Learning Domain-Independent Heuristics for Grounded and Lifted Planning

Dillon Z. Chen[1,2], **Sylvie Thiébaux**[1,2], **Felipe Trevizan**[1]

[1]School of Computing, The Australian National University
[2]LAAS-CNRS, Université de Toulouse
{dillon.chen, sylvie.thiebaux}@laas.fr, felipe.trevizan@anu.edu.au

Abstract

We present three novel graph representations of planning tasks suitable for learning domain-independent heuristics using Graph Neural Networks (GNNs) to guide search. In particular, to mitigate the issues caused by large grounded GNNs we present the first method for learning domain-independent heuristics with only the lifted representation of a planning task. We also provide a theoretical analysis of the expressiveness of our models, showing that some are more powerful than STRIPS-HGN, the only other existing model for learning domain-independent heuristics. Our experiments show that our heuristics generalise to much larger problems than those in the training set, vastly surpassing STRIPS-HGN heuristics.

1 Introduction

Graph Neural Networks (GNNs) have recently attracted the interest of the planning community, for learning heuristic cost estimators, task orderings, value functions, action policies, and portfolios, to name a few (Shen, Trevizan, and Thiébaux 2020; Garg, Bajpai, and Mausam 2020; Karia and Srivastava 2021; Ståhlberg, Bonet, and Geffner 2022a; Ma et al. 2020; Sharma et al. 2022; Teichteil-Königsbuch et al. 2023). GNNs exhibit great generalisation potential, since once trained, they offer outputs for any graph, regardless of size or structure. Representing the structure of planning domains as graphs, GNNs can train on a set of small problems to learn generalised policies and heuristics that apply to all problems in a domain. As noted by Shen, Trevizan, and Thiébaux (2020), this also allows for learning heuristics applicable to multiple domains, or even domain-independent heuristics that apply to domains unseen during training.

In this paper, we explore the use of GNNs for learning both domain-dependent and domain-independent heuristics for classical planning, with an emphasis on the latter. To the best of our knowledge, STRIPS-HGN (Shen, Trevizan, and Thiébaux 2020) is the only existing model designed to learn domain-independent heuristic functions from scratch. The models in (Ståhlberg, Bonet, and Geffner 2022a) are inherently domain-dependent, given that they use different update functions for predicates of the planning problems, and hence

cannot generalise to unseen problems with a different number of predicates. Neural Logic Machines (Dong et al. 2019; Gehring et al. 2022) are also domain-dependent models as they assume a maximum arity of input predicates.

STRIPS-HGN has several drawbacks when learning domain-independent heuristics: (1) its hypergraph representation of planning tasks ignores delete lists and thus cannot theoretically learn h^*, (2) its aggregation function is not permutation invariant due to ordering the neighbours of each node which may prevent it from generalising effectively, (3) it assumes a bound on the sizes of action preconditions and effects, meaning that it also has to discard certain edges in its hypergraph in its message updating step, and (4) it requires constructing the whole grounded hypergraph, whose size becomes impractical for large problems.

Our contributions remedy these issues and make the following advances to the state of the art. Building on well-known planning formalisms, namely propositional STRIPS, FDR, and lifted STRIPS, we define novel grounded and lifted graph representations of planning tasks suitable for learning domain-independent heuristics. In particular, this results in the first domain-independent GNN heuristic based on a lifted graph representation. We also establish the theoretical expressiveness of Message-Passing Neural Networks (MPNN) acting upon our graphs in terms of the known domain-independent heuristics they are able to learn, and suggest further research directions for learning h^*.

We then conduct two sets of experiments to complement our theory and evaluate the effectiveness of learned heuristics. The first set aims to measure the accuracy of learned heuristics on unseen tasks, while the second set evaluates the effectiveness of such learned heuristics for heuristic search. Planners guided by heuristics learnt using our new graphs solve significantly larger problems than those considered by Shen, Trevizan, and Thiébaux (2020), Karia and Srivastava (2021) and Ståhlberg, Bonet, and Geffner (2022b). In the domain-dependent setting, planners guided by our lifted heuristics achieves greater coverage than using h^{FF} in several domains and returns lower cost plans overall.

2 Background and Notation

Planning A classical planning task (Geffner and Bonet 2013) is a state transition model $\Pi = \langle S, A, s_0, G \rangle$ consisting of a set S of states, a set A of actions, an initial state s_0,

and a set G of goal states. Each action $a \in A$ is a function $a : S \to S \cup \perp$ mapping a state s in which the action is applicable to its successor $a(s)$, and states in which it is not applicable to $\perp$. The cost of the action is $c(a) \in \mathbb{N}$. A solution or a *plan* in this model is a sequence of actions $\pi = a_1, \ldots, a_n$ such that $s_i = a_i(s_{i-1}) \neq \perp$ for all $i \in \{1, \ldots, n\}$ and $s_n \in G$. In other words, a plan is a sequence of applicable actions which when executed, progresses our initial state to a goal state. The cost of π is $c(\pi) = \sum_{i=1}^{n} c(a_i)$. A planning task is *solvable* if there exists at least one plan. We now describe three ways to represent planning tasks.

A *STRIPS planning task* is a tuple $\Pi = \langle P, A, s_0, G \rangle$ with P a set of propositions (or facts), A a set of actions, $s_0 \subseteq P$ an initial state, and $G \subseteq P$ the goal condition. A state s is a subset of P and is a goal state if $G \subseteq s$. An action $a \in A$ is a tuple $\langle \mathrm{pre}(a), \mathrm{add}(a), \mathrm{del}(a) \rangle$ with $\mathrm{pre}(a), \mathrm{add}(a), \mathrm{del}(a) \subseteq P$ and $\mathrm{add}(a) \cap \mathrm{del}(a) = \emptyset$, and has an associated cost $c(a) \in \mathbb{R}$. The action is applicable in a state s if $\mathrm{pre}(a) \subseteq s$, and leads to the successor state $s' = (s \setminus \mathrm{del}(a)) \cup \mathrm{add}(a)$.

