ECAI 2025
I. Lynce et al. (Eds.)

© 2025 The Authors.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/FAIA251370

Weisfeiler-Leman Features for Planning: A 1,000,000 Sample Size Hyperparameter Study

Dillon Z. Chen

LAAS-CNRS, University of Toulouse, France

Abstract. Weisfeiler-Leman Features (WLFs) are a recently introduced classical machine learning tool for learning to plan and search. They have been shown to be both theoretically and empirically superior to existing deep learning approaches for learning value functions for search in symbolic planning. In this paper, we introduce new WLF hyperparameters and study their various tradeoffs and effects. We utilise the efficiency of WLFs and run planning experiments on single core CPUs with a sample size of 1,000,000 to understand the effect of hyperparameters on training and planning. Our experimental analysis show that there is a robust and best set of hyperparameters for WLFs across the tested planning domains. We find that the best WLF hyperparameters for learning heuristic functions minimise execution time rather than maximise model expressivity. We further statistically analyse and observe no significant correlation between training and planning metrics.

1 Introduction

Learning to plan has gained significant interest in recent years due to the advancements of machine learning approaches across various fields, and the desire to construct autonomous systems that can generalise in long horizon decision making problems. An aim of learning to plan involves designing automated, domain-independent algorithms for learning domain knowledge in an inductive manner from small training problems that *generalise* and scales up planning to problems of very large sizes [36, 41, 24, 70, 60, 61, 15, 52, 34, 10, 28, 66]. Indeed, real-world planning problems do not exhibit much training data for autonomous systems to learn from, meaning that the development of algorithms that can generalise from small training problems is indispensable.

A recently introduced approach involves learning heuristic functions using Weisfeiler-Leman Features (WLFs) automatically extracted from graph representations of planning tasks [11]. The approach involves (1) transforming planning tasks into graphs, and (2) embedding such graphs into feature vectors via the Weisfeiler-Leman (WL) algorithm. It yields cheap to learn heuristic functions that perform favourably compared to both traditional domain-independent heuristics and learned neural heuristics for planning. Its performance can be attributed to its faster evaluation speed and greater expressive power of the WL algorithm compared to existing learning for planning work that use neural networks.

In this work, we introduce various extensions and hyperparameters of WLFs with various tradeoffs between model expressivity, generalisation, and execution speed. Furthermore, we perform large scale experiments resulting in over 1,000,000 planning runs in order to rigorously understand the empirical effects of various hyperparameter settings on (1) training, (2) planning, and (3) the correlation between training and planning metrics. Our findings are positive and provide us a robust, best set of go-to hyperparameters for generating WLFs for planning. We observe that the best WLF hyperparameters for learning heuristic functions aim to minimise model size and execution time rather than maximise model expressivity. Furthermore, we identify that there is no statistically significant, strong correlation between various training metrics and planning performance of WLF models for heuristic search.

2 Related Work

The field of learning to plan has been tackled by various different approaches. In this section we cover more recent approaches for learning to plan and how our approach differs. We refer to the survey by Celorrio et al. [7] for earlier works in the field.

Deep and Reinforcement Learning It is no surprise that there is a large body of recent work employing deep or reinforcement learning approaches for learning to plan given their progress in various fields. Toyer et al. [60, 61] and Dong et al. [15] introduced the first works employing deep learning for symbolic planning via Graph Neural Networks (GNNs) for learning policies that can generalise to instances of arbitrary size. Later works employed deep learning architectures to learn heuristics to guide search in a domain-dependent [27, 34] and domain-independent [52, 10] heuristics. Expressivity limits of GNNs [44, 68, 2] were exploited to show that deep learning approaches cannot learn optimal domain knowledge for planning [57, 10]. Reinforcement learning has also been used for learning heuristics [42, 22] and policies [58], and transformers for learning policies [47].

Large Language Models Large Language Models have been used to perform one-shot planning via prompting with low success [63, 64]. They have also been used to perform heuristic evaluation during search but this has been shown again to be inefficient and incomplete [35]. However, they have shown preliminary success in certain domains for *generating* solvers [54] or heuristic functions [62, 13] for search as code.

Generalised Planning Generalised Planning (GP) refers to a class of approaches which learn interpretable and symbolic general policies that can loop [39, 55, 56, 32]. Works in GP learn such policies by generating nondeterministic abstractions of infinite sets of planning instances [55, 33, 5, 4, 14] for which a solution is a solution to the ground set of instances. Abstractions and policies can be synthesised and learned from features [5, 18] or found via search [48, 49, 69, 38, 50].

Our work differs from deep learning and large language model approaches as we instead employ *classical* and *statistical* learning approaches which are more efficient in terms of learning and evaluation than deep approaches and have the additional benefit of being interpretable. We also differ from *policy* approaches in GP that are often complete only under certain assumptions on the planning domain, whereas by employing *search* our approach maintains completeness over planning problems with finite state spaces. Lastly, our experiments are extensive in terms of scale to determine the major factors contributing to the performance of our approach.

3 Background

This section provides the formal definitions of planning tasks and graphs required for understanding the Weisfeiler-Leman Features (WLF) [11] used for learning, planning and search. For the sake of brevity, we focus our attention on classical, lifted planning tasks. However, we note that WLFs can also handle numeric planning tasks [9]. Furthermore, they are state-centric meaning that they are agnostic to action effects and hence can be extended to handle probabilistic tasks [72]. Let $[\![n]\!]$ denote the set of integers $\{1, \dots, n\}$.

Planning Task A classical planning task is a deterministic state transition model [21] given by a tuple $\mathbf{P} = \langle S, A, s_0, G\rangle$ where S is a set of states, A a set of actions, $s_0 \in S$ an initial state, and $G \subseteq S$ a set of goal states. Each action $a \in A$ is a function $a : S \to S \cup \{\bot\}$ where $a(s) = \bot$ if a is not applicable in s, and $a(s) \in S$ is the successor state when a is applied to s. A solution for a planning task is a plan: a sequence of actions $\pi = a_1, \dots, a_n$ where $s_i = a_i(s_{i-1}) \neq \bot$ for $i \in [\![n]\!]$ and $s_n \in G$. A state s in a planning task $\mathbf{P}$ induces a new planning task $\mathbf{P}' = \langle S, A, s, G\rangle$. A planning task is solvable if there exists at least one plan.

Lifted Representation Planning tasks are often compactly formalised in a lifted representation using predicate logic, such as via PDDL [23, 30]. More specifically, a lifted planning task is a tuple $\mathbf{P} = \langle \mathcal{O}, \mathcal{P}, \mathcal{A}, s_0, \mathcal{G}\rangle$, where $\mathcal{O}$ denotes a set of objects, $\mathcal{P}$ a set of predicate symbols, $\mathcal{A}$ a set of action schemata, s_0 the initial state, and $\mathcal{G}$ the goal condition. We define a domain to be a set of lifted planning tasks which share the same set of predicates $\mathcal{P}$ and action schemata $\mathcal{A}$. Understanding of the transition system induced by $\mathcal{A}$ is not required for the sake of this paper, so we instead focus on the representation of states and the goal condition next.

