# Graph Neural Networks and Graph Kernels For Learning Heuristics: Is there a difference?

**Dillon Z. Chen**[1,2] **Felipe Trevizan**[2] **Sylvie Thiébaux**[1,2]
[1]LAAS-CNRS, [2]Australian National University
{dillon.chen, sylvie.thiebaux}@laas.fr
felipe.trevizan@anu.edu.au

## Abstract

Graph neural networks (GNNs) have been used in various works for learning heuristics to guide search for planning. However, they are hindered by their slow evaluation speed and their limited expressiveness. It is also a known fact that the expressiveness of common GNNs is bounded by the Weisfeiler-Lehman (WL) algorithm for testing graph isomorphism, and for generating features for graphs. Thus, one may ask how do GNNs compare against machine learning models operating on WL features of planning problems represented as graphs? Our experiments show that linear models with WL features outpeform GNN models for learning heuristics for planning in the learning track of the 2023 International Planning Competition (IPC). Most notably, our model WL-GOOSE is the first model in the learning for planning literature which can reliably learn heuristics from scratch that are competitive with $h^{\text{FF}}$ on problem sizes much larger than those seen in the training set.

## 1 Introduction

There has been an increasing interest in the field of learning for planning due to the high computational complexity of classical planning, the rising popularity of deep learning, and the availability of compute resources. Learning for planning focuses on learning domain knowledge in an automated, domain-independent fashion for specified planning domains in order to speed up the solving process and/or improve the quality of plans returned. Learning for planning methods generally include learning heuristics [11, 27, 10, 17, 9], policies [35, 31] or sketches [7] for guiding search, and performing problem transformations such as learning to partially ground [13] and identifying useful objects [29]. For older works using more classical machine learning methods we refer to [3]. There have also been attempts with large language models but with mostly negative results [36, 30].

Most recent works (but not all) leverage neural network architectures such as graph neural networks (GNNs) for learning domain knowledge. However, we argue that in their current form, they are not best suited for providing domain knowledge due to various factors:

- GNNs have slow evaluation time which become a large bottleneck when called many times such as when using them as learned heuristic functions;
- GNNs are limited in their expressive power for learning to solve planning problems, as their simplest form is bounded by the expressiveness of the WL algorithm [37], a test for distinguishing non-isomorphic graphs; more expressive GNNs generally have marginally more expressiveness at a prohibitive cost,
- being a neural network architecture, GNNs have a large number of hyperparameters which are difficult to tune especially for planning where both acquiring data for training and validating models is expensive, and

- similarly, there is no silver bullet for the parameters used for training a GNN, including determining sufficient time for training. In contrast, classical learning models usually have a well defined optimisation procedure which does not require choosing parameters for determining termination criteria.

In our study we describe a classical machine learning architecture using graph kernels with the same theoretical expressivity as GNNs as motivated by [37], but is much faster and simpler to train and evaluate in Sec. 3. We evaluate our method against two previous graph neural network architectures, Muninn [33] and GOOSE [6], on the learning track benchmarks of the 2023 International Planning Competition [26] in Sec. 4. Our model, WL-GOOSE, trains in under a minute for any domain and achieves overall better performance than Muninn, GOOSE and also the $h^{\text{FF}}$ heuristic on the IPC2023 benchmarks. We conclude this paper in Sec. 5 by discussing some current issues and open questions in the general field of learning for planning beyond just neural network architectures.

## 2 Background and Notation

### 2.1 Planning

A classical planning task [12] is a state transition model $\Pi = \langle S, A, s_0, G\rangle$ where $S$ is a set of states, $A$ is a set of actions, $s_0 \in S$ is an initial state and $G \subseteq S$ is a set of goal states. Each action $a \in A$ is a function $a : S \to S \cup \bot$ where $a(s) = \bot$ means that action $a$ is not applicable at state $s$, and otherwise, $a(s)$ is the successor where $a$ is applied to $s$. An action also has an associated cost $c(a) \in \mathbb{N}$. A solution or *plan* in this model is a sequence of actions $\pi = a_1 \cdot \ldots \cdot a_n$ such that $s_i = a_i(s_{i-1}) \neq \bot$ for $i = 1, \ldots, n$ and $s_n \in G$. In other words, a plan is a sequence of applicable actions which progresses the initial state to a goal state when executed. The cost of a plan $\pi$ is $c(\pi) = \sum_{i=1}^{n} c(a_i)$. A planning task is *solvable* if there exists at least one plan. A plan is *optimal* if there does not exist any other plan with strictly lower cost.

We will represent planning tasks in a compact form which does not require enumerating all states and actions. A *lifted planning task* [18] is a tuple $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{A}, s_0, G\rangle$ where $\mathcal{P}$ is a set of first-order predicates, $\mathcal{O}$ is a set of objects, $\mathcal{A}$ is a set of action schema, $s_0$ is the initial state, and $G$ is the goal condition. A predicate $P \in \mathcal{P}$ has a set of parameters $x_1, \ldots, x_{n_P}$ where $n_P \in \mathbb{N}$ depends on $P$, and it is possible for a predicate to have no parameters. A predicate can be instantiated by assigning all of the $x_i$ with objects from $\mathcal{O}$ or other defined variables and in this case is known as a ground proposition. An action schema $a \in \mathcal{A}$ is a tuple $\langle \Delta(a), \mathrm{pre}(a), \mathrm{add}(a), \mathrm{del}(a)\rangle$ where $\Delta(a)$ is a set of parameter variables, and the preconditions $\mathrm{pre}(a)$, add effects $\mathrm{add}(a)$, and delete effects $\mathrm{del}(a)$ are sets of predicates from $\mathcal{P}$ instantiated with elements from $\Delta(a) \cup \mathcal{O}$. An action schema where each variable is instantiated with an object is an action.

States, including the initial state, in a lifted planning task are represented as sets of ground propositions. Note that all of the following are sets of ground propositions: states, the goal condition, and the preconditions, add effects, and delete effects of all actions. An action $a$ is applicable in a state $s$ if $\mathrm{pre}(a) \subseteq s$, in which case we define $a(s) = (s \setminus \mathrm{del}(a)) \cup \mathrm{add}(a)$ and otherwise we have $a(s) = \bot$. The cost of an action is given by the cost of its corresponding action schema. A state $s$ is a goal state if $G \subseteq s$.

