model to compare actions within the same abstraction, answering: *which actions in this pattern should receive a larger share of cost?*
2. **Abstraction self-attention.** Along the abstraction axis, for each action position $j \in \{1, \dots, m\}$ we slice the array as $\mathbf{X}(s)_{\cdot,j,\cdot} \in \mathbb{R}^{k \times d}$, and again apply self-attention. This contextualizes each action by looking at how it is represented across different patterns, answering: *in which kinds of abstractions is this action structurally important?*

We apply action self-attention first and abstraction self-attention on its output. The inputs of the two self-attention is passed through layer normalization. Outputs goes through a linear projection and a dropout layer before being fused with their input by element-wise addition. Final output is a contextualized array $\mathbf{H}$ with the same spatial dimensions as $\mathbf{X}(s)$.

Score matrix The contextualized array $\mathbf{H}$ is passed through then fed into a point-wise MLP applied independently to every (i, j) position. This MLP consist of two hidden layers with ReLU activations, followed by a dropout layer and a linear projection that reduces the feature dimension from d to 1. For each abstraction i and action j, the MLP outputs a scalar score $S_{i,j}$ resulting in a score matrix $S^{k \times m}$. A higher score indicates that the action is particularly salient for the abstraction, and therefore a larger portion of the original action cost should be allocated.

Admissible cost partition Finally, we convert the scores into valid cost-partition weights. For each action j, we interpret the vector $S_{\cdot,j}$ as the relative importance of the k patterns for that action at state s. To turn these scores into a proper allocation, we apply the softmax function over the pattern dimension:

$$\alpha_{i,j} = \frac{e^{S_{i,j}}}{\sum_{i'=1}^{k} e^{S_{i',j}}}$$

The softmax properties guarantee $\sum_{i=1}^{k} \alpha_{i,j} = 1, \forall j \in 1, \dots, m$ which is essential for admissibility. These coefficients can be interpreted as the fraction of the original cost of action $o_j \in O$ that should be assigned to pattern i. The predicted cost is then obtained by multiplying the original action cost by the coefficient $C_i(o_j) = \alpha_{i,j} \cdot c(o_j)$. Since the coefficients sum to one, we immediately satisfies the cost partition constraint (1) as an equality.

Loss function During training, we have access to the optimal cost partition for s, computed offline by solving OCP LP. This gives a target allocation matrix $\alpha^* \in [0, 1]^{k \times m}$, where $\alpha^*_{i,j}$ is the fraction of $c(o_j)$ that the optimal partition assigns to pattern i.

We train the model to match this optimal behaviour by minimising the Kullback–Leibler (KL) divergence (Kullback and Leibler 1951) between the target and predicted distributions, summed over all actions:

$$\mathcal{L}(s) = \sum_{j=1}^{m} \sum_{i=1}^{k} \alpha^*_{i,j} \log \frac{\alpha^*_{i,j}}{\alpha_{i,j}}$$

Minimising $\mathcal{L}(s)$ encourages the network to allocate cost in the same proportion as the optimal partition.

Training loop overview We train the model offline on synthetic planning tasks, using fixed training and validation splits where the validation tasks are larger than those used for training. For each instance, a sampler uses random walks from the initial state to draw valid states and computes $\alpha^*_{i,j}$ via LP. We implement a non-negative OCP LP (Pommerening et al. 2014) with a modified cost partition constraint (1):

$$\sum_{P \in \mathcal{S}} c^P(o) = c(o), \forall o \in O$$

to extract weights compatible with a softmax output. For each mini-batch of states, we compute the loss and backpropagate through the neural network. AWL feature extraction is not updated. After each epoch, we evaluate the loss on the validation set.

4 Experiments

In this section, we empirically evaluate our framework for learning domain-specific admissible cost partition heuristics on a subset of the benchmarks from the optimal track of the International Planning Competition (IPC). We train on optimal cost weights from sampled states of synthetic tasks described in Table 2. We generate samples for each split using Scorpion (Seipp, Keller, and Helmert 2020) with 500 maximum samples per task and 30min cutoff time. We use the set of all projections up to 2 variables and limited to interesting patterns (Pommerening, Röger, and Helmert 2013) with at least one goal variable.