Each symbol $P \in \mathcal{P}$ is associated with an arity $\text{ar}(P) \in \mathbb{N} \cup \{0\}$. Predicates take the form $P(x_1, \dots, x_{\text{ar}(P)})$, where the x_is denote their arguments. Propositions are defined by substituting objects into predicate arguments. More specifically, given $P \in \mathcal{P}$, and a tuple of objects $\mathbf{o} = \langle \mathbf{o}_1, \dots, \mathbf{o}_{\text{ar}(P)}\rangle$, we denote $P(\mathbf{o})$ as the proposition defined by substituting $\mathbf{o}$ into arguments of P. A state s is a set of propositions. The goal condition $\mathcal{G}$ also consists of a set of propositions, and a state s is a goal state if $s \supseteq \mathcal{G}$.

Graphs We denote a graph with categorical node features and edge labels by a tuple $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{F}, \mathbf{L}\rangle$. We have that $\mathbf{V}$ is a set of nodes, $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ a set of edges, $\mathbf{F} : \mathbf{V} \to \Sigma_{\text{V}}$ the categorical node features, and $\mathbf{L} : \mathbf{E} \to \Sigma_{\text{E}}$ the edge labels, where Σ_{V} and Σ_{E} are finite sets of symbols. The neighbourhood of a node $u \in \mathbf{V}$ in a graph is defined by $\mathbf{N}(u) = \{v \in \mathbf{V} \mid \langle u, v\rangle \in \mathbf{E}\}$. The neighbourhood of a node $u \in \mathbf{V}$ with respect to an edge label ι is defined by $\mathbf{N}_\iota(u) = \{v \in \mathbf{V} \mid e = \langle u, v\rangle \in \mathbf{E} \wedge \mathbf{L}(e) = \iota\}$.

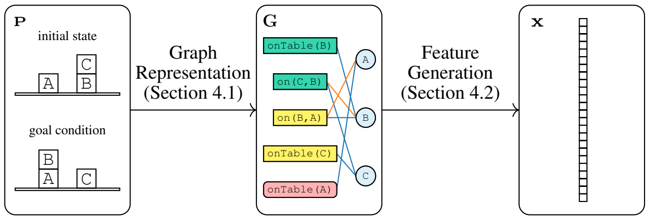

Figure 1. The WL Feature pipeline for a Blocksworld instance (`clear` propositions omitted).

4 WL Features for Planning

In this section, we describe the procedure for generating vector features for planning tasks without the need for training labels. Figure 1 summarises the pipeline which involves converting planning states into graphs, and applying a Weisfeiler-Leman (WL) algorithm to generate the WLFs.

4.1 Graph Representation

The first component of the WLF pipeline involves the transformation of planning tasks into graphs. Graphs with edge features are viewed as 'binary relational structures' in other communities, from which we can derive relational features with various sorts of algorithms. In this section, we describe the Instance Learning Graph (ILG) [11] for representing classical planning tasks.

The middle image in Figure 1 illustrates a subgraph of the ILG encoding of a simple Blocksworld problem in the left image. Nodes of the ILG represent the objects coloured in blue, goal conditions, and state information of the planning task, with colours encoding the semantics of nodes. More specifically, green nodes represent propositions in the state but not part of the goal condition, yellow nodes represent goal conditions that have not yet been achieved in the current state, and red nodes represent goal conditions that have been achieved. Edges connect objects to propositions that are predicates instantiated with the object, and edge labels encode the location of predicates in which objects are instantiated. In the image, blue/orange edges connect variable nodes to the object that is instantiated in the first/second argument. The formal definition is as follows.

Definition 4.1. The Instance Learning Graph (ILG) of a lifted planning task $\mathbf{P} = \langle \mathcal{O}, \mathcal{P}, \mathcal{A}, s_0, \mathcal{G}\rangle$ is a graph with categorical node features and edges labels $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{F}, \mathbf{L}\rangle$ where

- nodes $\mathbf{V} = \mathcal{O} \cup s_0 \cup \mathcal{G}$,
- edges $\mathbf{E} = \bigcup_{p=P(\mathbf{o}) \in s_0 \cup \mathcal{G}} \{\langle p, \mathbf{o}_i\rangle \mid i \in [\![\text{ar}(P)]\!]\}$,
- categorical node features $\mathbf{F} : \mathbf{V} \to \Sigma_{\text{V}}$ defined by

$$\mathbf{F}(u) = \begin{cases} \texttt{object} & \text{if } u \in \mathcal{O} \\ (\texttt{pred}(u), \texttt{ag}) & \text{if } u \in s_0 \cap \mathcal{G} \\ (\texttt{pred}(u), \texttt{ug}) & \text{if } u \in \mathcal{G} \setminus s_0 \\ (\texttt{pred}(u), \texttt{ap}) & \text{if } u \in s_0 \setminus \mathcal{G} \end{cases}$$

  where $\texttt{pred}(u)$ denotes the predicate symbol of a proposition u. We note that `object` and `ag`, `ug`, `ap` are constant symbols that are agnostic to the planning domain[1].
- edge labels $\mathbf{L} : \mathbf{E} \to \mathbb{N}$ defined by $\langle p, \mathbf{o}_i\rangle \mapsto i$.

In general, the maximum number of categorical node features in the ILG of any problem for a domain with predicates $\mathcal{P}$ is $|\Sigma_{\text{V}}| = 1 + 3\,|\mathcal{P}|$ and the number of edge labels is equal to the maximum predicate arity.

[1] Standing for achieved goal, unachieved goal, and achieved propositional nongoal, respectively.

Algorithm 1: WL algorithm

```
Input: A graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{F}, \mathbf{L} \rangle$, injective HASH function,
       and number of iterations $L$.
Output: Multiset of colours.
1 $c^0(v) \leftarrow \mathbf{F}(v), \forall v \in \mathbf{V}$
2 for $l = 1, \ldots, L$ do for $v \in \mathbf{V}$ do
3     $c^l(v) \leftarrow$
          $\textsc{hash}\left(c^{l-1}(v), \bigcup_{\iota \in \Sigma_{\mathrm{E}}} \{\!\{(c^{l-1}(u), \iota) \mid u \in \mathbf{N}_\iota(v)\}\!\}\right)$
4 return $\bigcup_{l=0,\ldots,L} \{\!\{c^l(v) \mid v \in \mathbf{V}\}\!\}$
```

4.2 Feature Generation

The second component of the WLF pipeline involves transforming graph representations of planning tasks into feature vectors for use with any downstream task. The go-to example is learning heuristic functions for heuristic search as we study in this paper.