An *FDR planning task* (Helmert 2009) is a tuple $\Pi = \langle \mathcal{V}, A, s_0, s_\star \rangle$ where $\mathcal{V}$ is a finite set of state variables v, each with a finite domain D_v. A fact is a pair $\langle v, d \rangle$ where $v \in \mathcal{V}, d \in D_v$. A partial variable assignment is a set of facts where each variable appears at most once. A total variable assignment is a partial variable assignment where each variable appears once. The initial state s_0 is a total variable assignment and the goal condition $s_\star$ is a partial variable assignment. Again, A is a set of actions of the form $a = \langle \mathrm{pre}(a), \mathrm{eff}(a) \rangle$ where $\mathrm{pre}(a)$ and $\mathrm{eff}(a)$ are partial variable assignments. An action a is applicable in s if $\mathrm{pre}(a) \subseteq s$, and leads to the successor state $s' = (s \cup \mathrm{eff}(a)) \setminus \{\langle v, d \rangle \in s \mid \exists d' \in D_v, \langle v, d' \rangle \in \mathrm{eff}(a) \wedge d \neq d' \}$.

A *lifted planning task* (Lauer et al. 2021) 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, $\mathcal{A}$ is a set of action schema, $\mathcal{O}$ is a set of objects, s_0 is the initial state and G is the goal condition. A predicate $P \in \mathcal{P}$ has parameters $x_1, \ldots, x_{n_P}$ for $n_P \in \mathbb{N}$, noting that n_P depends on P and it is possible for a predicate to have no parameters. A predicate with n parameters is an n-ary predicate. A predicate can be instantiated by assigning some of the x_i with objects from $\mathcal{O}$ or other defined variables. A predicate where all variables are assigned with objects is grounded, and is called a ground proposition. The initial state and goal condition are sets of ground propositions. 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 $\mathrm{pre}(a), \mathrm{add}(a)$ and $\mathrm{del}(a)$ are sets of predicates from $\mathcal{P}$ instantiated with either parameter variables or objects in $\Delta(a) \cup \mathcal{O}$. Similarly to predicates, an action schema with $n = |\Delta(a)|$ parameter variables is an n-ary action schema. An action schema where each variable is instantiated with an object is an action. Action application and successor states are defined in the same way for both STRIPS and lifted planning.

Graph neural networks The introduction of graph neural networks (GNN) requires additional terminology. In the context of learning tasks, we define a graph with edge labels to be a tuple $\langle V, E, \mathbf{X} \rangle$ where V is a set of nodes,

E a set of undirected edges with labels where $\langle v, u \rangle_\iota = \langle u, v \rangle_\iota \in E$ denotes an undirected edge with label ι between nodes $u, v \in V$, and $\mathbf{X} : V \to \mathbb{R}^d$ a function representing the node features of the graph. The edge neighbourhood of a node u in a graph under edge label ι is $\mathcal{N}_\iota(u) = \{\langle u, v \rangle_\kappa \in E \mid \kappa = \iota\}$. The edge neighbourhood of a node u in a graph is $\mathcal{N}(u) = \bigcup_{\iota \in \mathcal{R}} \mathcal{N}_\iota(u)$ where $\mathcal{R}$ is the set of edge labels for the graph.

A Message-Passing Neural Network (MPNN) (Gilmer et al. 2017) is a type of GNN which iteratively updates node embeddings of a graph with edge labels locally in one-hop neighbourhoods with the general message passing equation

$$\mathbf{h}_u^{(t+1)} = \mathrm{cmb}^{(t)}\big(\mathbf{h}_u^{(t)}, \mathrm{agg}_{\langle u,v \rangle_\iota \in \mathcal{N}(u)}^{(t)} f^{(t)}\big(\mathbf{h}_u^{(t)}, \mathbf{h}_v^{(t)}, \iota\big)\big)$$

where in the t-th iteration or layer of the network, $\mathbf{h}_u^{(t)} \in \mathbb{R}^{F^{(t)}}$ is the embedding of node u of dimension $F^{(t)}$, and $\mathbf{h}_u^{(0)}$ is given by the node feature corresponding to u in $\mathbf{X}$. We have that $\mathrm{cmb}^{(t)}$ and $f^{(t)}$ are arbitrary almost everywhere differentiable functions and $\mathrm{agg}^{(t)}$ is usually a differentiable permutation invariant function acting on sets of vectors such as sum, mean or component-wise max.

In order for an MPNN to produce a graph representation for an input, it is then common to pool all the node embeddings after a number of message passing updates with a *graph readout* function Φ which is again usually given by a differentiable permutation invariant function.

3 Representation

In this section, we introduce three novel graph representations designed for learning heuristic functions for planning tasks. Each graph is tailored to a specific task representation and all of them allow us to learn *domain-independent* heuristic functions.

Grounded Graphs

A graph representation for grounded STRIPS problems already exists, namely the STRIPS problem description graph (PDG) (Shleyfman et al. 2015). It was originally used to study which classical heuristics were invariant under symmetries in the planning task. In order to learn heuristics, we provide an alternative graph representation for STRIPS problems which includes node features and edge labels.

Definition 3.1. The *STRIPS learning graph (SLG)* of a STRIPS problem $\langle P, A, s_0, G \rangle$ is the graph $\langle V, E, \mathbf{X} \rangle$ with

- $V = A \cup P$,
- $E = E_{\mathrm{pre}} \cup E_{\mathrm{add}} \cup E_{\mathrm{del}}$ where for $\iota \in \{\mathrm{pre}, \mathrm{add}, \mathrm{del}\}$

$$E_\iota = \{\langle a, p \rangle_\iota \mid p \in \iota(a), a \in A\},$$

- $\mathbf{X} : V \to \mathbb{R}^3$ defined by $u \mapsto [u \in P; u \in s_0; u \in G]$.