We consider a combination of two hyperparameters in our configurations: graph representation $R \in \{\text{FLG}, \text{AOAG}\}$ and AWL iterations $L \in \{1, 2\}$. We train our model with batch size 4, initial learning rate of 10^{-5} and AdamW (Loshchilov and Hutter 2019) optimizer with weight decay of 0.01 for a maximum of 100 epochs. We apply early stopping if the validation loss does not improve by at least 10^{-4} for 15 epochs. The learning rate also decreases by a factor of 10 if the minimal validation loss did not decrease in the last 5 epochs. Hidden layers in pointwise MLP compress the embedding dimension d by a factor of 0.4 before restoring the original size right before the linear projection. All other learning layers have a dimension equal to d. The dropout rate is 0.1 for the scalar dot-product attention mechanism in axial attention, 0.2 for dropout layers after both axial attention and 0.3 before the MLP. Models are trained using PyTorch 2.11 on a cluster with 4 Intel Xeon Gold 6448Y cores, 62Go RAM and $\frac{3}{8}$th of the computing power of an NVidia H100 SXM5 with 40GB GPU memory.

For final evaluation, we build an additive heuristic $h^{\text{LCP}}(s)$ on top of the predicted costs C such that $h^{\text{LCP}}(s) = \sum_{P_i \in \mathcal{S}} h^{P_i}(s, C_i)$. We write $h^{\text{LCP}}_{\text{R-L}}$ for the learned heuristic

Domain	Train	#	Validation	#
blocks	[4, 9]	27	[10]	6
ferry	[2, 8]	25	[9]	4
logistics	[2, 6]	30	[7]	6
miconic	[3, 11]	32	[12]	4
spanner	[2, 9]	40	[10]	5
visitall	[2, 7]	35	[8]	6

Table 2: Synthetic problem splits with corresponding numbers of tasks per domain.

	blocks (35)	ferry (30)	logistics (63)	miconic (150)	spanner (30)	visitall (40)	**Total** (348)
h^{GZOCP}	**28**	13	27	70	**30**	30	198
h^{SCP_+}	**28**	**22**	26	**146**	**30**	**33**	**285**
h^{OCP_+}	17	11	**33**	50	**30**	16	157
$h^{\text{LCP}}_{\text{FLG-1}}$	19	11	17	40	**30**	20	137
$h^{\text{LCP}}_{\text{FLG-2}}$	18	11	18	40	**30**	20	137
$h^{\text{LCP}}_{\text{AOAG-1}}$	17	12	15	41	**30**	18	133
$h^{\text{LCP}}_{\text{AOAG-2}}$	18	12	18	42	28	16	134

Table 3: Coverage results per domain. Numbers in brackets shows the total number of instances per column. Best results among admissible configurations are highlighted in bold.

on graph representation R with L iterations. Our heuristic predicts a new cost partition at every evaluated state and only uses this new partition to compute the heuristic value. The learned heuristic is integrated into the Fast Downward (FD) planning system (Helmert 2006) as a separate heuristic evaluator using the LibTorch C++ CUDA library. We benchmark on the standard IPC optimal track instances for the same domains as training. Since Ferry and Spanner are not part of the optimal track, we use the easy testing task from the 2023 IPC learning track. We use three cost partitioning heuristics implemented in Scorpion as baselines: Greedy zero-one (h^{GZOCP}), Non-negative online saturated (h^{SCP_+}) and Non-negative optimal (h^{OCP_+}). h^{GZOCP} and h^{SCP_+} both use a greedy ordering of the patterns, h^{SCP_+} computes a new cost partition at every evaluated state. All methods are evaluated using A* and a timeout of 1800 seconds. Baselines are run on a cluster with single Intel Xeon Gold 6448Y core and 32Go RAM. Our learned heuristics also use $\frac{3}{8}$th of the computing power of an NVidia H100 SXM5 with 40GB GPU memory on the same cluster. In our current implementation, the GPU is only used to speedup the learning model prediction of our framework.

