Graph Learning for Planning: The Story Thus Far and Open Challenges

[1,2]Dillon Z. Chen, [2]Mingyu Hao, [1,2]Sylvie Thiébaux, [2]Felipe Trevizan
[1]LAAS-CNRS, University of Toulouse
[2]The Australian National University
firstName.lastName@anu.edu.au

Abstract

Graph learning is naturally well suited for use in planning due to its ability to exploit relational structures exhibited in planning domains and to take as input planning instances with arbitrary number of objects. In this paper, we study the usage of graph learning for planning thus far by studying the theoretical and empirical effects on learning and planning performance of (1) graph representations of planning tasks, (2) graph learning architectures, and (3) optimisation formulations for learning. Our studies accumulate in the GOOSE framework which learns domain knowledge from small planning tasks in order to scale up to much larger planning tasks. In this paper, we also highlight and propose the 5 open challenges in the general Learning for Planning field that we believe need to be addressed for advancing the state-of-the-art.

1 Introduction

Learning for Planning (L4P) has gained significant interest in recent years due to advancements of machine learning (ML) approaches across various fields, and also because planning is one of the few problems in AI that has been unsolved by deep learning and large models [VMH+23, VMSK23, VSK24]. An aim of L4P involves designing automated, domain-independent algorithms for learning domain knowledge from small training problems for scaling up planning to problems of arbitrary size [CdlRF+12, TTTX18, TTTX20, DML+19, STT20, KS21, SBG22, SBG23, MLTK23, CTT24a, CTT24b, WT24, HTT+24, CT24a]. Planning tasks can exhibit arbitrary numbers of objects and are represented with relational languages which begs for the use of learning approaches that operate on relational structures such as graphs.

In this paper, we recount the story so far for graph learning for planning. This will be done by studying the contributions in the literature and of the authors thus far regarding the three main components of graph learning for planning as summarised in Fig. 1: (1) graph representations of planning tasks, (2) graph learning architectures, and (3) optimisation formulations for learning. On the experimental side, we present the GOOSE[1] learning framework for leveraging graph learning for planning.

[1]**G**raphs **O**ptimised f**O**r **S**earch **E**valuation; code available at https://github.com/DillonZChen/goose

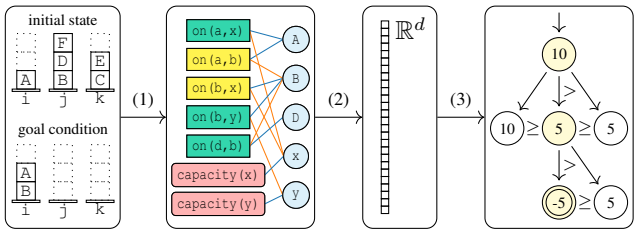


Figure 1: Typical graph learning for planning pipeline. (1) Planning tasks are represented as graphs. (2) Graph learning architectures are used to embed graphs. (3) Learning is performed via some optimisation criteria suited for making predictions in planning.

This paper summarises previous [CTT24a, CTT24b, HTT+24, CT24a] and current theoretical and experimental work by the authors. The paper is structured by introducing the general L4P problem setup in Section 2 and the formal background and definitions in Section 3. Next, we highlight contributions of the authors in graph learning for planning, summarised as follows:

(1) We provide a taxonomy and theoretical comparison of graph representations of planning tasks and their expressive power with graph learning approaches. (Section 4)

(2) We introduce a classical ML approach which outperforms deep learning methods by several orders of magnitude for planning over various metrics. (Section 5)

(3) We present optimisers better suited for learning for planning by representing heuristic functions as rankings of states rather than cost-to-go estimates. (Section 6)

Section 7 quantifies the contributions of our work by showcasing experimental results. We conclude by highlighting open problems and challenges for the general L4P field in Section 8, and avenues for future work in the field.

2 Learning for Planning

Here we briefly introduce and describe the general Learning for Planning (L4P) problem statement. The problem setup of L4P involves learning knowledge from a set of training problems in either a supervised, unsupervised, or reinforcement learning fashion that may be helpful for planning. This is in contrast to planning for each individual problem from

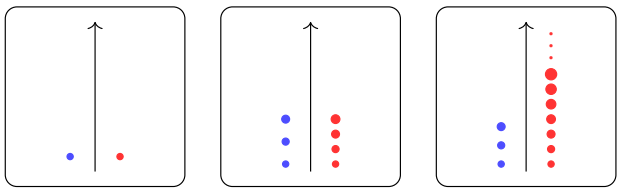


Figure 2: Visualisations of generalisation setups for planning and RL. The axis represents the number of objects in a planning task. Standalone planning and RL commonly follow the task-specific setup (left), with some RL approaches generalising across similar-sized tasks (middle). L4P approaches primarily focuses on generalisation across arbitrary numbers of objects (right).

scratch in typical planning setups [GB13], and learning to solve a specific problem instance from rewards typical in task-specific reinforcement learning [SB98] or across different initial states [FGT+22]. Furthermore, as illustrated in Fig. 2, L4P aims to generalise across an arbitrary number of objects in the planning domain, as opposed to transfer learning across similar-sized tasks or task-specific reinforcement learning. The primary objective of L4P is that by performing either few-shot training or bootstrapping, we can scale up to much larger planning tasks which domain-independent planners cannot handle due to no free lunch.

We define an L4P problem as any tuple $\langle \mathcal{D}, \mathcal{T}_{\text{train}}, \mathcal{T}_{\text{test}} \rangle$ where $\mathcal{D}$ is a domain characterising a class of planning tasks, $\mathcal{T}_{\text{train}}$ is a finite set of training tasks from $\mathcal{D}$, and $\mathcal{T}_{\text{test}}$ is a (possibly infinite) set of testing tasks from $\mathcal{D}$. An L4P solution is defined as a set of plans corresponding to problems in $\mathcal{T}_{\text{test}}$.