### 2.2 WL algorithm for generating graph features

We write $\langle V, E, c, l\rangle$ for a graph with coloured nodes and edges, where $V$ is a set of nodes, $E \subseteq \binom{V}{2}$ is a set of undirected edges, $c : V \to \Sigma_V$ maps nodes to a set of colours $\Sigma_V$, and $l : E \to \Sigma_E$ maps edges to a set of colours $\Sigma_E$. The edge neighbourhood of a node $u$ under edge colour $\iota$ is $\mathcal{N}_\iota(u) = \{e = \langle u, v\rangle \in E \mid l(e) = \iota\}$[1]. The neighbourhood of a node $u$ in a graph is $\mathcal{N}(u) = \bigcup_{\iota \in \Sigma_E} \mathcal{N}_\iota(u)$.

The $k$-Weisfeiler-Lehman ($k$-WL) algorithms [19] are a class of algorithms which provide tests for whether pairs of graphs are isomorphic or not. It operates by iteratively updating colours assigned to all size $k$ subsets of nodes of the graphs based on neighbours of such subsets. However, they cannot distinguish all pairs of non-isomorphic graphs, meaning that there exist pairs of non-isomorphic graphs appear isomorphic to the $k$-WL algorithm for any $k \in \mathbb{N} \setminus \{0\}$. The $k+1$-WL algorithm

[1]We assume undirected edges so $\langle u, v\rangle = \langle v, u\rangle$.

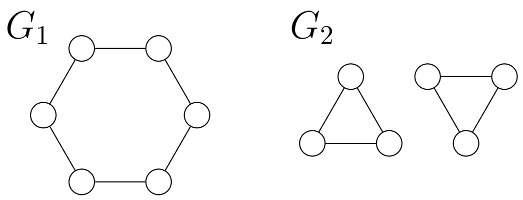

Figure 1: Two non-isomorphic graphs $G_1$ (6-cycle) and $G_2$ (two disjoint 3-cycles) which the WL algorithm returns the same outputs.

subsumes the $k$-WL algorithm as it can distinguish a greater class of non-isomorphic graphs, and furthermore is in correspondence with $k$-variable counting logics [2]. However, the complexity of the $k$-WL algorithm is exponential in $k$. We will focus on the 1-WL algorithm which is also known as colour refinement or just the WL algorithm. The original WL algorithm takes as input graphs without edge colours, i.e. $\forall e \in E, l(e) = 0$, and outputs a canonical form in terms of a multiset of colours, a set which is allowed to have duplicate elements. It has also been used to construct a kernel for graphs [28] which converts the multiset of colours in the WL algorithm into a feature vector and then uses the simple dot product kernel. We denote a multiset of elements by $\{\!\{\dots\}\!\}$.

```
Algorithm 1: WL algorithm
Data: A graph $G = \langle V, E, c \rangle$, hash function $f$, and number of WL iterations $h$.
Result: Multiset of colours.
1 $c^{(0)}(v) \leftarrow c(v), \quad \forall v \in V$
2 for $j = 1, \dots, h$ do $c^{(j)}(v) \leftarrow f\left(c^{(j-1)}(v), \{\!\{c^{(j-1)}(u) \mid u \in \mathcal{N}(v)\}\!\}\right), \quad \forall v \in V$ ;
3 return $\bigcup_{j=0,\dots,h} \{\!\{c^{(j)}(v) \mid v \in V\}\!\}$
```

The method of constructing features for a given graph is given in Alg. 1 which takes as input a graph $G$ with coloured nodes only, an injective hash function $f(\cdot, \cdot)$ which takes as input a tuple of a colour and multiset of colours and outputs a single colour, and a predefined number of WL iterations $h$. The algorithm begins in Line 1 by initialising the current colours of each node with the initial node colours. If no node colours are given in the graph, we can set them to 0. Line 2 iteratively updates each node's colour by collecting their neighbouring colours and hashing them with the injective $f$ function. In practice, $f$ is built lazily by using a map dataset and multisets are represented as sorted strings. The algorithm returns a multiset of the node colours seen over all iterations in Line 3.

If the WL algorithm outputs two different multisets for two graphs $G_1$ and $G_2$, then the graphs are non-isomorphic. However, if the algorithm outputs the same multisets for two graphs we cannot say for sure whether they are isomorphic or not. One such example is illustrated in Fig. 1 where the two graphs are not isomorphic but the WL algorithm returns the same output for both graphs because it views all nodes as the same since they have degree 2.

## 3 Method

In this section we describe how to generate features for planning states in order to learn heuristics. The process involves two main steps: (1) converting planning states into graphs with coloured nodes and edges, and then (2) running a variant of the WL algorithm on the states in order to generate features. The features are then used in classical learning models for predicting heuristics in Sec. 4.

### 3.1 Graph representation of planning tasks

We introduce the Instance Learning Graph (ILG) in Defn 3.1 for representing planning tasks. This is a modified subgraph of the Lifted Learning Graph (LLG) [6] which was used for learning domain-independent heuristics with GNNs. Fig. 2 provides an illustrative example of the ILG. Note that it only considers the propositions given in the state and goal condition, and ignores action schema and actions. Denote $[n] = \{1, \dots, n\}$.

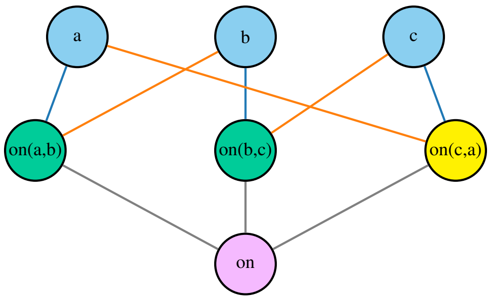

Figure 2: ILG subgraph of facts and goal condition corresponding to the on predicate of a Blocksworld instance. The current state contains a stacked on b, which is stacked on c, and the goal condition is for c to be stacked on a.