By allowing for edge labels, we only require one node for each proposition to encode the semantics of action effects in contrast to STRIPS PDG which requires three nodes for each proposition. Thus SLG is smaller while not losing any information. We also note that three dimensional node features are sufficient for encoding whether a node corresponds
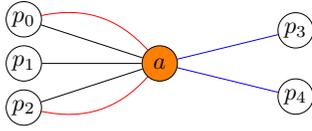
Figure 1: The SLG subgraph of an action a defined by $\mathrm{pre}(a) = \{p_0, p_1, p_2\}$, $\mathrm{add}(a) = \{p_3, p_4\}$ and $\mathrm{del}(a) = \{p_0, p_2\}$, indicated by black, blue and red edges respectively.
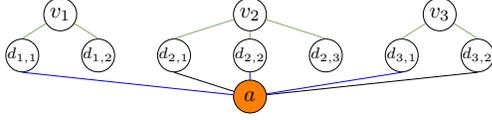

Figure 2: The FLG subgraph of an action a defined by $\mathrm{pre}(a) = \{\langle v_2, d_{2,1}\rangle, \langle v_3, d_{3,2}\rangle\}$ and $\mathrm{eff}(a) = \{\langle v_1, d_{1,1}\rangle, \langle v_2, d_{2,2}\rangle, \langle v_3, d_{3,1}\rangle\}$, indicated by black and blue edges respectively. Asparagus edges link variables and values.

to an action or proposition, and in the latter case, whether it is true in the initial state and present in the goal. Fig. 1 illustrates an example SLG subgraph.

The FDR problem description graph (PDG) (Pochter, Zohar, and Rosenschein 2011) is an existing graph representation designed to identify symmetrical states during search for FDR problems. Since PDG is not designed for learning, it lacks vector node features and edge labels. Def. 3.2 extends FDR PDG with learning in mind by retaining its graph structure and adding node features and edge labels. Fig. 2 illustrates an example of an FLG subgraph.

Definition 3.2. The *FDR learning graph (FLG)* of an FDR problem $\langle \mathcal{V}, A, s_0, s_\star \rangle$ is the graph $\langle V, E, \mathbf{X} \rangle$ with

- $V = \mathcal{V} \cup \bigcup_{v \in \mathcal{V}} D_v \cup A$,
- $E = E_{\mathrm{var:val}} \cup E_{\mathrm{pre}} \cup E_{\mathrm{eff}}$ with

$$E_{\mathrm{var:val}} = \bigcup_{v \in \mathcal{V}} \{\langle v, d \rangle_{\mathrm{var:val}} \mid d \in D_v\}$$
$$E_{\mathrm{pre}} = \bigcup_{a \in A} \{\langle d, a \rangle_{\mathrm{pre}} \mid (v, d) \in \mathrm{pre}(a)\}$$
$$E_{\mathrm{eff}} = \bigcup_{a \in A} \{\langle d, a \rangle_{\mathrm{eff}} \mid (v, d) \in \mathrm{eff}(a)\},$$

- $\mathbf{X} : V \to \mathbb{R}^5$ defined by

$$u \mapsto [u \in \mathcal{V}; u \in A; \mathrm{val}(u); \mathrm{true}(u); \mathrm{goal}(u)]$$

where $\mathrm{val}(u) = \exists v \in \mathcal{V}, u \in D_v$, $\mathrm{true}(u) = \exists v \in \mathcal{V}$, $\langle v, u \rangle \in s_0$ and $\mathrm{goal}(u) = \exists v \in \mathcal{V}, \langle v, u \rangle \in s_\star$.

Lifted Graphs

Lifted algorithms for planning offer an advantage by avoiding the need for grounding thus saving both time and memory. To leverage these benefits for heuristic learning, a lifted graph representation that is amenable to learning is needed. However, designing such graphs is non-trivial due to the extra relations to encode, namely the interactions between predicates, action schema, propositions true in the current state, the goal condition and objects. The only graph representation encoding all the information of a lifted planning task is the *abstract structure graph* (ASG) (Sievers et al. 2019). Similarly to PDG, ASG was designed to compute
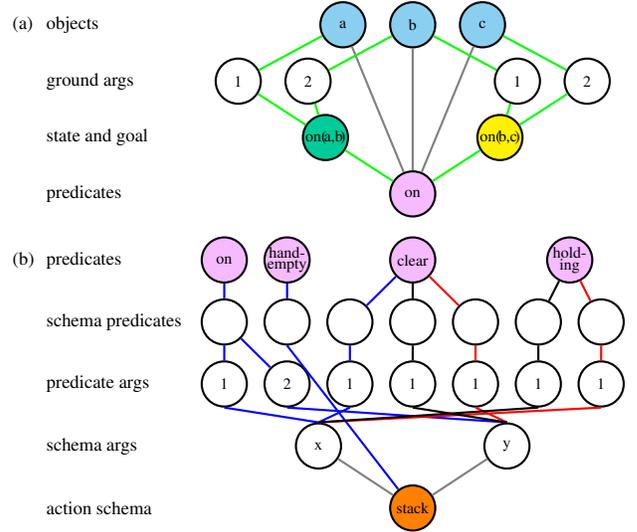

Figure 3: LLG instance subgraph (a) and schema subgraph (b) with graph layer descriptions of a Blocksworld instance.

symmetries but it has also been used for learning planning portfolios (Katz et al. 2018).

An ASG is constructed by first defining a coloured graph on *abstract structures*, a recursive structure defined with sets, tuples, and the input objects, and then defining a lifted planning task as an abstract structure. ASGs have several limitations when used with MPNNs for making predictions. Their encoding of predicate and action schema arguments is done via a sequence or directed path, where the graph uses a directed path of length n to encode n arguments. There are also many more auxiliary nodes to encode the abstract structures. These issues cause problems as a larger receptive field is required for MPNNs to learn the structure and semantics of the planning problem, and directed edges limit information flow and expressiveness when used with MPNNs.

To overcome the issues with ASGs, Def. 3.3 introduces a new graph representation for lifted planning tasks designed to be used with MPNNs. It consists of two main components: the schema subgraph which encodes the domain's action schemas and the instance subgraph which encodes the instance specific information containing the current state and the goal condition. We assume no partially instantiated action schema as is usual for most PDDL domains, but the definition could be extended to allow partial grounding as a tradeoff between expressiveness and graph size.