L4P opens a rich array of possibilities for how one may come up with algorithms for solving the problem, and methods for evaluating such algorithms. We can characterise an L4P algorithm as an algorithm with two components:

(1) a **learner module**, whose input is $\mathcal{D}$ and $\mathcal{T}_{\text{train}}$ and output is a *knowledge* artifact, and

(2) a **planner module**, whose input is a task from $\mathcal{T}_{\text{test}}$ and the *knowledge* artifact, and output is a plan.

Fig. 3 illustrates the general L4P setup and pipeline. The learner module learns from the training tasks and uses the domain as prior knowledge to produce a knowledge artifact, which can then be used by a planner module to plan for arbitrary tasks from $\mathcal{T}_{\text{test}}$. Common knowledge artifacts include heuristic or value functions, and global policies.

3 Background

In this section, we provide the technical background and formalisms of planning used in works described in the paper. However, they may be skipped upon first read as they are not entirely necessary for a majority of the paper.

3.1 Planning Task

Let $[\![n]\!]$ denote the set of integers $\{1, \ldots, n\}$. A planning task can be understood as a state transition model [GB13]

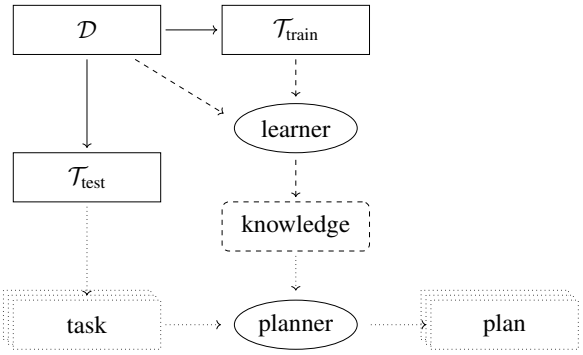


Figure 3: The general Learning for Planning (L4P) setup.

given by a tuple $\Pi = \langle S, A, s_0, G \rangle$ 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 $\pi = a_1, \ldots, a_n$ 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 Π induces a new planning task $\Pi' = \langle S, A, s, G \rangle$. A planning task is solvable if there exists at least one plan.

3.2 Planning Representation

We now describe the numeric planning formalism from PDDL2.1 [FL03]. Numeric planning encapsulates classical planning which is a compact, lifted representation of a planning task with the use of predicate logic and relational numeric variables. More specifically, a numeric planning task is a tuple $\Pi = \langle \mathcal{O}, \Sigma_p, \Sigma_f, \mathcal{A}, s_0, \mathcal{G} \rangle$, where $\mathcal{O}$ denotes a set of objects, Σ_p/Σ_f a set of predicate/function symbols, $\mathcal{A}$ a set of action schemata, s_0 the initial state, and $\mathcal{G}$ now the goal condition. Details of the representation of actions in a planning task induced by grounding action schemata from $\mathcal{A}$ are not required for understanding the paper. We instead focus on the representation of states and the goal condition.

States Each symbol $\sigma \in \Sigma_p \cup \Sigma_f$ is associated with an arity $\text{arity}(\sigma) \in \mathbb{N} \cup \{0\}$. Predicates and functions take the form $p(x_1, \ldots, x_{n_p})$ and $f(x_1, \ldots, x_{n_f})$ respectively, where the x_i denotes the ith argument. Propositional and numeric variables are defined by substituting objects into predicate and function variables. A state s is an assignment of values in $\{\top, \bot\}$ (resp. $\mathbb{R}$) to all possible propositional (resp. numeric) variables in a state. Following the closed world assumption, we can equivalently represent a state as a set of true propositions and numeric assignments.

Goal Condition A propositional condition is a positive (resp. negative) literal $x = \top$ (resp. $x = \bot$) where x is a propositional variable. A numeric condition has the form $\xi \trianglerighteq 0$ where ξ is an arithmetic expression over numeric variables and $\trianglerighteq \in \{\geq, >, =\}$. The goal condition $\mathcal{G}$ is a set of propositional and numeric conditions which we denote $\mathcal{G}_p$ and $\mathcal{G}_n$, respectively. A state s satisfies the goal condition $\mathcal{G}$ if s satisfies all its conditions.

Domain A planning domain is a set of planning tasks which share the same set of predicates, functions and action schemata. Constant objects are objects that occur in all planning tasks in a domain.

3.3 Graphs

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

4 Graph Representations

Representations of planning tasks into ML models have a great impact on both learning and planning performance. Graph representations are particularly well-suited due to their ability to model relational information of planning tasks, as well as their ability to model arbitrarily large planning tasks. In this section, we unify the graph representations of planning tasks in the learning for planning literature by taxonomising graph definitions. Next, we summarise theoretical expressivity results concerning such graphs through the lens of their ability to distinguish planning tasks in conjunction with message passing graph neural networks (MPNNs) [GSR+17].

4.1 Graph Taxonomy

The use of predicate logic in planning representations makes it natural for several relational or graph representations to arise from planning tasks. Indeed, early graph representations for planning tasks were geared towards constructing algorithms or heuristic functions for planners, starting with the planning graph which was a vital component of various early planners [BF97, HN01, GS02], to the usage of various different graphs and graph algorithms for computing transformations and heuristics of planning tasks [Hel04, HD09], and also to the use of detecting planning task symmetries [PZR11, SKH+15, SRWK19].