**Definition 3.1.** Let $T \in \mathbb{N}$. The *instance learning graph (ILG)* of a lifted planning problem $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{A}, s_0, G \rangle$ is the graph $G = \langle V, E, c, l \rangle$ with

- $V = \mathcal{P} \cup \mathcal{O} \cup s_0 \cup G$
- $E = \bigcup_{p=P(o_1,\ldots,o_{n_P}) \in s_0 \cup G} \left( \{ \langle p, o_i \rangle \mid i \in [n_P] \} \cup \{ \langle p, P \rangle \} \right)$
- $c : V \to P \cup \{\texttt{object}, \texttt{achieved\_prop}, \texttt{unachieved\_goal}, \texttt{achieved\_goal}\}$ defined by

$$u \mapsto \begin{cases} u, & \text{if } u \in \mathcal{P} \\ \texttt{object}, & \text{if } u \in \mathcal{O} \\ \texttt{achieved\_goal}, & \text{if } u \in s_0 \cap G \\ \texttt{achieved\_prop}, & \text{if } u \in s_0 \setminus G \\ \texttt{unachieved\_goal}, & \text{if } u \in G \setminus s_0 \end{cases}$$

- $l : E \to \mathbb{N} \cup \{0\}$ with $\langle p, P \rangle \mapsto 0$ and $\langle p, o_i \rangle \mapsto i$.

The graph consists of a node for each object, predicate and the union of propositions that are true in the state $s_0$ and the goal condition $G$. A proposition is connected to its corresponding arity $n$ predicate with an edge of label 0 and also to the $n$ object nodes which instantiates the proposition. The labels of the $n$ edges correspond to the position of the object in the predicate argument. The colours of the nodes indicate whether the node corresponds to an object or predicate of the problem, or determines whether it is a proposition belonging to $s_0$ or $G$ only or both. Note that each predicate has a unique colour while all objects have the same colour.

## 3.2 WL for edge coloured graphs and constructing features

There is no canonical WL algorithm for edge coloured graphs. We modify the WL algorithm to make use of edge coloured graphs by replacing Line 2 in Alg. 1 with the update function

$$c^{(j)}(v) \leftarrow f\left( c^{(j-1)}(v), \bigcup_{\iota \in \Sigma_E} \left\{\!\!\left\{ (\iota, c^{(j-1)}(u)) \mid u \in \mathcal{N}_\iota(v) \right\}\!\!\right\} \right). \tag{1}$$

Note that edge colours do not update during this modified WL algorithm. It is possible to run a variant of the WL algorithm which modifies edge colours but this comes at an additional computational cost given that usually $|E| \gg |V|$.

We follow the procedure of generating WL features [28] by representing multisets of colours as a histogram vector. The feature vector of a graph is a vector $v$ with size the number of seen colours, where $v[c]$ counts how many times the WL algorithm has seen colour $c$ over all its iterations. Formally, let $\langle V_1, E_1, c_1, l_1 \rangle, \ldots, \langle V_n, E_n, c_n, l_n \rangle$ be the set of training graphs. Then the colours the WL algorithm sees in the training graphs are given by

$$C = \{ c_i^{(j)}(v) \mid i = 1, \ldots, n; j = 0, \ldots, h; v \in V_i \} \tag{2}$$

where $c_i^{(j)}(v)$ is the colour of node $v$ in graph $i$ during the $j$-th iteration of WL for $j > 0$ and $c_i^{(0)}(v) = c_i(v)$. Let $I$ be an enumeration of $C$ which maps a colour into an integer from $1, \ldots, |C|$. Then given any graph (not necessarily a training graph), we construct a feature $v$ of size $|C|$ from a multiset $M$ returned by WL: for each element $\kappa$ in $M$, if $\kappa \in C$, then $v_{I(\kappa)}$ is set to be the count of $\kappa$ in $M$, while elements $\kappa$ not in $M$ are ignored and do not contribute to $v$.

## 4 Experiments

In this section, we conduct experiments to study the effectiveness of our method for learning heuristic functions in comparison to graph neural networks architectures. We name our architecture which combines the aforementioned methods for learning domain-dependent heuristic functions for classical planning problems WL-GOOSE [2]. WL-GOOSE views states $s$ in search of a given planning instance $\langle S, A, s_0, G\rangle$ as new individual planning tasks $\langle S, A, s, G\rangle$. The planning tasks are then converted into ILGs as input into the modified WL algorithm (Eq. 1) for generating vector features. The vector features are then trained using support vector regression using the dot product kernel. Experiments with nonlinear kernels, linear regression and neural networks yielded worse results. The learned heuristics are used for eager GBFS search in version 22.12 of the Fast Downward planning system [14].

### 4.1 Benchmarks

We consider the domains and training and test sets from the learning track of the 2023 International Planning Competition (IPC) [26]. The domains are Blocksworld, Childsnack, Ferry, Floortile, Miconic, Rovers, Satellite, Sokoban, Spanner, and Transport. Each domain contains instances categorised into easy, medium and hard difficulties depending on the number of objects of the instance. The training set of each domain consists of at most 99 easy instances which are disjoint from the easy testing instances. The test set of each domain consists of exactly 30 instances from each of the three easy, medium and hard difficulties that are not seen in the training set. Table 1 summarises the problem sizes of each difficulty of each domain. We test against GBFS with the $h^{\text{FF}}$ heuristic [15], and two other GNN architectures which learn heuristics: Muninn [33] and GOOSE [6].

Muninn is a GNN architecture based on previous work by the authors of [31] which learns a heuristic for use both as a greedy policy and for A* search and was a participant in the competition. Their paper is motivated by the well known result that the expressivity of GNNs is bounded by the WL algorithm [37] and can distinguish counting logics with 2 variables [1]. The authors provide theoretical insights into the expressiveness limits of GNNs and methods to deal with suboptimality of learned heuristic functions [32] as opposed to experimental performance.

GOOSE [6] is our approach of learning heuristics using GNNs. We introduced the Lifted Learning Graph (LLG) and our approach is the first domain-independent graph representation of the lifted planning tasks used for learning heuristics. We also introduced results concerning which domain-independent heuristics GOOSE can or cannot learn, such as $h^{\text{add}}$, $h^{\max}$, $h^{+}$ and $h^{*}$. For the following experiments, GOOSE uses 4 message passing layers, while Muninn uses 30 message passing layers. In our experiments, we will run GOOSE with the ILG instead of the LLG given that the latter graph representation is focused on domain-independent training while the former is a smaller graph than the LLG and is focused on domain-dependent training. Experimental results show that GOOSE with ILG performs better than with LLG across the board, although at the expense of being able to generalise across domains. Furthermore, we tested both the mean and max message passing aggregators. We will refer to GOOSE as GNN-GOOSE from here on.