Table 3 reports the number of tasks solved within the resource limits and Figure 3 the number of nodes expanded by $h^{\text{LCP}}_{\text{FLG-2}}$ vs the baselines. In the remainder of this section, we discuss our results and answer the following questions.

How well do our learned heuristics perform? Full coverage results are given in Table 3. On every instance solved by both a learned heuristic and an admissible baseline, the resulting plan has the same cost, empirically confirming the admissibility guarantee of Section 3.3. Our learned configurations achieve competitive coverage with h^{OCP_+} on most domains; the exceptions are logistics and miconic, where they fall behind. On all domains, however, the learned heuristics are outperformed by the faster h^{SCP_+} and h^{GZOCP} baselines because their higher per-state evaluation cost outweighs the gain in node expansions. We attribute the poor performance on logistics and miconic, at least in part, to the absence of static atoms from our graph representation, as both domains rely heavily on such static relations. Our graphs may therefore be insufficiently expressive in these settings, preventing the model from inferring appropriate cost partitions and leading to weak heuristic values on larger problems.

Are our heuristics computationally efficient? Another downside of our method is the evaluation time. For solved tasks, an average of only 3.42% of the evaluation time is spent on partition weight prediction thanks to our heavy GPU parallelization. This places the evaluation speed bottleneck on feature extraction and abstraction heuristic computation. In our implementation, feature generation and abstraction heuristic computation do not benefit from the available GPU. This leads to a considerable amount of time being dedicated to those steps before acquiring the heuristic value.

How informative is our learned heuristic compared to suboptimal partitions? Figure 3 compare the number of node expansions for the baselines and $h^{\text{LCP}}_{\text{FLG-2}}$, our most effective learned configuration on this metric. The left-hand plot shows that our method requires fewer expansions than h^{GZOCP} for the majority of commonly solved tasks, which confirms that the learning is effective. Plots with h^{SCP_+} and h^{OCP_+} show more mixed results. Nevertheless, a fair amount of tasks show a number of expanded nodes on par with h^{OCP_+}.

5 Conclusion and Future Works

We introduced a learning framework that infers admissible cost partitions for optimal planning. By exploiting the equivalence between optimal cost partitioning and Lagrangian duals, our method replaces the expensive per-state linear program with a learning model that predicts state-aware partition weights. The pipeline encodes a planning state and its pattern abstractions as a labelled graph, extracts action-centric WL features and uses axial self-attention with a softmax output to produce weights that satisfy the sum constraint by construction, guaranteeing admissibility.

Experiments on six IPC domains demonstrate that the learned partitions can reduce node expansions compared to the greedy zero-one and saturated suboptimal baselines. On the four domains where the model succeeds, coverage is competitive with the optimal cost partition. However, the current implementation suffers from feature-extraction and inference costs that make it slower than the suboptimal baselines and a lack of expressiveness on some domains, limiting its overall advantages.

Several directions can address the current limitations and extend our current work. A more expressive graph representation could help our model generalize to larger problems.