On the learning side, graph representations were heavily relied on for representing planning tasks given the unbounded nature of planning task sizes. ASNets [TTTX18, TTTX20] was the seminal work in this field, which made use of MPNNs for learning policies for probabilistic planning tasks, and was recently extended for numeric planning [WT24]. Later MPNN approaches for planning focused on learning heuristic or value functions [STT20, SBG22, SBG23, CTT24a, CTT24b, CT24a], with exceptions being learning policy rules [DML+19], portfolios [MFH+20] and detecting object importance [SCC+21].

An underlying component of all such learning works is that planning tasks are represented as some form of graph. However, there is no such work which compares all such definitions and provide a high-level view on similar or different characteristics of such representations. In our ongoing work, we have identified following classification of many existing graph representations of planning tasks:

(1) Grounded Graphs – nodes represent all possible ground propositions and actions of planning tasks, and edges are defined from precondition and effect relations of actions. Example graphs include ASNets [TTTX18, TTTX20], STRIPS-HGN [STT20] and the SLG in GOOSE [CTT24a].

(2) Lifted Graphs with Instantiation Relation (IR) – nodes represent task objects and only propositions that are true in the state or the goal condition, and edges are defined by the relations between object instantiations in ground propositions. Example graphs include Muninn [SBG22] and the ILG in GOOSE [CTT24a].

(3) Lifted Graphs with Predicate Relation (PR) – nodes represent only task objects, and edges are defined by the location of pairs of objects in n-ary predicates for $n \geq 2$. This can also be viewed as a line graph of lifted graphs with Object Relation (OR), where nodes represent only task atoms, and edges between atoms sharing an object. Example graphs include the PLOI graph [SCC+21] and the Object Binary Structure [HŠ24].

4.2 Graph Hierarchy

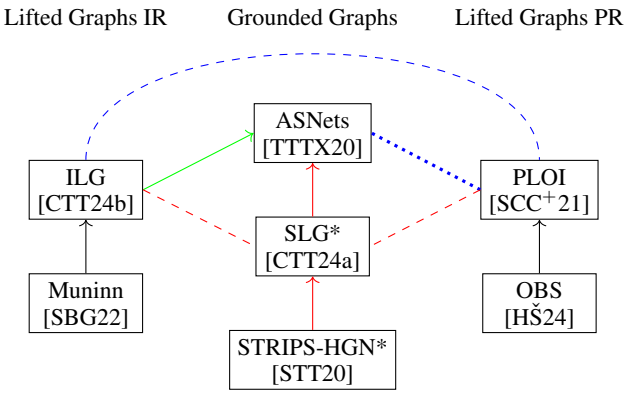
On top of achieving a taxonomy, we would like to understand whether there are any theoretical or practical relationships between such graphs. Existing [CTT24a, CTT24b] and ongoing work of authors have attempted to understand such relationships through the lens of expressive power.

More specifically, we compare the expressive power of graph representations based on planning tasks they can distinguish when used with MPNNs. Results primarily bootstrap from existing works measuring the distinguishability of existing general graph learning models by making use of the well-known result that the colour refinement or Weisfeiler-Leman (WL) algorithm subsumes MPNN expressivity [MRF+19, XHLJ19], with novelty lying in identifying the effect of graph representations of planning tasks. However, we note that our results differ from graph learning literature results which focus on expressiveness [MRM20, MRKR22, ZSA22, WCW+23, AKG24] of *architectures* rather than *representations* as the graph representations are assumed to be fixed in graph learning datasets. Results are summarised in Fig. 4 and key takeaways are:

- grounded representations implicitly encode instantiation relations and thus are more expressive than lifted representations with IR (green edges)
- not encoding planning domain information (predicates and schemata) results in lower expressivity across different graph classes (red edges)
- lifted graphs with PR are incomparable to both lifted graphs with IR, and grounded graphs under a weaker notion of expressivity (blue edges)

5 Graph Learning Models

Another core component of a graph learning approach for planning is the graph learning model itself. ML techniques



X ⟶ Y	Y is strictly more expressive than X
X - - - Y	X and Y are incomparable
X ····· Y	X and Y are incomparable with no h^* difference requirement
X*	X is used for multi-domain heuristics

Figure 4: Expressivity hierarchy of graph representations of planning tasks. Colours of edges denote the key takeaways classified in Section 4.2.

can generally be classified into deep learning and classical taxonomies. Deep learning [LBH15] pipelines automatically compute *parameterised encoder* functions which convert raw input data into latent features which they deem useful for inferring outputs, generally using some form of neural network. In contrast, classical machine learning pipelines predefine the *feature extractor* for converting raw input data into feature vectors which are used by an arbitrary chosen downstream inference model, such as a regression model or decision tree.

Classical ML is Better Suited than Deep Learning for Planning

The graph learning equivalent of such models include GNNs and graph kernels. Due to the popularity of deep learning, almost all recent works in learning for planning since the introduction of ASNets employ some variant of GNN as the underlying learning model. However, it has been shown very recently that classical ML approaches such as linear graph kernels significantly outperform GNN approaches for planning [CTT24b], over various metrics and often by several orders of magnitude.

This approach was again motivated by the simple, yet well-known result that the WL algorithm upper bounded the expressivity of MPNNs [MRF+19, XHLJ19], and such an algorithm can be converted into the WL graph kernel for extracting features for graphs [SSVL+11]. The WL graph kernel has the same polynomial computational complexity as the typical MPNN but without the need to perform matrix operations which increases its runtime often by a significant constant factor. Furthermore, the features can be combined with a simple linear model which lead to explainable and very fast models in terms of training and evaluation. Given that planning is a time-sensitive task and graph learning models may be called many times during planning, such speedups from using classical ML approaches over deep learning methods can lead to greater gains. It is also the case that generating training labels for many and large planning tasks is expensive such that one should use models with high data efficiency.