The other planners in the competition either learn a policy or policy sketches, or a problem transformation of a planning instance for use with a base domain-independent planner. Given that we are focused on the effectiveness of learned heuristics, we exclude these planners and other domain-independent satisficing planners from the experimental evaluation. For all planners except GNN-GOOSE, we run experiments on a cluster with single Intel Xeon 3.2 GHz CPU cores, memory limit of 8GB, and timeout of 1800 seconds. For GNN-GOOSE we use Intel Xeon Gold 5218R 2.1GHz CPU cores, an NVIDIA RTX A6000 GPU with 48GB memory, and a timeout of 1800 seconds. However, we note that GNN-GOOSE never uses more than 8GB of GPU or main memory before reaching the timeout.

[2]<u>**W**</u>eisfeiler-<u>**L**</u>ehman <u>**G**</u>raphs <u>**O**</u>ptimised f<u>**O**</u>r <u>**S**</u>earch <u>**E**</u>valuation

Table 1: Sizes of problems in the easy, medium and hard difficulties of each domain in the IPC 2023 Learning Track benchmarks [26]. Training problems have size in the easy category.

| Domain | Objects | Easy | Medium | Hard |
|---|---|---|---|---|
| Blocksworld | $b$ blocks | $b \in [5, 30]$ | $b \in [35, 150]$ | $b \in [160, 500]$ |
| Childsnack | $s$ sandwiches | $s \in [4, 15]$ | $s \in [15, 60]$ | $s \in [50, 450]$ |
| | $c$ children | $c \in [4, 10]$ | $c \in [15, 40]$ | $c \in [50, 300]$ |
| | $t$ trays | $t \in [1, 3]$ | $t \in [2, 5]$ | $t \in [4, 10]$ |
| Ferry | $c$ cars | $c \in [1, 20]$ | $c \in [10, 100]$ | $c \in [200, 1000]$ |
| | $l$ locations | $l \in [5, 15]$ | $l \in [20, 50]$ | $l \in [100, 500]$ |
| Floortile | $n$ grid size | $n \in [3, 8]$ | $n \in [10, 22]$ | $n \in [25, 37]$ |
| | $r$ robots | $r \in [1, 3]$ | $r \in [4, 15]$ | $r \in [15, 35]$ |
| Miconic | $p$ passengers | $p \in [1, 10]$ | $p \in [20, 80]$ | $p \in [50, 500]$ |
| | $f$ floors | $f \in [4, 20]$ | $f \in [30, 60]$ | $f \in [80, 200]$ |
| Rovers | $r$ rovers | $r \in [1, 4]$ | $r \in [5, 10]$ | $r \in [15, 30]$ |
| | $o$ objectives | $o \in [1, 10]$ | $o \in [15, 80]$ | $o \in [100, 200]$ |
| | $c$ cameras | $c \in [1, 4]$ | $c \in [5, 50]$ | $c \in [60, 100]$ |
| | $w$ waypoints | $w \in [4, 10]$ | $w \in [15, 90]$ | $w \in [100, 200]$ |
| Satellite | $s$ satellites | $s \in [3, 10]$ | $s \in [15, 40]$ | $s \in [50, 100]$ |
| | $i$ instruments | $i \in [3, 20]$ | $i \in [15, 80]$ | $i \in [50, 200]$ |
| | $m$ modes | $m \in [1, 3]$ | $m \in [3, 5]$ | $m \in [5, 10]$ |
| | $d$ directions | $d \in [4, 10]$ | $d \in [15, 30]$ | $d \in [40, 100]$ |
| Sokoban | $n$ grid size | $n \in [8, 13]$ | $n \in [20, 50]$ | $n \in [60, 100]$ |
| | $b$ boxes | $b \in [1, 4]$ | $b \in [5, 35]$ | $b \in [40, 80]$ |
| Spanner | $l$ locations | $l \in [4, 10]$ | $l \in [15, 45]$ | $l \in [50, 100]$ |
| | $s$ spanners | $s \in [1, 10]$ | $s \in [30, 90]$ | $s \in [100, 500]$ |
| | $n$ nuts | $n \in [1, 5]$ | $n \in [15, 50]$ | $n \in [50, 250]$ |
| Transport | $p$ packages | $p \in [1, 15]$ | $p \in [5, 45]$ | $p \in [20, 200]$ |
| | $l$ locations | $l \in [5, 15]$ | $l \in [20, 40]$ | $l \in [50, 100]$ |
| | $v$ vehicles | $v \in [3, 6]$ | $v \in [10, 20]$ | $v \in [30, 50]$ |
| | $m$ capacity | $m = 2$ | $m = 4$ | $m = 10$ |

## 4.2 Training

For all models, training was done in a domain-dependent fashion. This means that models are trained on data from a specific domain and are evaluated only from problems within that specific domain. We use the optimal planner scorpion [25] to return optimal plans on training instances with a 30 minute time out on each instance. We use the states and the corresponding cost to the goal from each plan as training data. All inference models are trained with the `scikit-learn` package [24], which makes use of `libsvm` [4] for training the support vector regression models.

We do not train Muninn but instead used the trained weights from the competition. Muninn uses as training data all reachable states in as many training problems as possible given the time limit and the corresponding optimal costs to the goal from each state. Its training process involves first trying to learn an optimal value function on the training data where possible, and if it fails switches to learn a suboptimal value function.

## 4.3 Results

We used the same metric in the competition for scoring planners. If a planner does not solve an instance, it is given a score 0 for that instance. Otherwise, it is given a score $\min(1, C^*/C)$ where $C$ is the cost of the plan returned by the planner on that instance, and $C^*$ is the cost of a reference plan, which is not necessarily optimal. A planner's total score is the sum of its scores for all tasks. Tab. 2 highlights these scores alongside the coverage of planners. Fig. 3 displays plots of number of expanded nodes and plan quality of WL-GOOSE against $h^{\text{FF}}$ and GNN-GOOSE.