Definition 3.3. Let $T \in \mathbb{N}$. The *lifted learning graph (LLG)* of a lifted problem $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{A}, s_0, G \rangle$ is the graph $G = \langle V, E, \mathbf{X} \rangle$ with
- $V = \mathcal{P} \cup \mathcal{O} \cup N(\mathcal{A}) \cup N(s_0 \cup G)$ with

$$N(s_0 \cup G) = \bigcup_{p = P(o_1, \ldots, o_{n_P}) \in s_0 \cup G} \{p, p_1, \ldots, p_{n_P}\}$$
$$N(\mathcal{A}) = \bigcup_{a \in \mathcal{A}} \big(\{a\} \cup \{a_\delta \mid \delta \in \Delta(a)\} \cup$$
$$\bigcup_{f \in \{\mathrm{pre,add,del}\}} \bigcup_{p = P(\delta_1, \ldots, \delta_{n_P}) \in f(a)}$$
$$\{p_{a,f}, p_{a,f,1}, \ldots, p_{a,f,n_P}\}\big)$$

$N(s_0 \cup G)$ contains nodes corresponding to the state and goal, and ground arguments layer as in Fig. 3(a), while $N(\mathcal{A})$ provides all the nodes corresponding to the action schema, schema argument, predicate argument and schema predicate layers in Fig. 3(b).

- $E = E_\nu \cup E_\gamma \cup \bigcup_{f \in \{\mathrm{pre,add,del}\}} E_f$ where

$$E_\nu = \big\{ \langle o, P \rangle_\nu \mid o \in \mathcal{O}, P \in \mathcal{P} \big\} \cup$$
$$\big\{ \langle a, a_\delta \rangle_\nu \mid \delta \in \Delta(a), a \in \mathcal{A} \big\}$$
$$E_\gamma = \bigcup_{p = P(o_1, \ldots, o_{n_P}) \in s_0 \cup G} \big(\big\{ \langle p, p_i \rangle_\gamma \mid i \in [n_P] \big\} \cup$$
$$\big\{ \langle p_i, o_i \rangle_\gamma \mid i \in [n_P] \big\} \cup \big\{ \langle p, P \rangle_\gamma \big\} \big)$$
$$E_f = \bigcup_{p = P() \in f(a)} \big\{ \langle P, p_{a,f} \rangle_f, \langle p_{a,f}, a \rangle_f \big\} \cup$$
$$\bigcup_{p = P(\delta_1, \ldots, \delta_{n_P}) \in f(a), n_P \geq 1} \big(\big\{ \langle P, p_{a,f} \rangle_f \big\} \cup$$
$$\big\{ \langle p_{a,f}, p_{a,f,i} \rangle_f, \langle p_{a,f,i}, a_{\delta_i} \rangle_f \mid i \in [n_P] \big\} \big)$$

for $f \in \{\mathrm{pre, add, del}\}$. E_ν connects objects to predicates and schemas to their arguments, as indicated by gray edges in Fig. 3, E_γ connects nodes in $\mathcal{P}$, $\mathcal{O}$, and $N(s_0 \cup G)$ in order to represent propositions in the goal and true in the state as instantiated predicates with objects in the correct arguments, and $\bigcup_{f \in \{\mathrm{pre,add,del}\}} E_f$ connects nodes in $\mathcal{P}$ and $N(\mathcal{A})$ to encode the semantics of action schema in the graph.

- $\mathbf{X} : V \to \mathbb{R}^{5+T}$ defined by

$$u \mapsto [u \in \mathcal{P}; u \in \mathcal{O}; u \in \mathcal{A}; u \in s_0; u \in G] \,\|\, \overline{\mathrm{IF}}(u)$$

where $\|$ denotes vector concatenation, $\overline{\mathrm{IF}}(u) = \mathrm{IF}(i)$ for u of the form p_i or $p_{a,f,i}$ with $f \in \{\mathrm{pre, add, del}\}$ and $\overline{\mathrm{IF}}(u) = \vec{0}$ otherwise, and $\mathrm{IF} : \mathbb{N} \to \mathbb{R}^T$ is defined by a fixed randomly chosen injective map from $\mathbb{N}$ to the sphere $\big\{ x \in \mathbb{R}^T \mid \|x\| = 1 \big\}$.

We use the index function IF to encode the index at which an object instantiates the argument of a predicate or action schemas. This usage of IF lets us address STRIPS-HGN's limitation of having a fixed maximum number of parameters for predicates and action schemas that is chosen before training. Specifically, IF provides a numerically stable representation of an unbounded range of indexes that is agnostic to the maximum arity of the problem. Moreover, IF improves generalisation to large and unseen indexes as they are mapped to normalised vectors already seen by the model.

IF draws inspiration from positional encoding functions used in Transformers (Vaswani et al. 2017) and GNNs (Li et al. 2020; Dwivedi et al. 2022; Wang et al. 2022) for encoding positions as vectors. However, while Transformers and GNNs use positional encodings to encode all input tokens and graph nodes respectively, we use IF features only in the subset of nodes required to encode argument indexes in the lifted planning task. Furthermore, positional encodings aim to correlate positions and their corresponding features, such that objects close to each other are given similar features. In contrast, IF generates features for each index that are independent of one another. Hence, IF features are randomly generated i.i.d. for each index and also *a priori* such that they can be used for domain-independent learning.
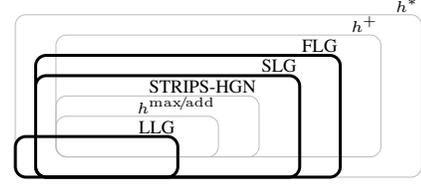

Figure 4: Expressiveness hierarchy of MPNNs on graph representations with respect to STRIPS-HGN and the heuristics h^{max}, h^{add}, h^+ and h^*. Bold outlines represent new graphs.

4 Expressiveness

We have defined three novel graph representations of planning tasks for the goal of learning domain-independent heuristics. In this section we will categorise the expressiveness of such graph representations when used with MPNNs by identifying which domain-independent heuristics they are able to learn. Our study also includes characterising the expressiveness of STRIPS-HGN (Shen, Trevizan, and Thiébaux 2020), the previous work on learning domain-independent heuristics. Fig. 4 summarises the main theorems of this section via an expressiveness hierarchy. Proofs of theorems are provided in a technical report (Chen, Thiébaux, and Trevizan 2023b).

We begin with a lower bound on what MPNNs can learn by showing that they can theoretically learn to imitate algorithms for computing h^{max} and h^{add} on our grounded graphs with the use of the approximation theorem for neural networks (Cybenko 1989; Hornik, Stinchcombe, and White 1989). We note that the theorem does not say anything about generalisability.