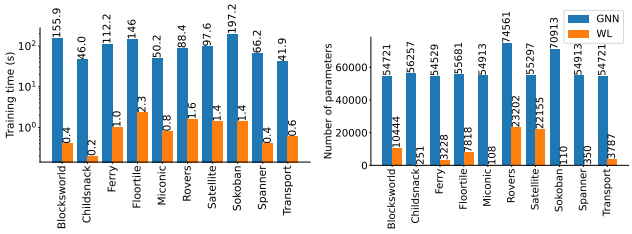


Figure 5: Time to train after data preprocessing in log scale (left) and number of learnable parameters (right) of GNN and WL models on various planning domains.

Indeed, Fig. 5 shows how linear models using WL features exhibit significantly faster training times and fewer parameters than GNN models, and Section 7 later provides results also showcasing better planning performance. We can further contrast this with the era of scaling with LLMs exhibiting parameters in the order of billions and training data and times so significant that they exhibit large monetary costs. This story concerning cheap models outperforming large models still holds true in numeric planning [CT24a], where one may think neural networks may be more suited to reasoning over numbers as function approximators.

6 Optimisation

Typical machine learning tasks have a well defined problem to solve, such as classification or regression, which usually give rise to obvious loss functions to optimise. Conversely, the focus of planning generally involves solving problems efficiently, and in some cases optimising over solution quality. Also in contrast to Reinforcement Learning, planning does not exhibit dense reward signals which can be used to guide learning. Thus, there is not obvious method for deciding the optimisation problem to solve for training learners for planning.

6.1 Learning Policies

The seminal deep learning for planning model, ASNets [TTTX18, TTTX20], learns policies for solving planning tasks in a RL-style fashion. A benefit of learning policies is that when learned correctly, execution of policies is linear time in the size of the solution. Furthermore, some planning domains only require a 'trick' to be learned in order to solve them which can be represented by a neural network architecture. One downside of learning policies is that execution of such policies have no completeness guarantees,

meaning that execution of the policy may not return goal-reaching plan if it exists. Thus, one usually combines the use of policies with search to achieve completeness [STT+19]. Orthogonally one may allow a user to provide advice or background knowledge to the learning planner for defining the solution space for a planning domain, and instead focus the learner on *optimising* solution quality, which is a more difficult task than solving domains we may already know the answer to [CHŠ24].

6.2 Learning Optimal Heuristic Functions

Common L4P approaches exploit the fact that most state-of-the-art planners are based on heuristic search, giving rise to the idea of learning reusable heuristic functions[2] in a supervised fashion for guiding search. Contrary to policies, learned heuristic functions used with search algorithms such as GBFS are complete and will always (eventually) return a solution if one exists.

However, most approaches end up performing the naive approach of trying to learn the optimal h^* heuristic for a domain from samples via mean squared error loss. This is generally not the best idea as theoretical arguments can be made from computational complexity theory, or explicit examples [CTT24a] stating that learning h^* is not possible for some domains, and furthermore, it may not even be necessary to just solve planning tasks. This is because h^* is derived from optimal solutions which are stronger than arbitrary solutions for planning tasks. Furthermore, optimising cost-to-go as mean squared error (MSE) estimates does not match up with the original derivation of MSE loss as maximising likelihood under Gaussian distribution assumptions, which is not the case for planning heuristic values [NAMF24].

Nevertheless, the reason why researchers generally opt to learn h^* is because it is a canonical label for training tasks that can be computed automatically and efficiently. More specifically, it is easier to run an optimal planner on an arbitrary task to receive optimal plans and h^* labels, than to manually label tasks with domain-dependent polynomial-time heuristic functions that still guide search efficiently. One such example of heuristics with this property are dead-end avoiding, descending heuristics [SPRH16]. Another reason for learning h^* is that in the best case scenario where h^* is learned correctly, problems can be solved in linear time with respect to solution size.

6.3 Learning Ranking Functions

Garrett et al. [GKL16] proposed to frame heuristic functions not as learning cost-to-go estimates but as ranking states for expansion in GBFS. An advantage of ranking as a learning task is that it aligns better with the intent of GBFS and furthermore allows for a larger hypothesis space of solutions. This idea was extended to neural network architectures in independent works [CEKP23, HTT+24]. However, the latter work identified that the data used in [CEKP23] is quadratic in the training plans by encoding all possible ranking pairs, but can be reduced to a linear number of rankings by exploiting the fact that ranking is a transitive relation.

[2]Drawing analogies to RL, this is similar to learning optimal value functions across multiple tasks.

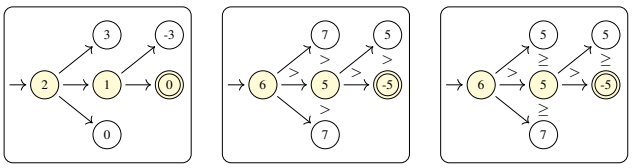


Figure 6: Visualisations of heuristics that achieve zero loss when optimising cost-to-go (left), classification ranking (middle), and constrained optimisation ranking (right). Yellow nodes correspond to optimal plan traces, and white nodes the siblings of trace states.

Classification Ranking can be interpreted as a classification task on pairs of states. This has been modelled and solved by RankSVMs in [GKL16] or by backpropagation in [HTT+24]. Ranking formulations treating pairs of distinct states (s, t) as single data point that can be labelled as either 1 or -1, where 1 means that s is ranked strictly better than t, and vice versa for -1. True labels are derived from optimal plan traces where $(s, t) = 1$ if t is either a parent or sibling of s in an optimal plan trace. After learning is performed, point-wise GBFS heuristic functions are extracted from the model that remain faithful to the ranking relation.