Table 2: Coverage (left) and competition score (right) of planners over various domains and difficulties. The competition score of a planner for an instance is $\min(1, C^*/C)$ if it returns a valid plan of cost $C$, where $C^*$ is the cost of a reference plan, and 0 otherwise. The first/second/third/fourth row of each domain corresponds to metric values on easy/medium/hard/all difficulties. The best planner is highlighted in bold, and the top three planners in each row is indicated by the intensity of the cell shading. - indicates zero coverage in the corresponding entry. **WL** is the new main contribution of this paper. The bottom tables are condensed version of the top tables by omitting the rows based on difficulties.

| | | | | GOOSE | | |
|---|---|---|---|---|---|---|
| Domain | blind | $h^{\text{FF}}$ | Muninn | GNN(max) | GNN(mean) | **WL** |
| blocksworld | 8 | 28 | **30** | **30** | **30** | **30** |
| | - | - | 10 | 16 | **24** | 19 |
| | - | - | - | 3 | **4** | - |
| | 8 | 28 | 40 | 49 | **58** | 49 |
| childsnack | 9 | **26** | 11 | 19 | 20 | 20 |
| | - | - | - | - | - | - |
| | - | - | - | - | - | - |
| | 9 | **26** | 11 | 19 | 20 | 20 |
| ferry | 10 | **30** | **30** | **30** | **30** | **30** |
| | - | **30** | 16 | **30** | **30** | **30** |
| | - | 8 | - | 4 | 12 | **14** |
| | 10 | 68 | 46 | 64 | 72 | **74** |
| floortile | 2 | **12** | - | - | - | 2 |
| | - | - | - | - | - | - |
| | - | - | - | - | - | - |
| | 2 | **12** | - | - | - | 2 |
| miconic | **30** | **30** | **30** | **30** | **30** | **30** |
| | - | **30** | - | **30** | **30** | **30** |
| | - | **30** | - | **30** | **30** | **30** |
| | 30 | **90** | 30 | **90** | **90** | **90** |
| rovers | 15 | 29 | 15 | 25 | 28 | **30** |
| | - | 5 | - | - | 1 | **15** |
| | - | - | - | - | - | - |
| | 15 | 34 | 15 | 25 | 29 | **45** |
| satellite | 12 | **30** | 16 | 29 | 27 | **30** |
| | - | **30** | 2 | 2 | 2 | 7 |
| | - | **5** | - | - | - | - |
| | 12 | **65** | 18 | 31 | 29 | 37 |
| sokoban | 27 | **30** | 26 | **30** | **30** | **30** |
| | - | 6 | - | 2 | 3 | **7** |
| | - | - | - | - | - | - |
| | 27 | 36 | 26 | 32 | 33 | **37** |
| spanner | **30** | **30** | **30** | **30** | **30** | **30** |
| | - | - | 2 | - | **3** | - |
| | - | - | - | - | - | - |
| | 30 | 30 | 32 | 30 | **33** | 30 |
| transport | 9 | **30** | 17 | **30** | **30** | **30** |
| | - | 11 | - | 8 | 5 | **19** |
| | - | - | - | - | - | - |
| | 9 | 41 | 17 | 38 | 35 | **49** |
| sum | 152 | **275** | 205 | 253 | 255 | 262 |
| | - | 112 | 30 | 88 | 98 | **127** |
| | - | 43 | - | 37 | **46** | 44 |
| | 152 | 430 | 235 | 378 | 399 | **433** |

| | | | | GOOSE | | |
|---|---|---|---|---|---|---|
| Domain | blind | $h^{\text{FF}}$ | Muninn | GNN(max) | GNN(mean) | **WL** |
| blocksworld | 8.0 | 14.1 | **30.0** | 28.1 | 27.2 | 26.7 |
| | - | - | 10.0 | 14.8 | **21.9** | 17.8 |
| | - | - | - | 2.7 | **3.6** | - |
| | 8.0 | 14.1 | 40.0 | 45.6 | **52.7** | 44.4 |
| childsnack | 9.0 | **20.1** | 11.0 | 17.6 | 19.9 | 18.9 |
| | - | - | - | - | - | - |
| | - | - | - | - | - | - |
| | 9.0 | **20.1** | 11.0 | 17.6 | 19.9 | 18.9 |
| ferry | 10.0 | 29.6 | **30.0** | 29.9 | 29.9 | 29.6 |
| | - | **30.0** | 16.0 | **30.0** | **30.0** | **30.0** |
| | - | 8.0 | - | 4.0 | 12.0 | **14.0** |
| | 10.0 | 67.6 | 46.0 | 63.9 | 71.9 | **73.6** |
| floortile | 2.0 | **11.2** | - | - | - | 1.8 |
| | - | - | - | - | - | - |
| | - | - | - | - | - | - |
| | 2.0 | **11.2** | - | - | - | 1.8 |
| miconic | **30.0** | 28.7 | **30.0** | 29.2 | 29.1 | 29.0 |
| | - | **30.0** | - | **30.0** | **30.0** | **30.0** |
| | - | 29.9 | - | **30.0** | 30.0 | **30.0** |
| | 30.0 | 88.5 | 30.0 | **89.2** | 89.1 | 89.0 |
| rovers | 15.0 | **27.7** | 14.2 | 19.7 | 23.8 | 24.9 |
| | - | 5.0 | - | - | 1.0 | **11.4** |
| | - | - | - | - | - | - |
| | 15.0 | 32.7 | 14.2 | 19.7 | 24.8 | **36.4** |
| satellite | 12.0 | **28.8** | 16.0 | 22.6 | 19.6 | 25.9 |
| | - | **30.0** | 2.0 | 2.0 | 2.0 | 7.0 |
| | - | **5.0** | - | - | - | - |
| | 12.0 | **63.8** | 18.0 | 24.6 | 21.6 | 32.9 |
| sokoban | **27.0** | 23.2 | 24.3 | 26.3 | 27.0 | 26.1 |
| | - | 3.1 | - | 2.0 | 3.0 | **7.0** |
| | - | - | - | - | - | - |
| | 27.0 | 26.3 | 24.3 | 28.3 | 30.0 | **33.1** |
| spanner | **30.0** | **30.0** | **30.0** | **30.0** | **30.0** | 27.6 |
| | - | - | 2.0 | - | **3.0** | - |
| | - | - | - | - | - | - |
| | 30.0 | 30.0 | 32.0 | 30.0 | **33.0** | 27.6 |
| transport | 9.0 | 28.3 | 17.0 | 27.9 | 27.5 | **28.3** |
| | - | 11.0 | - | 7.8 | 4.1 | **18.0** |
| | - | - | - | - | - | - |
| | 9.0 | 39.3 | 17.0 | 35.6 | 31.6 | **46.3** |
| sum | 152.0 | **241.5** | 202.4 | 231.2 | 234.2 | 238.8 |
| | - | 109.1 | 30.0 | 86.6 | 94.9 | **121.2** |
| | - | 42.9 | - | 36.7 | **45.5** | 44.0 |
| | 152.0 | 393.5 | 232.4 | 354.5 | 374.6 | **404.0** |