Theorem 4.1 (MPNNs can learn h^{add} and h^{max} on grounded graphs). *Let $L, B \in \mathbb{N}$, $\mathcal{G} \in \{SLG, FLG\}$, $\varepsilon > 0$ and $h \in \{h^{\mathrm{add}}, h^{\mathrm{max}}\}$. Then there exists a set of parameters Θ for an MPNN $\mathcal{F}_\Theta$ such that for all planning tasks Π, if naive dynamic programming for computing h converges within L iterations for Π, and $h(\Pi) \leq B$, then we have $|h(\Pi) - \mathcal{F}_\Theta(\mathcal{G}(\Pi))| < \varepsilon$.*

MPNNs acting on SLG and FLG are strictly more expressive than STRIPS-HGN. The idea of the theorem is that STRIPS-HGN discards delete effects which prohibits it from learning h^*. Furthermore, it is possible to imitate STRIPS-HGN with minor assumptions on MPNN architectures acting on either of our grounded graphs.

Theorem 4.2 (MPNNs on grounded graphs are strictly more expressive than STRIPS-HGN). *Let $\mathcal{G} \in \{SLG, FLG\}$. Given any set of parameters Θ for a STRIPS-HGN model $\mathcal{S}_\Theta$, there is a set of parameters Φ for an MPNN $\mathcal{F}_\Phi$ such that for any pair of planning tasks Π_1 and Π_2 where $\mathcal{S}_\Theta(\Pi_1) \neq \mathcal{S}_\Theta(\Pi_2)$, we have $\mathcal{F}_\Phi(\mathcal{G}(\Pi_1)) \neq \mathcal{F}_\Phi(\mathcal{G}(\Pi_2))$. Furthermore, there exists a pair of planning problems Π_1 and Π_2 such that there exists Φ where $\mathcal{F}_\Phi(\mathcal{G}(\Pi_1)) \neq \mathcal{F}_\Phi(\mathcal{G}(\Pi_2))$ but $\mathcal{S}_\Theta(\Pi_1) = \mathcal{S}_\Theta(\Pi_2)$ for all Θ.*

The first of our negative results is that MPNNs cannot learn h^{add} or h^{max} on the lifted LLG graph. This is due to the graph being too condensed in the lifted version so that

MPNNs cannot extract certain information for computing these heuristics. The proof idea is to find a pair of planning tasks which appear symmetric to MPNNs in the LLG representation but have different $h^{\max}$ and h^{add} values.

Theorem 4.3 (MPNNs cannot learn h^{add} and $h^{\max}$ on lifted graphs). *Let $h \in \{h^{\mathrm{add}}, h^{\max}\}$. There exists a pair of planning tasks Π_1 and Π_2 with $h(\Pi_1) \neq h(\Pi_2)$ such that for any set of parameters Θ for an MPNN we have $\mathcal{F}_{\Theta}(LLG(\Pi_1)) = \mathcal{F}_{\Theta}(LLG(\Pi_2))$.*

Next, we have that MPNNs cannot learn h^+ and thus h^* on any of our graphs. This result is not unexpected given that the expressiveness of MPNNs is bounded by the graph isomorphism class GI whose hardness is known to be in the low hierarchy of NP, unlike h^+ which is NP-complete. Similarly to the previous theorem, the proof follows the technique of finding a pair of planning tasks with different h^+ values that are indistinguishable by MPNNs on any of our graphs.

Theorem 4.4 (MPNNs cannot learn h^+ or h^* with our graphs). *Let $h \in \{h^+, h^*\}$ and $\mathcal{G} \in \{SLG, FLG, LLG\}$. There exists a pair of planning tasks Π_1 and Π_2 with $h(\Pi_1) \neq h(\Pi_2)$ such that for any set of parameters Θ for an MPNN we have $\mathcal{F}_{\Theta}(\mathcal{G}(\Pi_1)) = \mathcal{F}_{\Theta}(\mathcal{G}(\Pi_2))$.*

One may ask if it is possible to learn any approximation of h^+ or h^* on all planning problems. Unfortunately, it is not possible to learn either absolute or relative approximations. This is formalised in the following theorem, where the proof consists of a class of planning task pairs generalising the previous example.

Theorem 4.5 (MPNNs cannot learn any approximation of h^+ or h^*). *Let $h \in \{h^+, h^*\}$, $\mathcal{G} \in \{SLG, FLG, LLG\}$ and $c > 0$. There exists a pair of planning tasks Π_1 and Π_2 with $h(\Pi_1) \neq h(\Pi_2)$ such that for any set of parameters Θ for an MPNN we do not have $|\mathcal{F}_{\Theta}(\mathcal{G}(\Pi_1)) - h(\Pi_1)| \leq c \wedge |\mathcal{F}_{\Theta}(\mathcal{G}(\Pi_2)) - h(\Pi_2)| \leq c$. Also, for any set of parameters we do not have $|1 - \mathcal{F}_{\Theta}(\mathcal{G}(\Pi_1))/h(\Pi_1)| \leq c \wedge |1 - \mathcal{F}_{\Theta}(\mathcal{G}(\Pi_2))/h(\Pi_2)| \leq c$.*

We note that our theorems provide extreme upper and lower bounds on what MPNNs can learn with some of our graphs. Although in the general case we have the negative result that we cannot learn h^+ or h^*, it is still possible to learn h^* on subclasses of planning tasks. For example, Ståhlberg, Bonet, and Geffner (2022a) theoretically analyse which domains their MPNN architecture can learn h^* for, by using the well known result concerning the connection between MPNNs and 2-variable counting logics (Cai, Fürer, and Immerman 1992; Barceló et al. 2020). Furthermore, the results in this study concern MPNNs but there exist more expressive graph representation learning methods. For example, under additional assumptions, the universal approximation theorem with random node initialisation by Abboud et al. (2021) can be applied to learn h^*. Lastly, note that neither our results nor previous works examine the generalisability of learned heuristic functions.