Constrained Optimisation An issue with the classification formulation is that the hypothesis space is still restrictive given that cross-entropy forces ranking differences to be close to the value 1 even when this may not be necessary to represent a useful ranking function. This issue can be handled by formulating the problem as a *constrained optimisation* problem inspired by SVMs which allows additional flexibility in heuristic predictions as long as the rankings represented by inequalities are preserved. Furthermore, there is also room to model the difference between comparing optimal states against its siblings which may be equally good ($\geq$) and against its parent which will always be worse with positive action costs ($>$) inequalities. Such an example can be seen in the LP encoding in [CT24a]. Fig. 6 illustrates the differences between learning heuristic functions as cost-to-go estimates, classification rankings, and constrained optimisation rankings.

7 Experimental Results

In this section, we provide an outline of experimental results arising from our work on graph learning for both classical (Section 7.1) and numeric (Section 7.2) planning, and summarise key takeaways (Section 7.3).

7.1 Classical Planning

For classical planning, we use the benchmarks from the Learning Track of the 2023 International Planning Competition (IPC23LT) [TAE+24]. The IPC23LT benchmarks emphasise generalisation across number of objects, where as displayed in Fig. 7, the testing tasks are often up to an order of magnitude larger than the training tasks in the number of objects. Furthermore, sizes of planning task state spaces

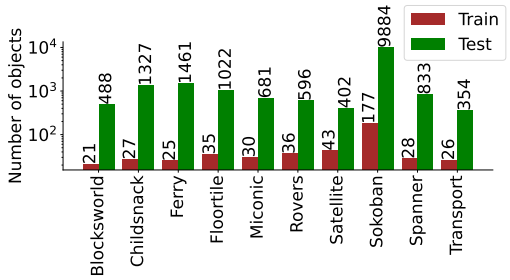


Figure 7: Sizes of training and testing tasks in the IPC23LT.

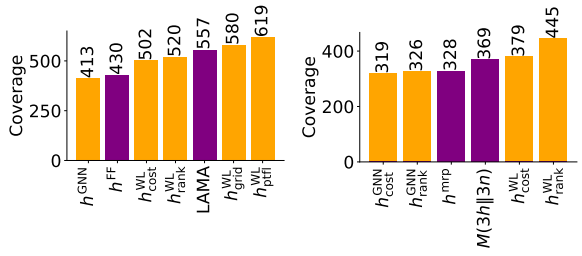


Figure 8: Coverage of classic/numeric (purple) and learning (orange) planners on classical (left) and numeric (right) encodings of the IPC23LT. Higher values are better (↑).

generally scale in a high polynomial variable with respect to the number of objects. The primary metric we discuss in this section is the coverage of graph learning planners across all 900 problems (10 domains $\times$ 90 problems) in the IPC23LT within a fixed 30 minute time and 8GB memory limit for testing, i.e. solving a single planning task. Fig. 8 displays the coverage of various baseline planners and graph learning planner configurations abbreviated as follows:

- h^{FF}: GBFS + the h^{FF} heuristic [HN01]
- LAMA: a strong satisficing planner which uses multiple queues, helpful actions, and lazy heuristic evaluation [RW10]
- h^{GNN}: GBFS + a GNN heuristic trained with mean squared error loss on h^* values, detailed in [CTT24b]
- $h^{\text{WL}}_{\text{cost}}$: GBFS + a linear WL heuristic trained with Gaussian Process Regression, detailed in [CTT24b]
- $h^{\text{WL}}_{\text{rank}}$: GBFS + a linear WL heuristic trained with the LP ranking formulation in [CT24a]
- $h^{\text{WL}}_{\text{grid}}$: GBFS + a grid search over WL heuristic configurations performed on each domain
- $h^{\text{WL}}_{\text{ptfl}}$: GBFS + a parallel portfolio over WL heuristic configurations

7.2 Numeric Planning

In [CT24a], we also performed experiments on numeric encodings of domains from the IPC23LT, with the exception of two domains which have no benefit from numeric encodings. Fig. 8 displays the coverage of various baseline numeric planners and graph learning planner configurations abbreviated as follows:

- h^{mrp}: GBFS + the h^{mrp} heuristic [SSSG20]
- $M(3h\|3n)$: state-of-the-art numeric planner with multiple queues and novelty heuristics [CT24b]
- $h^{\text{GNN}}_{\text{cost}}$: GBFS + a GNN heuristic trained with mean squared error loss on h^* values
- $h^{\text{GNN}}_{\text{rank}}$: GBFS + a GNN heuristic trained with a differentiable ranking loss
- $h^{\text{WL}}_{\text{cost}}$: GBFS + a linear WL heuristic trained with Gaussian Process Regression
- $h^{\text{WL}}_{\text{rank}}$: GBFS + a linear WL heuristic trained with a LP ranking formulation

7.3 Key Takeaways

Classical ML consistently outperform deep learning for symbolic planning From Fig. 8, we observe that classical ML approaches using learned WL heuristics outperform their deep learning counterparts over different optimisation formulations in terms of coverage. In classical planning, $h^{\text{WL}}_{\text{cost}}$ outperforms h^{GNN} by 89 problems, while in numeric planning, $h^{\text{WL}}_{\text{cost}}$ (resp. $h^{\text{WL}}_{\text{rank}}$) outperforms $h^{\text{GNN}}_{\text{cost}}$ (resp. $h^{\text{GNN}}_{\text{rank}}$) by 60 (resp. 119) problems.