| | | | | GOOSE | | |
|---|---|---|---|---|---|---|
| Domain | blind | $h^{\text{FF}}$ | Muninn | GNN(max) | GNN(mean) | **WL** |
| blocksworld | 8 | 28 | 40 | 49 | **58** | 49 |
| childsnack | 9 | **26** | 11 | 19 | 20 | 20 |
| ferry | 10 | 68 | 46 | 64 | 72 | **74** |
| floortile | 2 | **12** | - | - | - | 2 |
| miconic | 30 | **90** | 30 | **90** | **90** | **90** |
| rovers | 15 | 34 | 15 | 25 | 29 | **45** |
| satellite | 12 | **65** | 18 | 31 | 29 | 37 |
| sokoban | 27 | 36 | 26 | 32 | 33 | **37** |
| spanner | 30 | 30 | 32 | 30 | **33** | 30 |
| transport | 9 | 41 | 17 | 38 | 35 | **49** |
| sum | 152 | 430 | 235 | 378 | 399 | **433** |

| | | | | GOOSE | | |
|---|---|---|---|---|---|---|
| Domain | blind | $h^{\text{FF}}$ | Muninn | GNN(max) | GNN(mean) | **WL** |
| blocksworld | 8.0 | 14.1 | 40.0 | 45.6 | **52.7** | 44.4 |
| childsnack | 9.0 | **20.1** | 11.0 | 17.6 | 19.9 | 18.9 |
| ferry | 10.0 | 67.6 | 46.0 | 63.9 | 71.9 | **73.6** |
| floortile | 2.0 | **11.2** | - | - | - | 1.8 |
| miconic | 30.0 | 88.5 | 30.0 | **89.2** | 89.1 | 89.0 |
| rovers | 15.0 | 32.7 | 14.2 | 19.7 | 24.8 | **36.4** |
| satellite | 12.0 | **63.8** | 18.0 | 24.6 | 21.6 | 32.9 |
| sokoban | 27.0 | 26.3 | 24.3 | 28.3 | 30.0 | **33.1** |
| spanner | 30.0 | 30.0 | 32.0 | 30.0 | **33.0** | 27.6 |
| transport | 9.0 | 39.3 | 17.0 | 35.6 | 31.6 | **46.3** |
| sum | 152.0 | 393.5 | 232.4 | 354.5 | 374.6 | **404.0** |

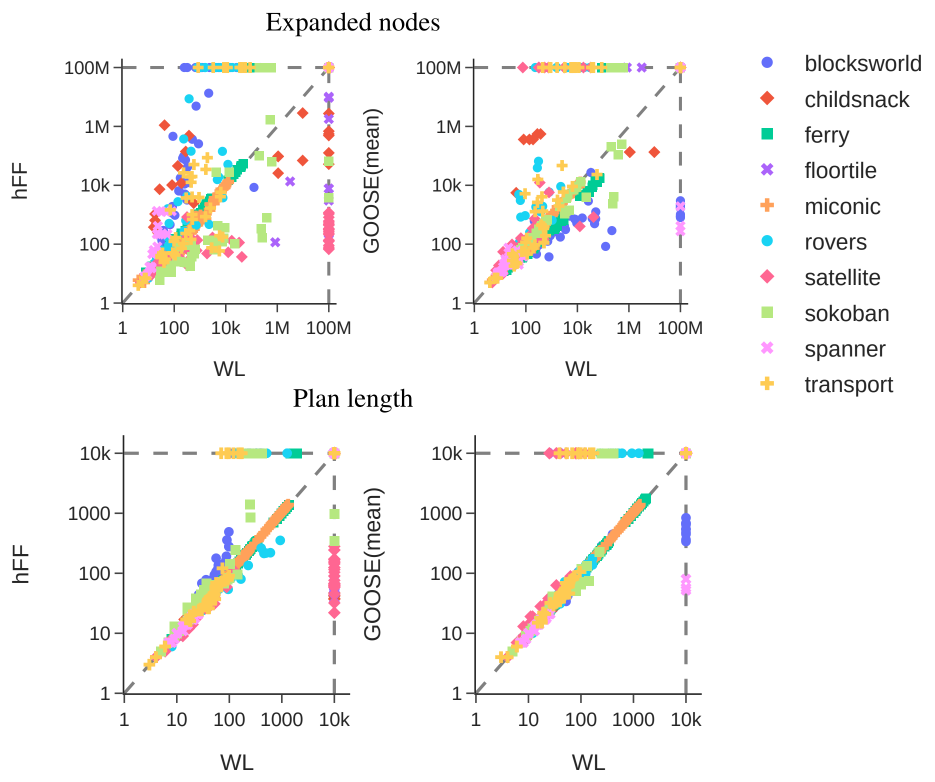

Figure 3: Number of expanded nodes (top) and length of plans (bottom) of various solvers over various domains. If a planner does not solve a problem, its corresponding data point is set to the maximum axis value. Points in the upper left triangle benefit WL-GOOSE, while points in the bottom right triangle benefit the corresponding model indicated on the $y$-axis.

### 4.3.1 Planning performance

We see from Tab. 2 that WL-GOOSE overall has the highest coverage and competition score with 433 and 404.0 respectively, outperforming both its GNN counterparts GNN-GOOSE with mean (399 and 374.6) and max (378 and 354.5) aggregators, and $h^{\text{FF}}$ (430 and 393.5). However, performance is specific to domains. GNN-GOOSE performs best on Blocksworld and marginally better on Spanner, and $h^{\text{FF}}$ performs best on Childsnack, Floortile and Satellite. WL-GOOSE performs best on the remaining domains (Ferry, Rovers, Sokoban and Transport). Miconic is solved by all planners except Muninn and blind search.