5 Experiments

We provide experiments in order to evaluate the effectiveness of our graph representations for use with both domain-

domain	training		validation		testing	
blocks	$b \in [3, 10]$	40	$b \in [11]$	3	$b \in [15, 100]$	90
ferry	$l, c \in [2, 10]$	125	$l, c \in [11]$	3	$l, c \in [15, 100]$	90
gripper	$b \in [1, 10]$	10	$b \in [11]$	1	$b \in [15, 100]$	18
n-puzzle	$n \in [2, 4]$	100	$n \in [5]$	3	$n \in [5, 9]$	50
sokoban	$n \in [5, 7]$	60	$n \in [8]$	3	$n \in [8, 12]$	90
spanner	$s, n \in [2, 10]$	75	$s, n \in [11]$	3	$s, n \in [15, 100]$	90
visitall	$n \in [3, 10]$	24	$n \in [11]$	3	$n \in [15, 100]$	90
visitsome	$n \in [3, 10]$	24	$n \in [11]$	3	$n \in [15, 100]$	90

Table 1: Problem training/validation/testing splits with sizes and number of tasks per domain.

dependent and domain-independent learning of heuristic functions, as well as to answer some open questions left behind in our theoretical discussion. In order to do so, we introduce our GOOSE planner which combines graph generation, graph representation learning and domain-independent planning. Code is available at (Chen, Thiébaux, and Trevizan 2023a).

GOOSE The **G**raphs **O**ptimised f**O**r **S**earch **E**valuation (GOOSE) architecture represents planning tasks with one of the three graphs described previously (SLG, FLG, LLG) and uses MPNNs to learn heuristic functions for search. During heuristic evaluation, GOOSE treats each state s of a planning task $\langle S, A, s_0, G \rangle$ as a new planning subtask $\langle S, A, s, G \rangle$ which is then transformed into the chosen graph and fed into an MPNN. We use a RGCN (Schlichtkrull et al. 2018) message passing step: $\mathbf{h}_u^{(t+1)} = \sigma\Big(\mathbf{W}_0^{(t)}\mathbf{h}_u^{(t)} + \sum_{\iota \in \mathcal{R}} \bigoplus_{\langle u,v \rangle_\iota \in \mathcal{N}_\iota(u)} \mathbf{W}_\iota^{(t)}\mathbf{h}_v^{(t)}\Big)$, where $\oplus$ denotes the aggregator over neighbours under different edge labels. GOOSE uses the eager GBFS component of Fast Downward (Helmert 2006) for search, but calls the trained models for heuristic evaluation and parallelises the evaluation of successor states on GPUs for each opened node. GOOSE only constructs graphs from static propositions computed by Fast Downward.

Setup For domain-dependent heuristic learning, we train 5 models for each domain on optimal plans with problems specified in Tab. 1. Each plan of length h^* contributes states $s_0, s_1, \ldots, s_g$ with corresponding labels $h^*, h^* - 1, \ldots, 0$. For domain-independent heuristic learning, we consider the problems and domains of the 1998 to 2018 IPC dataset, excluding the domains in Tab. 1. We train 5 models using optimal plans generated by scorpion (Seipp, Keller, and Helmert 2020) with a 30min cutoff time and unit costs. In both settings, a model is trained with the Adam optimiser (Kingma and Ba 2015), batch size 16, initial learning rate of 0.001 and MSE loss. We schedule our learning rate by extracting 25% of the training data and reducing the learning rate by a factor of 10 if the loss on this data subset did not decrease in the last 10 epochs. Training is stopped when the learning rate becomes less than 10^{-5} on this subset, which often occurs within a few minutes. Following a similar method to Ferber, Helmert, and Hoffmann (2020), we select the best model for both settings by choosing the model which solves the most
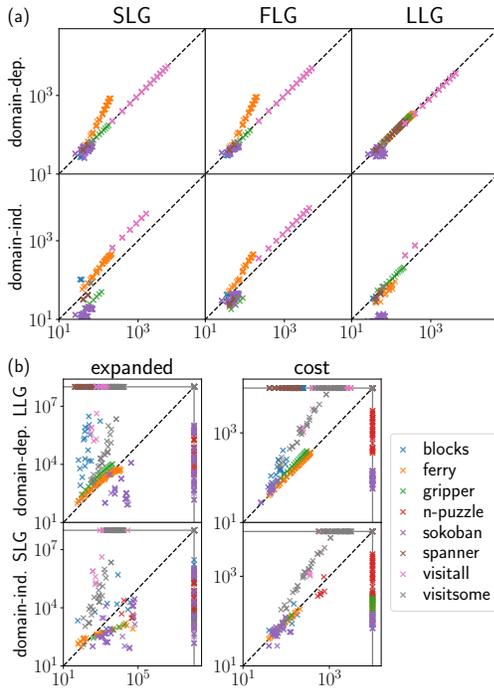
Figure 5: (a) GOOSE learned heuristics (y-axis) vs. h^* (x-axis). No n-puzzle problem could have h^* computed. (b) h^{FF} (y-axis) vs. GOOSE (x-axis) on number of expanded nodes (left) and plan cost (right). Points on the bottom right triangles favour h^{FF} and on the top left triangles favour GOOSE. Problems unsolved by a configuration get value set to the maximum of the plot's axis.

problems in the validation set (Tab. 1). We break ties with the sum of number of expanded nodes, and the training loss. We choose a hidden dimension of 64 with 8 message passing layers and the mean aggregator. GOOSE is run with a single NVIDIA GeForce RTX 3090 GPU.

Baselines We evaluate against blind search, Fast Downward's implementation of eager GBFS with h^{FF}, and domain-dependent STRIPS-HGN. STRIPS-HGN was trained using the parameters described in the original paper but with the same dataset as GOOSE and is called from Fast Downward's eager GBFS for heuristic evaluation. The other baselines are run on CPUs only. All GOOSE configurations and baselines are run with a 600 second timeout and 8GB main memory. We note that we also evaluated Powerlifted's (Corrêa et al. 2020) implementation of lifted eager GBFS with both h^{FF} and its extension with GOOSE but they do not offer any advantage over Fast Downward on the chosen problems.

Results

We recall from our theoretical study that it is not possible to learn any approximation of h^* over all planning tasks, although it may still be possible for certain planning subclasses or domains. Furthermore, we can learn domain-dependent h^* heuristics for certain domains simi-

larly to (Ståhlberg, Bonet, and Geffner 2022a). Thus, we propose and answer the following question empirically.