Learned ranking functions consistently outperform learned cost-to-go estimates From Fig. 8, we note that learned ranking heuristics outperform their cost-to-go estimate counterparts over graph learning models in terms of coverage. In classical planning, $h^{\text{WL}}_{\text{rank}}$ outperforms $h^{\text{WL}}_{\text{cost}}$ by 18 problems, while in numeric planning, $h^{\text{WL}}_{\text{rank}}$ (resp. $h^{\text{GNN}}_{\text{rank}}$) outperforms $h^{\text{WL}}_{\text{cost}}$ (resp. $h^{\text{GNN}}_{\text{cost}}$) by 66 (resp. 7) problems.

Learned heuristics with simple search are competitive with strong planners From Fig. 8 for classical planning, we observe that single instantiations of learned WL heuristics (502 and 520) with GBFS alone do not outperform the strong LAMA planner (557). However, taking the best configuration for each domain (580), or using multiple threads for parallelised portfolios (619), learned heuristics with simple search algorithms can be competitive with planners employing stronger search algorithms. However, for numeric planning, learned heuristics with GBFS alone (445) are a lot more competitive with numeric planners (369).

8 Open Challenges

The general L4P field is still in its infancy and poses multiple open challenges for future research. This is reflected by the fact that there are still many domains and problems in the tested benchmarks that remain unsolved for L4P approaches. We highlight the themes of core open challenges that we believe are crucial for advancing the state-of-the-art.

I Expressivity

A core challenge that needs to be addressed by L4P approaches is the concept of expressivity: the ability of a model to represent solutions to planning domains. Propositional planning is PSPACE-complete in general [Byl94] but even many domains that are solvable in polynomial time cannot be solved by learning approaches [SBG22,

CTT24a]. Recent L4P approaches use some variant of message passing neural networks which are known to be theoretically bounded by two-variable counting logics [MRF+19, XHLJ19, BKM+20, Gro21], with some works studying the effect of (approximate) higher-order graph learning approaches [CTT24b, SBG24] and performing distinguishability tests of models [HŠ24, DSBG24].

However, this direction of research mirrors the graph learning literature of building more expressive models [MRM20, KJM20, ACGL21, BHG+21, MRKR22, FCL+22, ZSA22, ZJAS22, WCW+23, BFZB23, AKG24] at the cost of runtime complexity, unclear generalisation performance and unclear relevance to downstream tasks [MFD+24]. Furthermore, tractable graph learning architectures alone have yet to be able to achieve expressivity for the basic P-time complexity time which is not directly achievable with finite-variable counting logics, and bounded number of message passing layers.

Possible directions for research in expressivity involve building or making use of models which can handle recursion such as inductive logic programming [CDEM22] or Generalised Planning [Sri10, CAJ19] approaches, motivated by the fact that P is captured by first-order logic with transitive closure [Var82, Imm82]. Indeed, some earlier works have studied rule-based approaches for learning policies or subgoals for planning [Kha99, GT04, IM19, BG24, DSG24], with recent work explicitly studying the use of Datalog for directly encoding planning domain solutions and solvability [GRH24, CHŠ24].

II Generalisation

Another key challenge in L4P is understanding both theoretical and empirical generalisation results for planning. L4P is inherently an out-of-distribution task as testing tasks are arbitrarily large and hence are drawn from a different distribution from bounded-size training tasks. Thus, exploiting common generalisation theory tools such as VC dimension [Vap98] and Rademacher complexity [BM01], which assume similar training and testing probability distributions, for bounding generalisation theory is not straightforward.

Conversely, planning representations often contain information encoded as rich relational or logical structures. Related to the concept of expressivity previously, researchers have developed theoretical frameworks to better understand the behaviour of planning domains such as novelty width [LG12], correlation complexity [SPRH16], the river measure [DH24a], and methods to bound such measures [DH24b]. These tools may provide insights for developing generalisation theory for planning.

III Optimisation

As discussed in Section 6, there is no clear consensus on the best optimisation criteria for learning for planning as this may depend on the domain, learning architecture and also training data. Although ranking as discussed provides a much better suited optimisation criteria for learning heuristics for planning, it may be the case that learning other forms of domain knowledge such as policies, subgoals [DSG24], and search effort estimates [OL21, FGT+22] may be easier and well-suited for specific domains. In other words, there is no single optimisation criterion that is best for all planning domains. Theoretical and empirical results on optimisation criteria that are well suited for specific planning domain characteristics are still unknown.

IV Collecting Data

Another challenge in L4P is the problem of deciding how and what data to collect for training, mirroring the exploration-exploitation tradeoff in RL [SB98]. This also relates to the optimisation problem as the choice of optimisation criteria depends on the type of data collected. For example, optimal plan traces are sufficient for learning cost-to-go or ranking heuristic functions, but not optimal policies. This is because there may be more than one optimal action to take at each state that cannot be deduced from optimal plans. ASNets [TTTX18, TTTX20] handles this issue similar to the RL approach of exploring with a partially learned policy and exploiting with a teacher planner as proxy for a reward function. Learning subgoals as policy sketches requires expanding entire state spaces [DSG24] which does not scale to high predicate or asymmetric domains, even with symmetry detection [DSBG24].

Similarly to the optimisation problem, theoretical and empirical results on how to best collect and how much data to collect for L4P are much appreciated.

V Fair Comparisons

A final challenge in L4P is the problem of fairly comparing different methodologies and approaches, given the additional experimental variables of training data and training time introduced when learning is involved. Thus, this resulted in various researchers and groups employing different benchmarks and evaluation strategies. The Learning Track of the 2023 International Planning Competition [TAE+24] was a welcome addition for standardising the set of planning domains and testing tasks. However, although training tasks are given, the method of generating useful labels is not standardised and instead competitors were given a fixed time limit to both generate labels and train models. This leads to an undesirable scheduling task of trading off between data generation and training time.