### 4.3.2 Heuristic and plan quality

Fig. 3 show that the number of node expansions and plan quality of WL-GOOSE in comparison to $h^{\text{FF}}$ and GNN-GOOSE varies depending on the domain. However, it is generally the case that the model with the better coverage on the domain expands fewer nodes and returns better quality plans. For example, GNN-GOOSE performs better than WL-GOOSE which in turns performs better than $h^{\text{FF}}$ on Blocksworld and this ranking of planners is the same as with the number of expanded nodes and plan quality. Meanwhile, WL-GOOSE has the overall best performance on Transport and also the fewest node expansions. Sokoban is an exception where WL-GOOSE expands more nodes than $h^{\text{FF}}$ but due to WL-GOOSE's fast heuristic evaluations it solves one more problem (37 vs 36).

### 4.3.3 Runtime of GNNs vs graph kernels

WL-GOOSE has much faster evaluation time than graph neural networks which balances out its poorer learned heuristic quality in order to achieve the same coverage as GNN-GOOSE. The complexity of the WL-GOOSE for producing a heuristic estimate is $O(ndh)$ where $h$ is the number of iterations of the WL-algorithm, $n$ is the number of nodes in the graph and $d$ is the degree of the graph. This is because the WL algorithm performs $h$ iterations of updating $n$ nodes' colours by collecting colours from each node's neighbours of size at most $d$. A heuristic estimate performed on a linear combination of the returned colours is upper bounded by $ndh$. This is in contrast to GNNs which has the same complexity multiplied with at least an additional cubic term arising from matrix operations on high dimensional feature vectors when run on a single core CPU. For example, GNN-GOOSE has over 100000 weight parameters which have to be multiplied with latent embeddings. In Fig.3, we see that GNN-GOOSE expands up to $10^6$ nodes before it solves a problem or times out when using GPUs, in comparison to WL-GOOSE which expands up to $10^7$ nodes. However, GNN-GOOSE is up to 10-100x times slower when run on single core CPUs compared to GPUs.

## 5 Discussion

In this section we comment about other open problems and prevalent issues in the field of learning for planning, beyond the current limitations of neural network architectures for planning. The learning track of the 2023 IPC was a welcome addition to the field of learning for planning, which aimed to provide an even playing field for works in this field and proposing a consistent benchmark of domains and training splits for evaluating such works. It also provided many insights and conversely also highlighted many open problems and issues in this field. Most notably is that classical planners are still outperforming planners which make use of learned domain knowledge as seen in the full competition results. Furthermore, in most domains, learning planners generally struggle to generalise to unseen problem sizes as seen in the low coverage on medium and hard problems on almost all domains. Being a programming competition, this could be attributed to bugs in the submitted planners but it is also reasonable to attribute the results to some currently open problems in the field. Such problems are also applicable to learning as a whole, but are made especially difficult in the computational difficult task of planning.

### 5.1 Learning planners are not robust

One difficulty of learning for planning is ensuring robustness of planners. Even in the domain-dependent training setting, unseen test problems generally differ greatly than the training problems as a result of the computational complexity of reasoning in planning. This is in contrast to classical applications of machine learning to datasets which are generally drawn from a nice distribution. Some methods such as validation by using learned domain knowledge for search [9, 31] have been attempted for model selection but such methods have no theoretical backing and are generally expensive to perform.

Besides the absence of generalisation results of learning planners, the results from the 2023 IPC learning track also highlights the poor robustness of learning planners. Other learning planners which outperform Muninn generally learn a transformation of the given planning task before providing the transformation to a classical domain-independent planner for use, with the hope that the transformation prunes the search space in order to return plans faster or return higher quality plans. However, the competition results show that learning such transformations actually *decrease* the performance of the base planner.

### 5.2 Absence of automatic training data selection methods

Another open question in learning for planning is an automatic and efficient method for select good training data for arbitrary domains. There are many issues with acquiring training data for learning for planning. Even if we can assume that we have a generator for any given domain, constructing training data from generated instances is expensive as it usually requires solving the instances. It is also quite unlikely that we can generate good training data efficiently, given that we cannot generate useful labelled training data for $\mathsf{NP}$-hard problems efficiently under the common assumption that $\mathsf{NP} \neq \mathsf{coNP}$ [38], while propositional planning in general is $\mathsf{PSPACE}$-complete [8].

Ignoring the cost of generating training data, it is also difficult to know how much training data we need and whether a training instance is useful for our model. For example, how many Ferry instances do we have to generate in order for a planner to learn a policy for this domain, and what size instances do we need? Is five training problems with between 5-10 ferries enough? Is it redundant to provide two training problems with the same number of ferries? Some experimental insights have been provided with the effect of training data size and sample strategies from given optimal plans [10] although this was done for the case of a specific neural network architecture and in the setting of learning to generalise on similar size problems. Methods such as bootstrapping can aid this problem [16, 17, 9, 22] as it allows for continuously learning from increasingly difficult problems and combining the training and solving process together. Nevertheless, such methods still require hand choosing training problems a priori. One could also explore ideas of transfer learning, zero shot or one shot learning, for example by learning from a model pretrained on a large set planning instances [5] from different domains [27].

### 5.3 Forms of learned domain knowledge do not complement planners well

Most learning for planning works aim to learn a heuristic or policy with the aim of solving planning problems quickly. The hope is that domain-dependent learners can learn the optimal heuristic or complete policies for planning problems under the assumption that they only have to do so for a specific domain. However, this is not possible for all domains for any learning models considered so far [31, 7, 5], meaning that we have to resort to using search methods to aid suboptimal learned domain knowledge [27, 32]. Nevertheless, classical planners are still outperforming learning planners on computationally difficult domains.

Another issue concerning such works is the absence of well motivated loss functions for learning models. Almost all works consider using common regression loss functions, such as mean squared error loss, when optimising learned heuristic functions, or classification loss functions, such as cross entropy, for learning policies. However, such optimisation problems are originally derived from a statistical viewpoint which may not be best suited for combinatorially hard problems such as planning. We may want to derive or study new learning for planning algorithms whose learning module's optimisation problem is better suited for its downstream task.

One recent approach in this direction is by modelling a heuristic as truncated Gaussians bounded below by admissible heuristics [21] instead of a single Gaussian distribution as implicitly done when optimising a mean squared error loss function. Levin Search [23, 22] offers a solution to this problem as its optimisation problem involves directly minimising the number of node expansions for search. Descending and dead-end avoiding heuristics can be learned with a mixed integer program encoding [11] from description logic features [20]. The same description logic features can also be used to learn policy sketches [7] which is used in conjunction with bounded width search, and for learning domain-dependent dead-end detectors [34].