The algorithms for feature generation of graphs are generally some extension of the colour refinement algorithm, a special case of the general k-Weisfeiler-Leman algorithm [67, 6]. In this section, we describe the colour refinement, or 1-Weisfeiler-Leman (WL) algorithm, followed by how the algorithm is used for constructing feature vectors from graphs.

The WL Algorithm The underlying concept of the WL algorithm is to iteratively update node colours based on the colours of their neighbours. The original WL algorithm was designed for graphs without edge labels. We present the WL algorithm which can support edge labels [3] in Algorithm 1. The algorithm's input is a graph with node features and edge labels as described in Section 3, alongside a hyperparameter L determining how many WL iterations to perform. The output of the algorithm is a canonical form for the graph that is invariant to node orderings.

Line 1 initialises node graph colours as their categorical node features. Lines 2 and 3 iteratively update the colour of each node v in the graph by collecting all its neighbours and the corresponding edge label (u, ι) into a multiset. This multiset is then hashed alongside v's current colour with an injective function to produce a new refined colour. In practice, the injective hash function is built lazily, where every time a new multiset is encountered, it is mapped to a new, unseen hash value. After L iterations, the multiset of all node colours seen throughout the algorithm is returned.

Embedding graphs The WL algorithm has been used to generate features for the WL graph kernel [53]. Each node colour constitutes a feature, and its value for a graph is the count of the number of nodes that exhibit or has exhibited the colour. Then given a set of colours $\mathbf{C}$ known a priori, the WL algorithm can return a fixed sized feature vector of size $|\mathbf{C}|$ for every input graph. In a learning for planning pipeline, we collect the colours $\mathbf{C}$ from a set of training planning tasks, followed by using the colours to embed arbitrary graphs (i.e. converted from either training or testing tasks) into fixed sized feature vectors from such colours. The steps are formalised as follows.

1. We construct $\mathbf{C}$ from a given set of graph representations of planning tasks $\mathbf{G}_1, \ldots, \mathbf{G}_m$ by running the WL algorithm, with the same HASH function and number of iterations L, on all of them and then taking the set union of all multiset outputs, i.e. $\mathbf{C} = \bigcup_{i \in [\![m]\!]} \mathrm{WL}(\mathbf{G}_i)$.
2. Now suppose we have collected a set of colours and enumerated them by $\mathbf{C} = \{c_1, \ldots, c_{|\mathbf{C}|}\}$. Then given a graph $\mathbf{G}$ and its multiset output from the WL algorithm $\mathbf{M}$, we can define an embedding of the graph into Euclidean space by the feature vector

$$[\textsc{count}(\mathbf{M}, c_1), \ldots, \textsc{count}(\mathbf{M}, c_{|\mathbf{C}|})] \in \mathbb{R}^{|\mathbf{C}|}, \tag{1}$$

where $\textsc{count}(\mathbf{M}, c_i)$ is an integer which counts the occurrence of the colour c_i in $\mathbf{M}$. We note importantly that any colours returned in $\mathbf{M}$ that are not in $\mathbf{C}$ are defined as *unseen* colours and are entirely ignored in the output.

We can view colours as features, i.e. functions that map planning tasks to real values, by $c_i(\mathbf{P}) = \textsc{count}(\mathbf{M}, c_i)$ where $\mathbf{M}$ is the multiset output of WL on $\mathbf{P}$ encoded to a graph such as via the ILG.

5 WL Feature Hyperparameters

In this section, we introduce the various hyperparameters available in a learning for planning pipeline employing WLFs. We describe hyperparameters specific to WLFs as internal (Section 5.1) and those that are general to a learning for planning pipeline as external (Section 5.2). The hyperparameters we cover are summarised in Table 1.

5.1 Internal Hyperparameters

WL Algorithm

The WL algorithm from Section 4.2 is the canonical graph kernel baseline for graph learning tasks due to the theoretical result that it upper bounds distinguishing power of the message passing GNN architecture [44, 68]. It is also an efficient algorithm that runs in low polynomial time in the input graph and considered the first choice to apply to graphs as described in the extensive graph kernel survey by Kriege et al. [37]. Nevertheless, the graph learning community has proposed various extensions of the WL algorithm and corresponding GNN architectures that have provably more distinguishing power than the WL algorithm, and yet are still computationally feasible.

Most notably, it is well known that the $(k + 1)$-WL algorithm is strictly more powerful than the k-WL algorithm for $k \geq 2$ but its runtime scales exponentially in k. Thus, many extensions of the WL algorithm and corresponding GNNs have been proposed to either approximate or achieve orthogonal expressiveness of higher order WL algorithms [45, 46, 73, 65, 1]. Furthermore, graph kernels have been proposed that also handle graphs with continuous node attributes [9]. We have implemented some of these WL extensions alongside completely new graph kernels in the current version of WLPlan which we outline as follows. Figure 2 illustrates the expressivity hierarchy of mentioned WL algorithms.

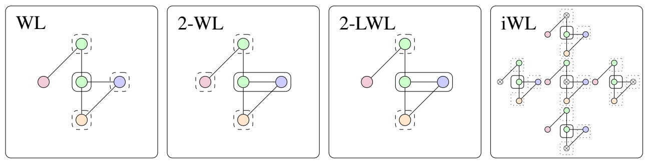

Figure 2. Visualisations of neighbours (dotted) of the center node or node pair (solid) in the WL, 2-WL, 2-LWL and iWL algorithms. In iWL, the WL algorithm is run $|\mathbf{V}|$ times, where each time a different node is individualised with a special colour.

2-WL The k-WL algorithms are a suite of incomplete graph isomorphism algorithms which have a one-to-one correspondence to k-variable counting logics [6]. However, the k-WL algorithms scale exponentially in k, with the 2-WL algorithm exhausting memory limits

	Setting	**Description**	**E**	**G**	**S**
Internal	WL Algorithm				
	`WL`	The vanilla colour refinement algorithm.	–	–	–
	`iWL`	WL extended with individualisation but incurs an additional runtime cost as WL is repeated for each node in the graph.	↑	↓	↓
	`niWL`	Equivalent to `iWL` except that node features are normalised by the number of node features.	↑	↓	↓
	`2-LWL`	A feasible approximation of 2-WL but still incurs the worst case runtime cost which is quadratic in the number of nodes.	↑	↓	↓
	`2-WL`	WL extended to pairs of nodes. Computationally infeasible both runtime and memory-wise for the tested datasets.	↑↑	↓↓	↓↓
	Iterations				
	4	An arbitrarily chosen number in the original WLF paper that works well for optimal heuristic estimators.	–	–	–
	Low (< 4)	Trades expressivity for improved speed and model size.	↓	↑	↑
	High (> 4)	Trades speed and generalisation for improved expressivity.	↑	↓	↓
	Feature Pruning				
	`none`	No pruning of collected features.	–	–	–
	`i-mf`	Combination of MaxSAT and frequency counting for pruning.	↓	↑	↑
	Hash Function				
	`multiset`	The hash input in the original WL algorithm which corresponds to collecting subtrees of a graph.	–	–	–
	`set`	Collapses duplicate neighbour colours with the aim of reducing the number of unseen colours outside of training.	↓	↑	↑
External	State Representation				
	`part`	Prunes propositions deemed static or unreachable by Fast Downward before generating WLFs for faster generation.	↓	–	↑
	`cmpl`	Uses all propositions from each state for generating WLFs. Does not lose information but is slower to generate.	↑	–	↓
	Optimiser				
	`Lasso`	Linear Regression with L1 prior for predicting optimal heuristics.			
	`GPR`	Gaussian Process Regression for predicting optimal heuristics.			
	`SVR`	Support Vector Regression for predicting optimal heuristics.			
	`rkLP`	Linear Programs for predicting ranking heuristics.			
	`rkGPC`	Gaussian Process Classification for predicting ranking heuristics.			
	`rkSVM`	Support Vector Machines for predicting ranking heuristics.			