Possible suggestions for future proposed benchmarks or good practices to follow include (1) providing all baselines with the same set of *labelled* training data generated by a fixed time limit that benefits all models, and/or (2) recording training time and amount of data used for all baselines.

9 Conclusion

Learning for Planning (L4P) is an increasingly popular research field, with planning being one of the few problems in AI that has been unsolved by deep learning and large models [VMH+23, VMSK23, VSK24]. This paper summarises the contributions of the authors in L4P, with a specific focus on using cheap and efficient graph learning approaches for planning. Furthermore, we have identified 5 key challenges in L4P that still remain unsolved and offer plenty of opportunities for future research for progressing the field.

References

[ACGL21] Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *IJCAI*, 2021.

[AKG24] Nurudin Alvarez-Gonzalez, Andreas Kaltenbrunner, and Vicenç Gómez. Improving subgraph-gnns via edge-level ego-network encodings. *Trans. Mach. Learn. Res.*, 2024, 2024.

[BF97] Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artif. Intell.*, 90(1-2):281–300, 1997.

[BFZB23] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(1), 2023.

[BG24] Blai Bonet and Hector Geffner. General policies, subgoal structure, and planning width. *J. Artif. Intell. Res.*, 80:475–516, 2024.

[BHG+21] Muhammet Balcilar, Pierre Héroux, Benoit Gaüzère, Pascal Vasseur, Sébastien Adam, and Paul Honeine. Breaking the limits of message passing graph neural networks. In *ICML*, 2021.

[BKM+20] Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *ICLR*, 2020.

[BM01] Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. In *COLT/EuroCOLT*, 2001.

[Byl94] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[CAJ19] Sergio Jiménez Celorrio, Javier Segovia Aguas, and Anders Jonsson. A review of generalized planning. *Knowl. Eng. Rev.*, 34:e5, 2019.

[CDEM22] Andrew Cropper, Sebastijan Dumancic, Richard Evans, and Stephen H. Muggleton. Inductive logic programming at 30. *Mach. Learn.*, 111(1):147–172, 2022.

[CdlRF+12] 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.

[CEKP23] Leah Chrestien, Stefan Edelkamp, Antonín Komenda, and Tomás Pevný. Optimize planning heuristics to rank, not to estimate cost-to-goal. In *NeurIPS*, 2023.

[CHŠ24] Dillon Ze Chen, Rostislav Horčík, and Gustav Šír. Deep learning for generalised planning with background knowledge. *CoRR*, abs/2410.07923, 2024.

[CT24a] Dillon Z. Chen and Sylvie Thiébaux. Graph learning for numeric planning. In *NeurIPS*, 2024.

[CT24b] Dillon Z. Chen and Sylvie Thiébaux. Novelty heuristics, multi-queue search, and portfolios for numeric planning. In *SOCS*, 2024.

[CTT24a] Dillon Z. Chen, Sylvie Thiébaux, and Felipe Trevizan. Learning domain-independent heuristics for grounded and lifted planning. In *AAAI*, 2024.

[CTT24b] Dillon Z. Chen, Felipe Trevizan, and Sylvie Thiébaux. Return to tradition: Learning reliable heuristics with classical machine learning. In *ICAPS*, 2024.

[DH24a] Simon Dold and Malte Helmert. Higher-dimensional potential heuristics: Lower bound criterion and connection to correlation complexity. In *ICAPS*, 2024.

[DH24b] Simon Dold and Malte Helmert. Novelty vs. potential heuristics: A comparison of hardness measures for satisficing planning. In *AAAI*, 2024.

[DML+19] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. In *ICLR*, 2019.

[DSBG24] Dominik Drexler, Simon Ståhlberg, Blai Bonet, and Hector Geffner. Equivalence-based abstractions for learning general policies. In *KR*, 2024.

[DSG24] Dominik Drexler, Jendrik Seipp, and Hector Geffner. Expressing and exploiting subgoal structure in classical planning using sketches. *J. Artif. Intell. Res.*, 80, 2024.

[FCL+22] Jiarui Feng, Yixin Chen, Fuhai Li, Anindya Sarkar, and Muhan Zhang. How powerful are k-hop message passing graph neural networks. In *NeurIPS*, 2022.

[FGT+22] Patrick Ferber, Florian Geißer, Felipe Trevizan, Malte Helmert, and Jörg Hoffmann. Neural network heuristic functions for classical planning: Bootstrapping and comparison to other methods. In *ICAPS*, 2022.

[FL03] Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, 20:61–124, 2003.

[GB13] 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.

[GKL16] Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to rank for synthesizing planning heuristics. In *IJCAI*, 2016.

[GRH24] Claudia Grundke, Gabriele Röger, and Malte Helmert. Formal representations of classical planning domains. In *ICAPS*, 2024.

[Gro21] Martin Grohe. The logic of graph neural networks. In *LICS*, 2021.

[GS02] Alfonso Gerevini and Ivan Serina. LPG: A planner based on local search for planning graphs with action costs. In *AAAI*, 2002.

[GSR+17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[GT04] Charles Gretton and Sylvie Thiébaux. Exploiting first-order regression in inductive policy selection. In *UAI*, 2004.

[HD09] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*, 2009.

[Hel04] Malte Helmert. A planning heuristic based on causal graph analysis. In *ICAPS*, 2004.

[HN01] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.*, 14:253–302, 2001.

[HŠ24] Rostislav Horčík and Gustav Šír. Expressiveness of graph neural networks in planning domains. In *ICAPS*, 2024.

[HTT+24] Mingyu Hao, Felipe Trevizan, Sylvie Thiébaux, Patrick Ferber, and Jörg Hoffmann. Guiding GBFS through learned pairwise rankings. In *IJCAI*, 2024.