Alternatively, we could strive to learn other forms of domain knowledge for planning beyond heuristics, policies and transformations of the planning task, or maybe a combination of all such methods.

## 6 Conclusion

In this work we have proposed a classical machine learning architecture for learning to solve planning problems. The architecture, WL-GOOSE, makes use of the WL algorithm for generating feature vectors of planning tasks represented as graphs, which are then used for learning heuristic functions with linear regression and kernel methods. An informal theoretical argument suggests that it has the same inference power as GNNs but are orders of magnitude faster to evaluate and train. Experimental results indeed support this as WL-GOOSE achieved overall better coverage than its GNN counterpart, GNN-GOOSE. Furthermore, WL-GOOSE is the first model which can learn heuristics from scratch that are competitive with $h^{\text{FF}}$ on greedy best first search on nontrivial problem sizes.

## Acknowledgements

The authors would like to thank the reviewers for their comments. A proportion of the computational resources for this project were provided by the Australian Government through the National Computational Infrastructure (NCI) under the ANU Startup Scheme. Sylvie Thiébaux's work was supported by Australian Research Council grant DP220103815, by the Artificial and Natural and Intelligence Toulouse Institute (ANITI), and by the European Union's Horizon Europe Research and Innovation program under the grant agreement TUPLES No. 101070149.

## References

[1] Pablo Barceló, Egor V Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[2] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[3] Sergio Jiménez Celorrio, Tomás de la Rosa, Susana Fernández, Fernando Fernández, and Daniel Borrajo. A review of machine learning for automated planning. *Knowl. Eng. Rev.*, 27(4):433–467, 2012.

[4] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[5] Dillon Z Chen. *GOOSE: Learning Heuristics and Parallelising Search for Grounded and Lifted Planning*. Bachelor's thesis, 2023.

[6] Dillon Z Chen, Sylvie Thiébaux, and Felipe Trevizan. GOOSE: Learning domain-independent heuristics. In *Workshop on Generalization in Planning (GenPlan)*, 2023.

[7] Dominik Drexler, Jendrik Seipp, and Hector Geffner. Learning sketches for decomposing planning problems into subproblems of bounded width. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 32, pages 62–70, 2022.

[8] Kutluhan Erol, Dana S. Nau, and V.S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artif. Intell.*, 76(1):75–88, 1995.

[9] Patrick Ferber, Florian Geißer, Felipe W. Trevizan, Malte Helmert, and Jörg Hoffmann. Neural network heuristic functions for classical planning: Bootstrapping and comparison to other methods. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 583–587, 2022.

[10] Patrick Ferber, Malte Helmert, and Jörg Hoffmann. Neural network heuristics for classical planning: A study of hyperparameter space. In *European Conference on Artificial Intelligence (ECAI)*, volume 325, pages 2346–2353, 2020.

[11] Guillem Francès, Augusto B. Corrêa, Cedric Geissmann, and Florian Pommerening. Generalized potential heuristics for classical planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 5554–5561, 2019.

[12] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

[13] Daniel Gnad, Álvaro Torralba, Martín Ariel Domínguez, Carlos Areces, and Facundo Bustos. Learning how to ground a plan - partial grounding in classical planning. In *AAAI Conference on Artificial Intelligence*, pages 7602–7609, 2019.

[14] Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246, 2006.

[15] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.*, 14:253–302, 2001.

[16] Shahab Jabbari Arfaee, Sandra Zilles, and Robert C. Holte. Learning heuristic functions for large state spaces. *Artif. Intell.*, 175(16):2075–2098, 2011.

[17] Rushang Karia and Siddharth Srivastava. Learning generalized relational heuristic networks for model-agnostic planning. In *AAAI Conference on Artificial Intelligence*, pages 8064–8073, 2021.

[18] Pascal Lauer, Alvaro Torralba, Daniel Fiser, Daniel Höller, Julia Wichlacz, and Joerg Hoffmann. Polynomial-time in pddl input size: Making the delete relaxation feasible for lifted planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[19] AA Leman and Boris Weisfeiler. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.

[20] Mario Martín and Hector Geffner. Learning generalized policies in planning using concept languages. In *Principles of Knowledge Representation and Reasoning (KR)*, pages 667–677, 2000.

[21] Carlos Núñez-Molina, , Masataro Asai, Juan Fernández-Olivares, and Pablo Mesejo. On using admissible bounds for learning forward search heuristics. *arXiv preprint*, 2023.

[22] Laurent Orseau, Marcus Hutter, and Levi H. S. Lelis. Levin tree search with context models. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 5622–5630, 2023.

[23] Laurent Orseau, Levi Lelis, Tor Lattimore, and Theophane Weber. Single-agent policy tree search with guarantees. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.

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

[25] Jendrik Seipp, Thomas Keller, and Malte Helmert. Saturated cost partitioning for optimal classical planning. *J. Artif. Intell. Res.*, 67:129–167, 2020.

[26] Jendrik Seipp and Javier Segovia-Aguas. International planning competition 2023 - learning track, 2023.

[27] William Shen, Felipe Trevizan, and Sylvie Thiébaux. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2020.

[28] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[29] Tom Silver, Rohan Chitnis, Aidan Curtis, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Planning with learned object importance in large problem instances using graph neural networks. In *AAAI Conference on Artificial Intelligence*, pages 11962–11971, 2021.

[30] Tom Silver, Soham Dan, Kavitha Srinivas, Josh Tenenbaum, Leslie Kaelbling, and Michael Katz. Generalized planning in PDDL domains with pretrained large language models. *arXiv preprint*, 2023.

[31] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 629–637, 2022.

[32] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general policies with policy gradient methods. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 647–657, 2023.

[33] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Muninn. *International Planning Competition*, 2023.

[34] Simon Ståhlberg, Guillem Francès, and Jendrik Seipp. Learning generalized unsolvability heuristics for classical planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4175–4181, 2021.

[35] Sam Toyer, Sylvie Thiébaux, Felipe W. Trevizan, and Lexing Xie. Asnets: Deep learning for generalised planning. *J. Artif. Intell. Res.*, 68:1–68, 2020.

[36] Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*, 2022.

[37] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

[38] Gal Yehuda, Moshe Gabel, and Assaf Schuster. It's not what machines can learn, it's what we cannot teach. In *International Conference on Machine Learning, (ICML)*, volume 119, pages 10831–10841, 2020.