Table 1. A summary of various WLF hyperparameters and descriptions. Default values of internal features are marked as so.

on medium sized graph datasets. We describe and implement the 2-WL algorithm and refer to [6, Section 5, page 13] and [25, Section 5, page 4] for the general k-WL algorithm.

The idea of the 2-WL algorithm, outlined in Algorithm 2, is to now assign and refine colours to ordered pairs of nodes. The algorithm begins in Lines 1-3 by assigning all possible ordered pairs of nodes a tuple of the node colours as well as the edge label between them. If there is no edge between a pair of nodes, a special $\perp$ value is used as the edge label. Lines 4-5 then iteratively refine the colour of each node pair by defining the neighbours of a pair (v, u) to be a *multiset of sequence of pairs*[2] $((w, u), (v, w))$ where w ranges over all nodes in the graph. Then the algorithm applies the colouring function of the current iteration to all node pairs to create a multiset of tuples of colours which are then hashed alongside the current node pair's colour. Finally, the algorithm returns the multiset of all node pair colours seen throughout the algorithm in Line 6. To generalise to the k-WL algorithm, one replaces node pairs with node k-tuples.

2-LWL The k-LWL algorithms [43] provide efficient approximations of the k-WL algorithms but still have the same worst case computational complexity. The main approximations made are that node tuples are converted to node sets, reducing the number of possible node tuples to consider per iteration by a constant factor ($n^k \to \binom{n}{k}$), and relaxing the definition of neighbours of node tuples. Now the neighbour for a 2-sets of nodes $\{v, u\}$ in 2-LWL is defined by the set of set of 2-sets $\{\{w, u\}, \{v, w\}\}$ where w now ranges over the union of neighbours of u and v, instead of over all nodes. Algorithm 3 outlines the 2-LWL algorithm and Figure 2 illustrates the different neighbour definitions of 2-WL and 2-LWL.

Algorithm 2: 2-WL algorithm

```
Input: A graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{F}, \mathbf{L} \rangle$, injective HASH function,
       and number of iterations $L$.
Output: Multiset of colours.
1 $e(v, u) \leftarrow \mathbf{L}(v, u), \forall (v, u) \in \mathbf{E}$
2 $e(v, u) \leftarrow \perp, \forall (v, u) \in (\mathbf{V}^2) \setminus \mathbf{E}$
3 $c^0(v, u) \leftarrow (\mathbf{F}(v), \mathbf{F}(u), e(v, u)), \forall (v, u) \in \mathbf{V}^2$
4 for $j = 1, \ldots, L$ do for $(v, u) \in \mathbf{V}^2$ do
5     $c^j(v, u) \leftarrow \textsc{hash}\big(c^{j-1}(v, u), \{\{(c^{j-1}(w, u), c^{j-1}(v, w)) \mid w \in \mathbf{V}\}\}\big)$
6 return $\bigcup_{j=0,\ldots,L} \{\{c^j(v, u) \mid (v, u) \in \mathbf{V}^2\}\}$
```

[2] In contrast, the *Oblivious* k-WL (cf. [25, Section 5, page 5]) defines the neighbours as a *sequence of multiset of pairs* for $k = 2$.

Algorithm 3: 2-LWL algorithm

```
Input: An undirected graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{F}, \mathbf{L} \rangle$, injective
       HASH function, and number of iterations $L$. Let $\{u, v\}$
       denote a node pair without order or undirected edge.
Output: Multiset of colours.
1 $e\{v,u\} \leftarrow \mathbf{L}\{v,u\}, \forall \{v,u\} \in \mathbf{E}$; $e\{v,u\} \leftarrow \perp, \forall \{v,u\} \in \binom{\mathbf{V}}{2} \setminus \mathbf{E}$
2 $c^0\{v,u\} \leftarrow (\mathbf{F}(v), \mathbf{F}(u), e\{v,u\}), \forall \{v,u\} \in \binom{\mathbf{V}}{2}$
3 for $j = 1, \dots, L$ do for $\{v,u\} \in \binom{\mathbf{V}}{2}$ do
4   $c^j\{v,u\} \leftarrow \text{HASH}\big(c^{j-1}\{v,u\}, \{\!\{\{\!\{c^{j-1}\{w,u\}, c^{j-1}\{v,w\}\}\!\} \mid$
      $w \in \mathbf{N}(v) \cup \mathbf{N}(u)\}\!\}\big)$
5 return $_{j=0,\dots,L}\{\!\{c^j\{v,u\} \mid \{v,u\} \in \binom{\mathbf{V}}{2}\}\!\}$
```

Algorithm 4: iWL algorithm

```
Input: A graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{F}, \mathbf{L} \rangle$, injective HASH function,
       and number of iterations $L$.
Output: Multiset of colours.
1 for $w \in \mathbf{V}$ do
2   $c^0_w(v) \leftarrow \mathbf{F}(v), \forall v \in \mathbf{V} \setminus \{w\}$ ; $c^0_w(w) \leftarrow (\mathbf{F}(w), \otimes)$
3   for $j = 1, \dots, L$ do for $v \in \mathbf{V}$ do
4     $c^j_w(v) \leftarrow$
        $\text{HASH}\left(c^{j-1}_w(v), \bigcup_{\iota \in \Sigma_\mathrm{E}} \{\!\{(c^{j-1}_w(u), \iota) \mid u \in \mathbf{N}_\iota(v)\}\!\}\right)$
5 return $\bigcup_{w \in \mathbf{V}} \bigcup_{j=0,\dots,L} \{\!\{c^j_w(v) \mid v \in \mathbf{V}\}\!\}$
```

iWL We introduce an expressive WL algorithm extension inspired by Identity-aware GNNs (ID-GNNs) [71], orthogonal to the k-WL algorithms. ID-GNNs run a GNN $|\mathbf{V}|$ times on a graph which uses different parameters for embedding updates on a selected, individualised node on each GNN run. We kernelise the ID-GNN algorithm into what we call the individualised WL algorithm, presented in Algorithm 4 and Figure 2.