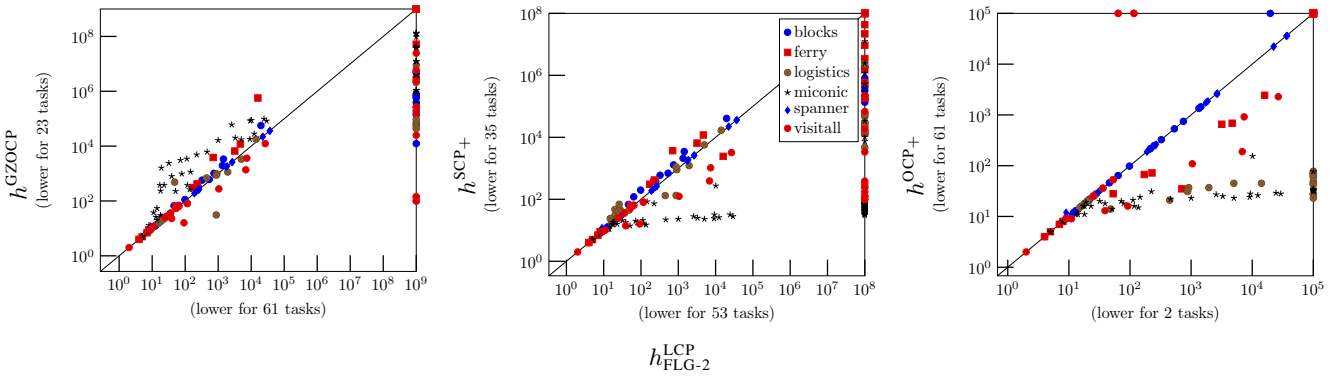


Figure 3: Number of expanded nodes of h^{GZOCP}, h^{SCP_+}, h^{OCP_+} and $h^{\text{LCP}}_{\text{FLG-2}}$. On each plots, unsolved task by one planner has their respective metric set to the axis limit. Points on the top left triangle favour $h^{\text{LCP}}_{\text{FLG-2}}$ while points on the bottom right triangle favour baselines.

Our architecture is currently trained on optimal cost partitions, but an optimal partition is only one of many possible ways to achieve an informative heuristic value. The training pipeline could therefore be adjusted to incorporate the desired heuristic value directly as an additional label alongside the partition weights, so that the model learns to prefer partitions that not only mimic the optimal allocation but also maximize the resulting bound. Another adjustment could be made to our way of exploring the state-space of our training instances to extract optimal partitions. When the space and possible actions grows, the random walk might drift away from the optimal plan toward less relevant states and hinder training. As we have noted, most of the evaluation time is not spent on model inference; speeding up feature extraction and abstraction heuristic computation could thus alleviate the main bottleneck. Both steps are currently implemented on CPU, whereas they are heavily parallelizable. Moving them to the GPU could substantially reduce their runtime. Finally, other strategies such as interval-based online partitioning (Seipp 2021) could further improve speed if the predicted cost partition is sufficiently robust.

6 Acknowledgements

We thank the the anonymous reviewers for their helpful suggestions. Sylvie Thiébaux was funded by the Australian Research Council (ARC) under the Discovery Project grant DP220103815 and by the Artificial and Natural Intelligence Toulouse Institute (ANITI) under the grant agreement ANR-23-IACL-0002.

References

Abbas, A.; and Swoboda, P. 2024. DOGE-Train: Discrete Optimization on GPU with End-to-End Training. In *Proc. 38th AAAI Conference on Artificial Intelligence (AAAI)*, 20623–20631.

Bessa, S.; Dabert, D.; Bourgeat, M.; Rousseau, L.; and Cappart, Q. 2025. Learning Valid Dual Bounds in Constraint Programming: Boosted Lagrangian Decomposition with Self-Supervised Learning. In *Proc. 39th AAAI Conference on Artificial Intelligence (AAAI)*, 11113–11121.

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

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

Chen, D. Z.; Zenn, J.; Cinquin, T.; and McIlraith, S. A. 2025. Language Models For Generalised PDDL Planning: Synthesising Sound and Programmatic Policies. In *Proc. 18th European Workshop on Reinforcement Learning (EWRL)*.

Corrêa, A. B.; Pereira, A. G.; and Seipp, J. 2025. Classical Planning with LLM-Generated Heuristics: Challenging the State of the Art with Python Code. In *proc. 38th Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Ernandes, M.; and Gori, M. 2004. Likely-Admissible and Sub-Symbolic Heuristics. In *Proc. 16th Euroopean Conference on Artificial Intelligence (ECAI)*, 613–617.

Futuhi, E.; and Sturtevant, N. 2026. Learning Admissible Heuristics for A*: Theory and Practice. In *Proc. 14th International Conference on Learning Representations (ICLR)*.