How close to the optimal heuristic are our learned heurstics? To answer this question, we report the heuristic of the initial state $h(s_0)$ computed by our models on all planning problems for which we can compute the optimal heuristic either by a hard coded solver or an optimal planner in Fig. 5(a). In the domain-dependent training setting, GOOSE with LLG provides the best predictions over most domains except for Sokoban. GOOSE achieves close to perfect heuristic estimates even as problems scale in size. This can be explained by LLG encoding predicate information which the other graph representations do not have access to and for some domains, it suffices to count the predicates of true propositions to compute h^*. Meanwhile, MPNNs are not expressive enough to decode predicate information from the grounded graphs. In the domain-independent training setting, GOOSE heuristics tend to overestimate h^* on VisitAll, whose perfect heuristic can often be computed by counting unreached goals, and underestimate on Sokoban. The grounded graphs (SLG and FLG) underestimate on Gripper, Spanner and Sokoban. This suggests GOOSE heuristics are overfitting on the training set for the more expressive grounded graphs as it is not possible to compute h^*. In particular, FLG is more prone to overfitting since it encodes additional planning task structure in the conversion of STRIPS to FDR.

How useful are learned domain-dependent heuristics for search? To answer this question, we refer to Fig. 6 for a coverage table over all domains with various planners. We notice that GOOSE with LLG trained in a domain-dependent fashion provides the best coverage on Blocksworld and Spanner, and is tied with Fast Downward's eager GBFS with h^{FF} on Gripper. GOOSE with SLG performs best on the grid based path finding domains VisitAll and VisitSome. Meanwhile, h^{FF} performs best on the remaining Ferry, n-puzzle and Sokoban domains. However, all GOOSE configurations perform worse than blind search on Sokoban. Even though it expands fewer nodes than blind search, the runtime cost of computing GOOSE heuristics is too high. This may be due to the difficulty of the domain (PSPACE-complete) as size increases, given that in problems with similar size to the training set GOOSE outperforms the other baselines. STRIPS-HGN solves significantly fewer problems due to its slower evaluation on CPUs.

Fig. 5(b) shows the number of node expansions and returned plan quality of the best performing domain-dependent GOOSE graph, LLG, against h^{FF}. In domains where one planner solves significantly more problems than the other, it also has fewer node expansions. We also note that GOOSE generally has higher plan quality than h^{FF} over all problems which both planners were able to solve.

How useful are learned domain-independent heuristics for search? We again refer to Fig. 6 for the coverage of GOOSE trained with domain-independent heuristics. With the exception of Sokoban, domain-independent GOOSE outperforms blind search which suggests that the

domain	baselines			domain-dep.			domain-ind.		
	blind	h^{FF}	HGN	SLG	FLG	LLG	SLG	FLG	LLG
blocks	-	19	-	-	6	**62**	9	8	6
ferry	-	**90**	-	32	33	88	28	22	2
gripper	1	**18**	5	9	6	**18**	5	3	9
n-puzzle	-	**36**	-	10	10	-	6	3	-
sokoban	74	**90**	10	31	29	34	45	40	15
spanner	-	-	-	-	-	60	-	-	-
visitall	-	6	25	46	**50**	44	16	41	-
visitsome	3	26	33	72	39	65	**73**	65	15

Figure 6: Coverage of planners and GOOSE over various domains. Cell intensities indicate the top 3 planners per row.

aggr.	L	domain-dep.			domain-ind.		
		SLG	FLG	LLG	SLG	FLG	LLG
mean	4	0.40	0.43	0.94	0.19	0.15	0.18
	8	**0.53**	0.40	**1.00**	**0.38**	**0.32**	0.33
	12	0.44	0.37	0.85	0.37	**0.32**	0.21
	16	0.31	0.18	0.75	0.36	**0.32**	0.12
max	4	0.46	**0.50**	0.89	0.33	0.29	0.30
	8	0.41	0.43	0.88	0.36	0.30	**0.52**
	12	0.36	0.43	0.80	0.12	0.24	0.39
	16	0.41	0.36	0.53	0.06	0.24	0.20

Figure 7: Total coverage normalised per domain of GOOSE over various parameters and training paradigms, and normalised again by the coverage of the best performing configuration. Higher scores are better and the maximum score is 1. The best scores per *column* are highlighted in bold.

learned domain-independent heuristics have some informativeness. This is supported by Fig. 5(a) which shows that in most domains domain-independent heuristics provide a mostly-linear approximation of h^*. Most notably, domain-independent grounded graphs still outperform h^{FF} on VisitAll and VisitSome, and domain-independent LLG is able to solve some Spanner problems.

The best performing domain-independent GOOSE graph with 8 message passing layers and mean aggregator is the grounded graph SLG. It provides enough information to learn domain-independent heuristics with MPNNs in comparison to LLG, but also does not provide too much information to prevent overfitting in comparison to FLG which computes additional structure. Domain-independent GOOSE with SLG returns better quality plans, and expands fewer nodes than h^{FF} on VisitAll, VisitSome, and more than half the Blocksworld instances which both planners were able to solve. Domain-independent GOOSE also outperforms or ties with *domain-dependent* STRIPS-HGN across all domains except VisitAll. However, domain-independent GOOSE generally expands more nodes and returns lower quality plans than their domain-dependent trained variants with the same graph.

How important is finding the right graph neural network parameters? We report the normalised coverage of

GOOSE with hyperparameters $L \in \{4, 8, 12, 16\}$ layers and $\oplus \in \{\max, \text{mean}\}$ aggregator in Fig. 7. We omitted results with the sum aggregator as it yielded unstable training and poor predictions. Increasing the number of layers theoretically improves informativeness and accuracy of predictions but requires longer evaluation time and is more difficult to train. There is no single set of parameters that performs well over all graphs and training settings. Generally 4 or 8 layers result in similar coverages for domain-dependent training, and 8 or 12 layers for domain-independent training, while increasing the number of layers beyond this results in worse performance due to the aforementioned reasons. We note that the effectiveness of max and mean aggregations vary with the graph representation and domain as both aggregators lose information in different ways. However, in the domain-dependent setting, LLG with the mean aggregator generally outperforms the max aggregator given that the model can recover the information lost during normalisation through the grouping of edge labels and node types.