We have a single outer loop iterating over all nodes $w \in \mathbf{V}$ in a graph in Line 1, and within each inner loop, all nodes are assigned their initial node colour, except for w which is augmented a special, individualised colour $\otimes$ that is not in Σ_V of $\mathbf{F}$. The remainder of the algorithm is equivalent to the WL algorithm, except that iWL now returns $|\mathbf{V}|$ more colours in the output multiset due to the outer loop.

Given that the number of colours returned by iWL is quadratic in the number of refinable items, we proposed normalising the embeddings of the multisets by dividing $\text{COUNT}(\mathbf{M}, c)$ by $|\mathbf{V}|$. In the experiments below, we notate this feature generator as niWL.

Iterations

The number of iterations in a WLF configuration refers to the L parameter in Algorithm 1 and similar WL algorithms. One can view the number of iterations as analogous to the number of message passing layers in a graph neural network, which determines the receptive field of the network around each graph node.

Feature Pruning

Feature pruning is a technique for reducing the number of redundant or irrelevant features in a machine learning model in the context of planning. Different feature pruning approaches trade off soundness, referring the maximal preservation of feature information, and computation arising from faster evaluation and lower memory footprint while training models. In our evaluation, we experiment with no pruning and the iterative MaxSAT plus frequency pruning (`i-mf`) [29]. More specifically, `i-mf` prunes features whose evaluations on the training set are equivalent to existing features as seen in previous works (e.g. [40, 5, 19, 16]) but uses a MaxSAT encoding to add the constraint that features are pruned only if no other features that depend on it are pruned. After pruning via MaxSAT, features that appear less than 1% of the time are also pruned.

Hash Function

The hash function hyperparameter determines if we remain using the `multiset` hash input in Line 3 of the WL algorithm in Algorithm 1 and similar variants, or replace the multiset hash with a `set` hash. In both cases however, the *output* still uses a multiset in order to return numeric feature vectors. The reasoning for using a set hash is generate a smaller number of features at the expense of expressivity to improve runtime and generalisation, and to reduce the number of unseen colours during inference. For example, consider a training set consisting of star graphs of degree at most 5. If at inference a new star graph of degree 6 is introduced, the center node becomes an unseen colour after one WL iteration, which prematurely limits its receptive field. Indeed, Drexler et al. [17] show that for a sample of states on most planning domains that reducing the hash function from multisets to sets does not compromise model expressivity.

5.2 External Hyperparameters

State Representation

The external state representation hyperparameter refers to the proposition which are used for feature generation. The canonical approach is to use all propositions in the state for WLF generation which we denote as the `complete` representation. Alternatively, as performed in the original WLF paper [11], one can generate features for a subset of facts such as those detected as relevant and nonstatic by Fast Downward's grounder [31]. We name this configuration `partial`. The reasoning for this is that certain static propositions may be redundant for feature generation, such as the 'up' and 'down' propositions in the Miconic domain, which are only relevant for action applicability. Removing such propositions in certain domains significantly speeds up the feature generation runtime and hence increases the number of node expansions to be performed during search. Conversely, there are cases where relevant static facts could be pruned and thus, this option provides a tradeoff between speed and expressivity, depending on the planning domain.

Optimiser

The external optimiser hyperparameter refers to the method used to compute parameters of a prediction model using WLFs. In this paper we focus on linear models for predicting heuristic functions for search. More specifically, given a set of WLFs $c_1, \dots, c_n$, we aim to find a set of weights $w_1, \dots, w_n \in \mathbb{R}$ that gives us the 'best' heuristic function $\sum_{i \in [\![n]\!]} w_i c_i(s)$.

The original WLF paper experimented with two optimiser formulations for computing linear models using Support Vector Machines (`SVR`) and linear kernel Gaussian Process Regression (`GPR`), as well as higher order kernel approaches. However, higher order kernels yielded poorer performance, and similarly for other non-linear predictors such as XGBoost and neural networks from additional informal experiments. Our hypothesis for these observations is that WL features are already generating sufficiently informative features such that a linear model is best for generalisation and other approaches lead to overfitting to the training label range.

In this paper, we focus on new linear models formulated by Linear Regression with L1 regularisation (`Lasso`), and additional ranking formulations: the Rank Support Vector Machine (`rkSVM`) setup for learning heuristic functions introduced by Garrett et al. [20], a similar formulation using Gaussian Process Classification (`rkGPC`) where one replaces the SVM optimiser with a linear kernel GPC optimiser, and the Linear Programming ranking formulation (`rkLP`) introduced by Chen and Thiébaux [9].

6 Experiments

In this section, we perform a one million sample size experiment to evaluate the effects of the various WLF hyperparameters covered in Section 5 and summarised in Table 1 on learning and planning. The source code, scripts for running the experiments, and appendix for the paper are publicly available in [8].

Benchmarks Our benchmarks consist of the 10 domains and the training and testing splits from the International Planning Competition Learning Track 2023 [59]. More specifically, the domains are Blocksworld (BL), Childsnack (CH), Ferry (FE), Floortile (FL), Miconic (MI), Rovers (RO), Satellite (SA), Sokoban (SO), Spanner (SP), and Transport (TR). We generate optimal plans from the given training tasks via the Scorpion planner [51] with a time limit of 30 minutes and 8GB memory limit for each training task on a cluster with Intel Xeon Platinum 8274 CPUs. We note however that the median planning time of solved tasks lies in a few seconds for each domain. Only states and their corresponding h^* values in optimal plan traces were used for non-ranking models (`Lasso`, `GPR`, `SVR`), while both states and their siblings were used for ranking models (`rkLP`, `rkGPC`, `rkSVM`). Each domain contains 90 testing tasks, for a total of 900 testing tasks. Further benchmark details are provided in Appendix A. We refer to previous work [11] for a detailed discussion and comparison of learned WL heuristics with classical planners, as the benchmarks are the same and this work focuses on the effect of hyperparameters on WLF models.

Configurations Our experiments involve training various models for learning heuristic functions for search. The configurations we experiment with are all possible combinations of the 6 hyperparameters discussed in the previous section. More specifically, the hyperparameter options and their ranges are

(1) WL Algorithm: $\{$`WL`, `iWL`, `niWL`, `2-LWL`$\}$,
(2) Iterations: $\{$`1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`$\}$,
(3) Feature Pruning: $\{$`none`, `i-mf`$\}$,
(4) Hash Function: $\{$`mset`, `set`$\}$,
(5) State Representation: $\{$`part`, `cmpl`$\}$,
(6) Optimiser: $\{$`Lasso`, `GPR`, `SVR`, `rkLP`, `rkGPC`, `rkSVM`$\}$.