Garrett, C. R.; Kaelbling, L. P.; and Lozano-Pérez, T. 2016. Learning to Rank for Synthesizing Planning Heuristics. In *Proc. 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 3089–3095.

Hao, M.; Chen, D. Z.; Trevizan, F. W.; and Thiébaux, S. 2025. Effective Data Generation and Feature Selection in Learning for Planning. In *Proc. 28th European Conference on Artificial Intelligence (ECAI)*, 4969–4976.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New Admissible Heuristics for Domain-Independent Planning. In *Proc. 20th American National Conference on Artificial Intelligence (AAAI)*, 1163–1168.

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

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, 173(5-6): 503–535.

Ho, J.; Kalchbrenner, N.; Weissenborn, D.; and Salimans, T. 2019. Axial Attention in Multidimensional Transformers. arXiv:1912.12180.

Horcík, R.; Sír, G.; Simek, V.; and Pevný, T. 2025. State Encodings for GNN-Based Lifted Planners. In *Proc. 39th AAAI Conference on Artificial Intelligence, Thirty (AAAI)*, 26525–26533. AAAI Press.

Karpas, E.; and Domshlak, C. 2009. Cost-Optimal Planning with Landmarks. In *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 1728–1733.

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

Kullback, S.; and Leibler, R. A. 1951. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1): 79–86.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. arXiv:1711.05101.

Musayev, F.; Drexler, D.; Gnad, D.; and Seipp, J. 2025. Combining Heuristics and Transition Classifiers in Classical Planning. In *Proc. 28th European Conference on Artificial Intelligence (ECAI)*, 4694–4701.

Ofir Marom, O.; and Rosman, B. 2020. Utilising Uncertainty for Efficient Learning of Likely-Admissible Heuristics. In *Proc. 30th International Conference on Automated Planning and Scheduling (ICAPS)*, 560–568.

Parjadis, A.; Cappart, Q.; Dilkina, B.; Ferber, A.; and Rousseau, L.-M. 2024. Learning Lagrangian Multipliers for the Travelling Salesman Problem. In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, 22–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From Non-Negative to General Operator Cost Partitioning. In *Proc. 29th AAAI Conference on Artificial Intelligence (AAAI)*, 3335–3341.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the Most Out of Pattern Databases for Classical Planning. In *Proc. 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2357–2364.

Pommerening, F.; Röger, G.; Helmert, M.; Cambazard, H.; Rousseau, L.; and Salvagnin, D. 2019. Lagrangian Decomposition for Optimal Cost Partitioning. In *Proc. 29th International Conference on Automated Planning and Scheduling (ICAPS)*, 338–347.

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

Seipp, J. 2021. Online Saturated Cost Partitioning for Classical Planning. In *Proc. 31st International Conference on Automated Planning and Scheduling (ICAPS)*, 317–321.

Seipp, J.; Keller, T.; and Helmert, M. 2017. A Comparison of Cost Partitioning Algorithms for Optimal Classical Planning. In *Proc. 27th International Conference on Automated Planning and Scheduling (ICAPS)*, 259–268.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated Cost Partitioning for Optimal Classical Planning. *J. Artif. Intell. Res.*, 67: 129–167.

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

Shervashidze, N.; Schweitzer, P.; van Leeuwen, E. a.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-Lehman Graph Kernels. *J. Mach. Learn. Res.*, 12: 2539–2561.

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 *Proc. 38th AAAI Conference on Artificial Intelligence (AAAI)*, 20256–20264.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022a. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In *Pro. 32nd International Conference on Automated Planning and Scheduling (ICAPS)*, 629–637.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022b. Learning Generalized Policies without Supervision Using GNNs. In *Proc. 19th International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Toyer, S.; Thiébaux, S.; Trevizan, F. W.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *J. Artif. Intell. Res.*, 68: 1–68.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Proc. 38th Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Wang, R. X.; and Trevizan, F. 2025. Leveraging Action Relational Structures for Integrated Learning and Planning. In *Proc. 35th International Conference on Automated Planning and Scheduling (ICAPS)*.