[IM19] León Illanes and Sheila A. McIlraith. Generalized planning via abstraction: Arbitrary numbers of objects. In *AAAI*, 2019.

[Imm82] Neil Immerman. Relational queries computable in polynomial time (extended abstract). In *STOC*, 1982.

[Kha99] Roni Khardon. Learning action strategies for planning domains. *Artif. Intell.*, 113(1-2):125–148, 1999.

[KJM20] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Appl. Netw. Sci.*, 5(1):6, 2020.

[KS21] Rushang Karia and Siddharth Srivastava. Learning generalized relational heuristic networks for model-agnostic planning. In *AAAI*, 2021.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nat.*, 521(7553):436–444, 2015.

[LG12] Nir Lipovetzky and Hector Geffner. Width and serialization of classical planning problems. In *ECAI*, 2012.

[MFD+24] Christopher Morris, Fabrizio Frasca, Nadav Dym, Haggai Maron, İsmail İlkan Ceylan, Ron Levie, Derek Lim, Michael M. Bronstein, Martin Grohe, and Stefanie Jegelka. Position: Future directions in the theory of graph machine learning. In *ICML*, 2024.

[MFH+20] Tengfei Ma, Patrick Ferber, Siyu Huo, Jie Chen, and Michael Katz. Online planner selection with graph neural networks and adaptive scheduling. In *AAAI*, 2020.

[MLTK23] Jiayuan Mao, Tomás Lozano-Pérez, Joshua B. Tenenbaum, and Leslie Pack Kaelbling. What planning problems can A relational neural network solve? In *NeurIPS*, 2023.

[MRF+19] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

[MRKR22] Christopher Morris, Gaurav Rattan, Sandra Kiefer, and Siamak Ravanbakhsh. Speqnets: Sparsity-aware permutation-equivariant graph networks. In *ICML*, 2022.

[MRM20] Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. In *NeurIPS*, 2020.

[NAMF24] Carlos Núñez-Molina, Masataro Asai, Pablo Mesejo, and Juan Fernández-Olivares. On using admissible bounds for learning forward search heuristics. In *IJCAI*, 2024.

[OL21] Laurent Orseau and Levi H. S. Lelis. Policy-guided heuristic search with guarantees. In *AAAI*, 2021.

[PZR11] Nir Pochter, Aviv Zohar, and Jeffrey S. Rosenschein. Exploiting problem symmetries in state-based planners. In *AAAI*, 2011.

[RW10] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res.*, 39:127–177, 2010.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[SBG22] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *ICAPS*, 2022.

[SBG23] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general policies with policy gradient methods. In *KR*, 2023.

[SBG24] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general policies for classical planning domains: Getting beyond c_2. *CoRR*, abs/2403.11734, 2024.

[SCC+21] 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*, 2021.

[SKH+15] Alexander Shleyfman, Michael Katz, Malte Helmert, Silvan Sievers, and Martin Wehrle. Heuristics and symmetries in classical planning. In *AAAI*, 2015.

[SPRH16] Jendrik Seipp, Florian Pommerening, Gabriele Röger, and Malte Helmert. Correlation complexity of classical planning domains. In *IJCAI*, 2016.

[Sri10] Siddharth Srivastava. *Foundations and Applications of Generalized Planning*. PhD thesis, University of Massachusetts Amherst, 2010.

[SRWK19] Silvan Sievers, Gabriele Röger, Martin Wehrle, and Michael Katz. Theoretical foundations for structural symmetries of lifted PDDL tasks. In *ICAPS*, 2019.

[SSSG20] Enrico Scala, Alessandro Saetti, Ivan Serina, and Alfonso Emilio Gerevini. Search-guidance mechanisms for numeric planning through subgoaling relaxation. In *ICAPS*, 2020.

[SSVL+11] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011.

[STT+19] William Shen, Felipe W. Trevizan, Sam Toyer, Sylvie Thiébaux, and Lexing Xie. Guiding search with generalized policies for probabilistic planning. In *SOCS*, 2019.

[STT20] William Shen, Felipe Trevizan, and Sylvie Thiébaux. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *ICAPS*, 2020.

[TAE+24] Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fiser, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 international planning competition. *AI Mag.*, 45(2):280–296, 2024.

[TTTX18] Sam Toyer, Felipe W. Trevizan, Sylvie Thiébaux, and Lexing Xie. Action schema networks: Generalised policies with deep learning. In *AAAI*, 2018.

[TTTX20] Sam Toyer, Sylvie Thiébaux, Felipe Trevizan, and Lexing Xie. Asnets: Deep learning for generalised planning. *J. Artif. Intell. Res.*, 68:1–68, 2020.

[Vap98] Vladimir Vapnik. *Statistical learning theory*. Wiley, 1998.

[Var82] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, 1982.

[VMH+23] Karthik Valmeekam, Matthew Marquez, Alberto Olmo Hernandez, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *NeurIPS*, 2023.

[VMSK23] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models - A critical investigation. In *NeurIPS*, 2023.

[VSK24] Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. Llms still can't plan; can lrms? A preliminary evaluation of openai's o1 on planbench. *CoRR*, abs/2409.13373, 2024.

[WCW+23] Qing Wang, Dillon Ze Chen, Asiri Wijesinghe, Shouheng Li, and Muhammad Farhan. $\mathcal{N}$-wl: A new hierarchy of expressivity for graph neural networks. In *ICLR*, 2023.

[WT24] Ryan X. Wang and Sylvie Thiébaux. Learning generalised policies for numeric planning. In *ICAPS*, 2024.

[XHLJ19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[ZJAS22] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *ICLR*, 2022.

[ZSA22] Lingxiao Zhao, Neil Shah, and Leman Akoglu. A practical, progressively-expressive GNN. In *NeurIPS*, 2022.