Model training for all configurations was given a memory limit of 8GB and 5 minutes. The 2-WL configuration was not included in the results because their models were too memory intensive to train in the given limits. Learned heuristic functions that use the `cmpl` state representation were used with the Powerlifted search engine [12], while those which used the `part` state representation were used with Version 24.06 of the Fast Downward search engine. Models using the `part` state representation were given the FDR input of a planning task after at most 5 minutes of grounding [31]. The reason for using different search engines is that profiling showed that a significant proportion of planning time is spent on heuristic evaluation and not successor generation. Due to the large volume of experiments and resource constraints, all planning runs were given a *1 minute timeout* and 4GB memory limit. Feature pruning is not supported for the `iWL` and `niWL` algorithms. In summary, we have $2 \times 8 \times 2 \times 2 \times 2 \times 6 + 2 \times 8 \times 1 \times 2 \times 2 \times 6 = 1152$ different model configurations over 10 domains, and 900 problems. Thus, we have up to $1152 \times 10 = 11520$ learned models for up to $1152 \times 900 = 1036800$ planning runs.

Which hyperparameters provide the smallest and fastest training models? The first two rows of Figure 3 illustrate distributions of model size and training time of various models conditioned on different hyperparameter configurations. For all internal hyperparameters (top half of Table 1), the ranking of model size and train time is correlated. The most efficient configuration from each hyperparameter choice is (1) `WL`, (2) `1`, (3) `i-mf`, (4) `set`, (5) `part`, and (6) `Lasso`. These results are not too surprising as discussed in their introductions. One note however is that ranking methods generally take longer to train due to the introduction of pairwise comparisons between data compared to regression methods (`SVR`, `GPR`, `Lasso`).

Which hyperparameters provide the overall best planning performance? The third column of Figure 3 illustrates the distribution of planning coverage of various models, while Table 2 displays the scores of the best performing configuration, both conditioned on different hyperparameter configurations. The hyperparameters in each group except Optimiser with the best median performance also match those with the best maximal performance (MΣ column in Table 2). This exception for the Optimiser group is because the best (`rkSVM`) models are more expensive to train and thus its median performance is degraded by configurations using other expensive hyperparameters. Regardless, we decide that choosing hyperparameters based on the best overall, maximal performance is the best starting point as median performance is influenced by suboptimal hyperparameter choices. To summarise from the data, the best overall choice of hyperparameters for WLFs are (1) `WL`, (2) `1`, (3) `i-mf`, (4) `set`, (5) `part`, (6) `rkSVM`. We note that there are exceptions on a per domain basis. This is the case for Feature Pruning and Iterations (`none` and `2` are better) for Blocksworld, State Representation (`cmpl` is better) for Childsnack, and Optimiser (`SVR` is better) for Satellite. Appendices B and C provide additional details and metrics concerning top performing configurations and how they perform compared to the original WLF configuration in [11].

Is there any correlation between training and planning metrics? Our final question aims to answer whether training metrics give us a sense of planning performance. In order to answer this question, we analyse the Pearson's correlation coefficient ρ between planning coverage and optimiser evaluation function (Eval), training time (Time), and model size (Size) summarised in Table 3, and more fine grained results per domain provided in Appendix D. Unfortunately, we do not find any statistically significant ($p < 0.05$), strong correlation ($|\rho| \geq 0.5$) between any training metric and planning performance for any optimiser. This is not surprising for classical and statistical machine learning methods as in this study, which are bound to the bias-variance tradeoff at least when analysing training evaluation metrics. Related and concurrent work study model selection using validation sets for policies [26].

7 Conclusion

In this paper, we have introduced several new hyperparameters associated with Weisfeiler-Leman Features (WLFs) in the context of learning to plan. We focus on the task of learning heuristics in this

	ΣM	MΣ	BL	CH	FE	FL	MI	RO	SA	SO	SP	TR
WL Algorithm‡												
WL	513	447	66	47	66	3	88	45	51	33	64	50
iWL	326	311	24	30	50	3	58	31	31	30	38	31
niWL	336	314	28	31	50	2	58	32	31	31	38	35
2-LWL	321	285	31	30	48	2	54	30	30	28	37	31
Iterations‡,*												
1	497	447	50	47	66	3	88	45	51	33	64	50
2	474	421	66	47	65	3	83	37	35	33	63	42
3	464	404	66	45	65	3	81	34	36	33	63	38
4	448	410	57	42	64	2	79	34	37	33	63	37
5	453	400	58	44	63	2	78	34	42	33	62	37
6	439	396	55	40	63	2	77	33	40	32	62	35
7	434	402	55	39	62	2	75	32	40	32	62	35
8	432	400	52	41	62	1	74	32	40	32	62	36
Feature Pruning†												
none	504	444	66	47	66	3	87	42	46	33	64	50
i-mf	492	447	50	46	66	3	88	45	51	33	63	47
Hash Function†												
mset	502	440	66	46	65	3	88	43	50	33	64	44
set	512	447	66	47	66	3	87	45	51	33	64	50
State Repr.†												
part	499	447	66	33	66	3	88	45	51	33	64	50
cmpl	444	418	62	47	63	2	64	30	46	32	61	37
Optimiser‡												
Lasso	388	351	34	30	65	1	88	31	31	33	36	39
GPR	447	427	55	22	65	2	87	41	43	33	63	36
SVR	471	431	58	30	65	2	87	42	51	32	63	41
rkLP	498	437	66	47	66	3	88	40	44	33	64	47
rkGPC	480	401	66	45	65	3	86	41	31	33	63	47
rkSVM	506	447	66	46	66	3	87	45	46	33	64	50
Orig. Conf. [11]	–	398	55	15	63	1	79	34	33	32	60	26

† The top value in the column is highlighted.
‡ The top 3 values per column are highlighted.
* The Feature Pruning option i-mf sometimes reduces the number of iterations. These occurrences are omitted from the Iterations rows.

Table 2. Best performance (↑) conditioned on hyperparameter option and planning domain. The ΣM column sums over the domain values, while the MΣ column takes the maximum total coverage conditioned on the row feature. The training timeout is *5 minutes* and memory limit is 8GB. The planning timeout is *60 seconds* and memory limit is 4GB. The final row (Orig. Conf.) corresponds to hyperparameters (WL, 4, none, mset, part, GPR) in the original WLF paper [11] using the resource limits in this study.