How long do GOOSE evaluations take? GOOSE on a single core CPU takes 0.2-0.9s to perform a full GNN evaluation on grounded graphs, and 0.2-0.3s on lifted graphs. With *optimal* GPU usage, grounded graphs take 0.1-5ms per state evaluation and lifted graphs take 0.07-0.7ms. Optimal usage is achieved when the batch size is greater than 32 for lifted graphs and 4 for grounded graphs. In our experiments, the grounded graphs were able to optimally use the GPU while the lifted graphs were not, resulting in a higher average state evaluation time. We note that evaluating states of successor nodes further in the queue to increase the batch size is also not optimal as the expanded states may never be evaluated in sequential GBFS. Evaluation of heuristics on GPUs is almost always faster than on CPU due to the parallel execution of GNN matrix and scatter operations.

6 Conclusion

We have constructed various novel graph representations of planning problems for the task of learning domain-independent heuristics. In particular we provide the first domain-independent graph representation of lifted planning. All our new models are also complemented by a theoretical analysis of their expressive power in relation to domain-independent heuristics and the previous work on learning domain-independent heuristics, STRIPS-HGN. We also construct the GOOSE planner using heuristic search with heuristics learned from our new graph representations. GOOSE has also been optimised for runtime with the use of GPU batch evaluation and is able to solve significantly larger problems than those seen in the training set, vastly surpassing STRIPS-HGN learned heuristics, and outperforming the h^{FF} heuristic on several domains. It remains for future work to implement search algorithms used by stronger satisficing planners in GOOSE, and to optimise GPU utilisation when computing heuristics. Furthermore GOOSE can be extended to predict deadends alongside a heuristic for further pruning the search space. Lastly, we aim to improve the expressiveness of learned heuristics by leveraging stronger graph representation learning techniques.

Acknowledgements

References

Abboud, R.; Ceylan, İ. İ.; Grohe, M.; and Lukasiewicz, T. 2021. The Surprising Power of Graph Neural Networks with Random Node Initialization. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Barceló, P.; Kostylev, E. V.; Monet, M.; Pérez, J.; Reutter, J.; and Silva, J.-P. 2020. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations (ICLR)*.

Cai, J.-Y.; Fürer, M.; and Immerman, N. 1992. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4): 389–410.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. 2023a. Code for Learning Domain-Independent Heuristics for Grounded and Lifted Planning. https://doi.org/10.5281/zenodo.10404209.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. 2023b. Learning Domain-Independent Heuristics for Grounded and Lifted Planning. arXiv:2312.11143.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation Using Query Optimization Techniques. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 80–89.

Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4): 303–314.

Dong, H.; Mao, J.; Lin, T.; Wang, C.; Li, L.; and Zhou, D. 2019. Neural Logic Machines. In *International Conference on Learning Representations (ICLR)*.

Dwivedi, V. P.; Luu, A. T.; Laurent, T.; Bengio, Y.; and Bresson, X. 2022. Graph neural networks with learnable structural and positional representations. In *International Conference on Machine Learning (ICML)*.

Ferber, P.; Helmert, M.; and Hoffmann, J. 2020. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In Giacomo, G. D.; Catalá, A.; Dilkina, B.; Milano, M.; Barro, S.; Bugarín, A.; and Lang, J., eds., *European Conference on Artificial Intelligence (ECAI)*, volume 325, 2346–2353.

Garg, S.; Bajpai, A.; and Mausam. 2020. Symbolic Network: Generalized Neural Policies for Relational MDPs. In *International Conference on Machine Learning (ICML)*, volume 119, 3397–3407.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gehring, C.; Asai, M.; Chitnis, R.; Silver, T.; Kaelbling, L. P.; Sohrabi, S.; and Katz, M. 2022. Reinforcement Learning for Classical Planning: Viewing Heuristics as Dense Reward Generators. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 588–596.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *International Conference on Machine Learning (ICML)*, volume 70, 1263–1272.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.*, 26: 191–246.

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

Hornik, K.; Stinchcombe, M.; and White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5): 359–366.

Karia, R.; and Srivastava, S. 2021. Learning Generalized Relational Heuristic Networks for Model-Agnostic Planning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 8064–8073.

Katz, M.; Sohrabi, S.; Samulowitz, H.; and Sievers, S. 2018. Delfi: Online planner selection for cost-optimal planning. *International Planning Competition*, 57–64.

Kingma, D. P.; and Ba, J. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Lauer, P.; Torralba, A.; Fiser, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Li, P.; Wang, Y.; Wang, H.; and Leskovec, J. 2020. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 33: 4465–4478.

Ma, T.; Ferber, P.; Huo, S.; Chen, J.; and Katz, M. 2020. Online Planner Selection with Graph Neural Networks and Adaptive Scheduling. In *AAAI Conference on Artificial Intelligence (AAAI)*, 5077–5084.

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting Problem Symmetries in State-Based Planners. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Schlichtkrull, M. S.; Kipf, T. N.; Bloem, P.; van den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling Relational Data with Graph Convolutional Networks. In *Extended Semantic Web Conference (ESWC)*, Lecture Notes in Computer Science, 593–607.

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

Sharma, V.; Arora, D.; Geißer, F.; Mausam; and Singla, P. 2022. SymNet 2.0: Effectively handling Non-Fluents and Actions in Generalized Neural Policies for RDDL Relational

MDPs. In Cussens, J.; and Zhang, K., eds., *Uncertainty in Artificial Intelligence, (UAI)*, volume 180, 1771–1781.

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

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and Symmetries in Classical Planning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 3371–3377.

Sievers, S.; Röger, G.; Wehrle, M.; and Katz, M. 2019. Theoretical Foundations for Structural Symmetries of Lifted PDDL Tasks. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 29, 446–454.

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

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

Teichteil-Königsbuch, F.; Povéda, G.; Guillermo Gonzalez de Garibay Barba, T. L.; and Thiébaux, S. 2023. Fast and Robust Resource-Constrained Scheduling with Graph Neural Networks. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 5998–6008.

Wang, H.; Yin, H.; Zhang, M.; and Li, P. 2022. Equivariant and Stable Positional Encoding for More Powerful Graph Neural Networks. In *International Conference on Learning Representations (ICLR)*.