	Lasso	GPR	SVR	rkLP	rkGPC	rkSVM
Eval	[-.10, .00]	[-.05, .06]	[-.10, .01]	[-.32, -.21]	[.12, .25]	[.07, .19]
Time	[-.09, .01]	[-.07, .04]	[-.07, .04]	[-.13, -.00]	[-.15, -.02]	[-.15, -.02]
Size	[-.22, -.12]	[-.20, -.09]	[-.23, -.12]	[-.30, -.18]	[-.14, -.01]	[-.38, -.27]

Table 3. Confidence intervals of Pearson's correlation coefficient between training metrics (rows) and planning coverage of optimisers (columns). Statistically insignificant cells ($p \geq 0.05$) are indicated in gray.

paper, although WLFs are agnostic to the downstream planning task. We classified the WLF hyperparameters and performed an experimental evaluation with a 1,000,000 sample size in order to understand the effects of various hyperparameters. Our evaluation aimed to answer three core questions related to the effect of hyperparameters on (1) training, (2) planning, and (3) the correlation between training and planning metrics in the context of learning to plan. Indeed, from our analysis we have identified a best set of hyperparameters with almost consistent performance across different planning domains.

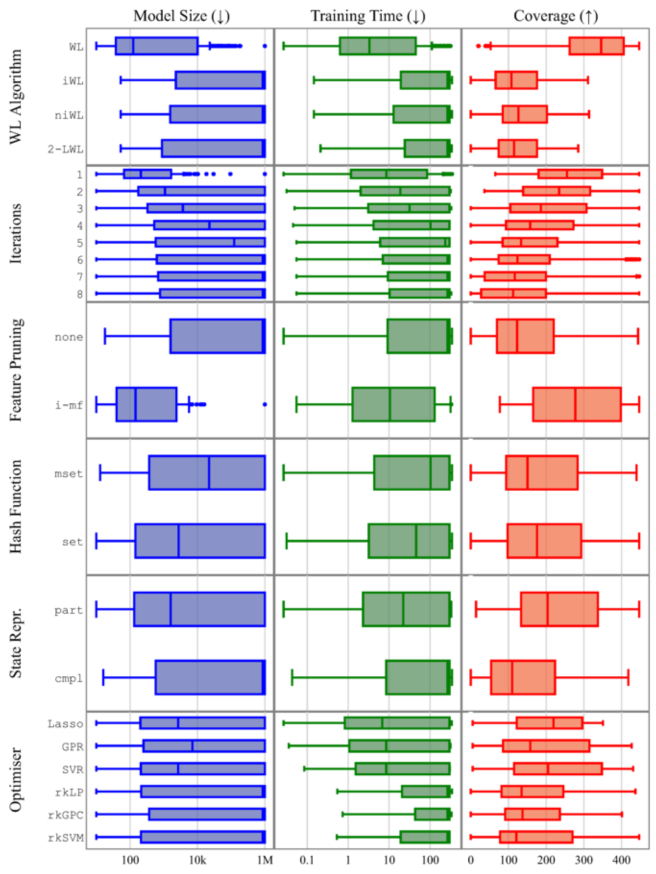

Figure 3. Boxplots of WLF counts (left, log-scale), training time in seconds (middle, log-scale), and coverage (right) of WLF models conditioned on hyperparameter options. Arrows indicate whether lower (↓) or higher (↑) values are better. If no model is learned under the resource constraints for a hyperparameter set, its model size and training time value is set to 1e6 and 300, respectively.

Acknowledgements The author thanks the anonymous reviewers for their comments that helped improve this paper.

References

[1] N. Alvarez-Gonzalez, A. Kaltenbrunner, and V. Gómez. Improving subgraph-gnns via edge-level ego-network encodings. *Trans. Mach. Learn. Res.*, 2024.
[2] P. Barceló, E. V. Kostylev, M. Monet, J. Pérez, J. L. Reutter, and J. P. Silva. The logical expressiveness of graph neural networks. In *ICLR*, 2020.
[3] P. Barceló, M. Galkin, C. Morris, and M. A. R. Orth. Weisfeiler and leman go relational. In *LoG*, 2022.
[4] B. Bonet and H. Geffner. Qualitative numeric planning: Reductions and complexity. *J. Artif. Intell. Res.*, 69, 2020.
[5] B. Bonet, G. Francès, and H. Geffner. Learning features and abstract actions for computing generalized plans. In *AAAI*, 2019.
[6] J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. In *FOCS*, 1989.
[7] S. J. Celorrio, T. de la Rosa, S. Fernández, F. Fernández, and D. Borrajo. A review of machine learning for automated planning. *Knowl. Eng. Rev.*, 27(4), 2012.
[8] D. Z. Chen. Weisfeiler-leman features for planning: A 1,000,000 sample size hyperparameter study (source code and extended version), 2025. URL https://doi.org/10.5281/zenodo.16452442.
[9] D. Z. Chen and S. Thiébaux. Graph learning for numeric planning. In *NeurIPS*, 2024.

[10] D. Z. Chen, S. Thiébaux, and F. Trevizan. Learning domain-independent heuristics for grounded and lifted planning. In *AAAI*, 2024.

[11] D. Z. Chen, F. Trevizan, and S. Thiébaux. Return to tradition: Learning reliable heuristics with classical machine learning. In *ICAPS*, 2024.

[12] A. B. Corrêa, F. Pommerening, M. Helmert, and G. Francès. Lifted successor generation using query optimization techniques. In *ICAPS*, 2020.

[13] A. B. Corrêa, A. G. Pereira, and J. Seipp. Classical planning with llm-generated heuristics: Challenging the state of the art with python code. *CoRR*, abs/2501.18784, 2025.

[14] Z. Cui, W. Kuang, and Y. Liu. Automatic verification for soundness of bounded QNP abstractions for generalized planning. In *IJCAI*, 2023.

[15] H. Dong, J. Mao, T. Lin, C. Wang, L. Li, and D. Zhou. Neural logic machines. In *ICLR*, 2019.

[16] D. Drexler, J. Seipp, and H. Geffner. Learning sketches for decomposing planning problems into subproblems of bounded width. In *ICAPS*, 2022.

[17] D. Drexler, S. Ståhlberg, B. Bonet, and H. Geffner. Symmetries and expressive requirements for learning general policies. In *KR*, 2024.

[18] G. Francès, A. B. Corrêa, C. Geissmann, and F. Pommerening. Generalized potential heuristics for classical planning. In *IJCAI*, 2019.

[19] G. Francès, B. Bonet, and H. Geffner. Learning general planning policies from small examples without supervision. In *AAAI*, 2021.

[20] C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez. Learning to rank for synthesizing planning heuristics. In *IJCAI*, 2016.

[21] H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[22] C. Gehring, M. Asai, R. Chitnis, T. Silver, L. P. Kaelbling, S. Sohrabi, and M. Katz. Reinforcement learning for classical planning: Viewing heuristics as dense reward generators. In *ICAPS*, 2022.

[23] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, and D. Weld. Pddl – the planning domain definition language. Technical report, 1998.

[24] C. Gretton and S. Thiébaux. Exploiting first-order regression in inductive policy selection. In *UAI*, 2004.

[25] M. Grohe. The logic of graph neural networks. In *LICS*, 2021.

[26] T. P. Gros, N. J. Müller, D. Fišer, I. Valera, V. Wolf, and J. Hoffmann. Per-domain generalizing policies: On validation instances and scaling behavior. In *ICAPS*, 2025.

[27] E. Groshev, M. Goldstein, A. Tamar, S. Srivastava, and P. Abbeel. Learning generalized reactive policies using deep neural networks. In *ICAPS*, 2018.

[28] M. Hao, F. Trevizan, S. Thiébaux, P. Ferber, and J. Hoffmann. Guiding GBFS through learned pairwise rankings. In *IJCAI*, 2024.

[29] M. Hao, D. Z. Chen, F. Trevizan, and S. Thiebaux. Effective data generation and feature selection in learning for planning. In *ECAI*, 2025.

[30] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool Publishers, 2019.

[31] M. Helmert. Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, 173(5-6), 2009.

[32] Y. Hu and G. D. Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI*, 2011.

[33] L. Illanes and S. A. McIlraith. Generalized planning via abstraction: Arbitrary numbers of objects. In *AAAI*, 2019.

[34] R. Karia and S. Srivastava. Learning generalized relational heuristic networks for model-agnostic planning. In *AAAI*, 2021.

[35] M. Katz, H. Kokel, K. Srinivas, and S. Sohrabi. Thought of search: Planning with language models through the lens of efficiency. In *NeurIPS*, 2024.

[36] R. Khardon. Learning action strategies for planning domains. *Artif. Intell.*, 113(1-2), 1999.

[37] N. M. Kriege, F. D. Johansson, and C. Morris. A survey on graph kernels. *Appl. Netw. Sci.*, 5(1), 2020.

[38] C. Lei, N. Lipovetzky, and K. A. Ehinger. Novelty and lifted helpful actions in generalized planning. In *SOCS*, 2023.

[39] H. J. Levesque. Planning with loops. In *IJCAI*, 2005.

[40] M. Martín and H. Geffner. Learning generalized policies in planning using concept languages. In *KR*, 2000.

[41] M. Martín and H. Geffner. Learning generalized policies from planning examples using concept languages. *Appl. Intell.*, 20(1), 2004.

[42] A. Micheli and A. Valentini. Synthesis of search heuristics for temporal planning via reinforcement learning. In *AAAI*, 2021.

[43] C. Morris, K. Kersting, and P. Mutzel. Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs. In *ICDM*, 2017.

[44] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

[45] C. Morris, G. Rattan, and P. Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. In *NeurIPS*, 2020.

[46] C. Morris, G. Rattan, S. Kiefer, and S. Ravanbakhsh. Speqnets: Sparsity-aware permutation-equivariant graph networks. In *ICML*, 2022.

[47] N. Rossetti, M. Tummolo, A. E. Gerevini, L. Putelli, I. Serina, M. Chiari, and M. Olivato. Learning general policies for planning through GPT models. In *ICAPS*, 2024.

[48] J. Segovia-Aguas, S. Jiménez, and A. Jonsson. Generalized planning as heuristic search. In *ICAPS*, 2021.

[49] J. Segovia-Aguas, S. J. Celorrio, L. Sebastiá, and A. Jonsson. Scaling-up generalized planning as heuristic search with landmarks. In *SOCS*, 2022.

[50] J. Segovia-Aguas, S. J. Celorrio, and A. Jonsson. Generalized planning as heuristic search: A new planning search-space that leverages pointers over objects. *Artif. Intell.*, 330, 2024.

[51] J. Seipp, T. Keller, and M. Helmert. Saturated cost partitioning for optimal classical planning. *J. Artif. Intell. Res.*, 67, 2020.

[52] W. Shen, F. Trevizan, and S. Thiébaux. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *ICAPS*, 2020.

[53] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12, 2011.

[54] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. Kaelbling, and M. Katz. Generalized planning in pddl domains with pretrained large language models. In *AAAI*, 2024.

[55] S. Srivastava, N. Immerman, and S. Zilberstein. Learning generalized plans using abstract counting. In *AAAI*, 2008.

[56] S. Srivastava, N. Immerman, and S. Zilberstein. A new representation and associated algorithms for generalized planning. *Artif. Intell.*, 175 (2), 2011.

[57] S. Ståhlberg, B. Bonet, and H. Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *ICAPS*, 2022.

[58] S. Ståhlberg, B. Bonet, and H. Geffner. Learning general policies with policy gradient methods. In *KR*, 2023.

[59] A. Taitler, R. Alford, J. Espasa, G. Behnke, D. Fiser, M. Gimelfarb, F. Pommerening, S. Sanner, E. Scala, D. Schreiber, J. Segovia-Aguas, and J. Seipp. The 2023 international planning competition. *AI Mag.*, 45 (2), 2024.

[60] S. Toyer, F. W. Trevizan, S. Thiébaux, and L. Xie. Action schema networks: Generalised policies with deep learning. In *AAAI*, 2018.

[61] S. Toyer, S. Thiébaux, F. Trevizan, and L. Xie. Asnets: Deep learning for generalised planning. *J. Artif. Intell. Res.*, 68, 2020.

[62] A. Tuisov, Y. Vernik, and A. Shleyfman. Llm-generated heuristics for AI planning: Do we even need domain-independence anymore? *CoRR*, abs/2501.18784, 2025.

[63] K. Valmeekam, M. Marquez, S. Sreedharan, and S. Kambhampati. On the planning abilities of large language models - A critical investigation. In *NeurIPS*, 2023.

[64] K. Valmeekam, K. Stechly, and S. Kambhampati. Llms still can't plan; can lrms? A preliminary evaluation of openai's o1 on planbench. *CoRR*, abs/2409.13373, 2024.

[65] Q. Wang, D. Z. Chen, A. Wijesinghe, S. Li, and M. Farhan. $\mathcal{N}$-wl: A new hierarchy of expressivity for graph neural networks. In *ICLR*, 2023.

[66] R. Wang and F. Trevizan. Leveraging action relational structures for integrated learning and planning. In *ICAPS*, 2025.

[67] B. Weisfeiler and A. Leman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9), 1968.

[68] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[69] R. Yang, T. Silver, A. Curtis, T. Lozano-Pérez, and L. P. Kaelbling. PG3: policy-guided planning for generalized policy generation. In *IJCAI*, 2022.

[70] S. W. Yoon, A. Fern, and R. Givan. Learning control knowledge for forward search planning. *J. Mach. Learn. Res.*, 9, 2008.

[71] J. You, J. M. G. Selman, R. Ying, and J. Leskovec. Identity-aware graph neural networks. In *AAAI*, 2021.

[72] M. J. Zhang. *Learning for Planning Under Uncertainty: Predicting Features of SSPs with Novel Graph Representation*. Bachelor's thesis, The Australian National University, 2024.

[73] L. Zhao, N. Shah, and L. Akoglu. A practical, progressively-expressive GNN. In *NeurIPS*, 2022.