GOOSE: Learning Domain-Independent Heuristics

Dillon Z. Chen[1,2] **Sylvie Thiébaux**[1,2] **Felipe Trevizan**[2]
[1]LAAS-CNRS [2]Australian National University
{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 et al. [2020], Garg et al. [2020], Karia and Srivastava [2021], Ståhlberg et al. [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 et al. [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 et al. [2020] is the only existing model designed to learn domain-independent heuristic functions from scratch. The models in Ståhlberg et al. [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, so that its 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 informativeness 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 et al. [2020], Karia and Srivastava [2021] and Ståhlberg et al. [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

2.1 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 $a : S \to S \cup \bot$ mapping a state s in which the action is applicable to its successor $a(s)$, and states in which it is not applicable to $\bot$. The cost of the action is $c(a) \in \mathbb{N}$. A solution or a *plan* in this model is a sequence of actions $\pi = a_1, \dots, a_n$ such that $s_i = a_i(s_{i-1}) \neq \bot$ for all $i \in \{1, \dots, n\}$ 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 $c(a) \in \mathbb{N}$. 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 $x_1, \dots, x_{n_P}$ 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 known as 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.

2.2 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) 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

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

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. In particular, our graph for lifted tasks avoids grounding and allows us to learn a lifted heuristic.

3.1 Grounded Graphs

A graph representation for grounded STRIPS problems already exists, namely the STRIPS problem description graph (PDG) [Pochter et al., 2011]. 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.

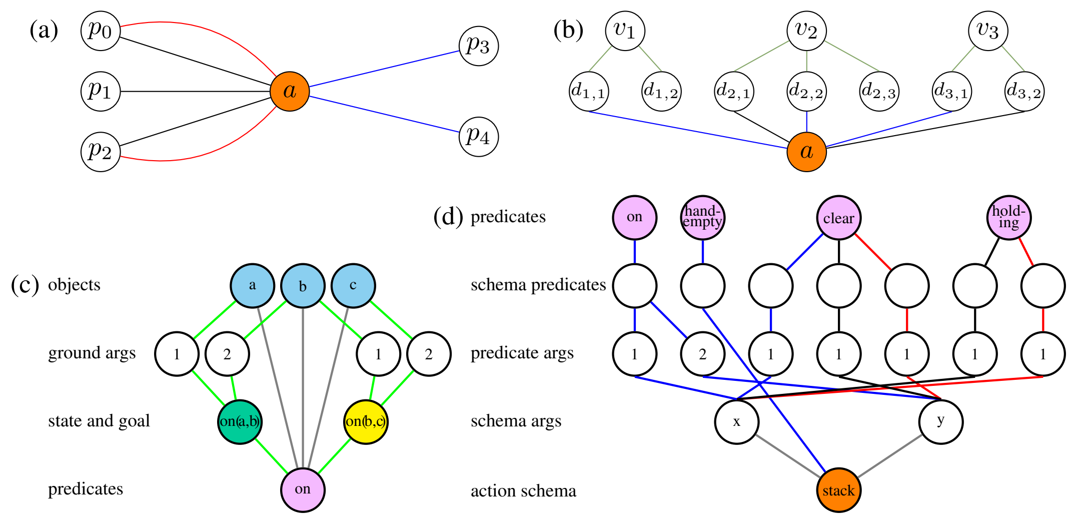


Figure 1: (a) SLG subgraph of an action a with $\mathrm{pre}(a) = \{p_0, p_1, p_2\}$, $\mathrm{add}(a) = \{p_3, p_4\}$ and $\mathrm{del}(a) = \{p_0, p_2\}$, indicated by black, blue and red edges respectively. (b) FLG subgraph of an action a with $\mathrm{pre}(a) = \{\langle v_2, d_{2,1}\rangle, \langle v_3, d_{3,2}\rangle\}$ 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. (c-d): LLG instance subgraph and schema subgraphs, with layer descriptions of a Blocksworld instance.

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

- $V = A \cup P$,
- $E = E_{\text{pre}} \cup E_{\text{add}} \cup E_{\text{del}}$ where for $\iota \in \{\text{pre}, \text{add}, \text{del}\}$,
$$E_\iota = \{\langle a, p\rangle_\iota \mid p \in \iota(a), a \in A\},$$
- $\mathbf{X} : V \to \mathbb{R}^3$ 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. Three dimensional node features are sufficient for encoding whether a node corresponds 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(a) illustrates an example SLG subgraph.

The FDR problem description graph (PDG) [Shleyfman et al., 2015] 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. 1(b) 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

- $V = \mathcal{V} \cup \bigcup_{v\in\mathcal{V}} D_v \cup A$,
- $E = E_{\text{var:val}} \cup E_{\text{pre}} \cup E_{\text{eff}}$ with
$$\begin{aligned} E_{\text{var:val}} &= \textstyle\bigcup_{v\in\mathcal{V}}\{\langle v, d\rangle_{\text{var:val}} \mid d \in D_v\} \\ E_{\text{pre}} &= \textstyle\bigcup_{a\in A}\{\langle d, a\rangle_{\text{pre}} \mid (v, d) \in \text{pre}(a)\} \\ E_{\text{eff}} &= \textstyle\bigcup_{a\in A}\{\langle d, a\rangle_{\text{eff}} \mid (v, d) \in \text{eff}(a)\}, \end{aligned}$$
- $\mathbf{X} : V \to \mathbb{R}^5$ defined by
$$u \mapsto [u \in \mathcal{V}; u \in A; \text{val}(u); \text{true}(u); \text{goal}(u)]$$
where $\text{val}(u) = \exists v \in \mathcal{V}$, $u \in D_v$, $\text{true}(u) = \exists v \in \mathcal{V}$, $\langle v, u\rangle \in s_0$ and $\text{goal}(u) = \exists v \in \mathcal{V}$, $\langle v, u\rangle \in s_\star$.

3.2 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 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. Both 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 expressivity when used with MPNNs.

To overcome the issues with ASGs, we introduce a new graph representation for lifted planning tasks designed to be used with MPNNs. Our new representation in Def. 3.3 consists of two main components: the schema subgraph which encodes the action schemas of the domain, and the instance subgraph which encodes the instance specific information containing the current state and the goal condition. In the following definition, will assume that there are no partially instantiated action schema as is generally the case for most PDDL files.

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

$$N(s_0 \cup G) = \bigcup_{p=P(o_1,\ldots,o_{n_P}) \in s_0 \cup G} \{p, p_1, \ldots, p_{n_P}\}$$

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

$N(s_0 \cup G)$ contains nodes corresponding to the state and goal, and ground arguments layer as in Fig. 1(c), while $N(\mathcal{A})$ provides all the nodes corresponding to the action schema, schema argument, predicate argument and schema predicate layers in Fig. 1(d).

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

$$E_\nu = \{\langle o, P \rangle_\nu \mid o \in \mathcal{O}, P \in \mathcal{P}\} \ \cup \ \{\langle a, a_\delta \rangle_\nu \mid \delta \in \Delta(a), a \in \mathcal{A}\}$$

$$E_\gamma = \bigcup_{p=P(o_1,\ldots,o_{n_P}) \in s_0 \cup G} \big(\{\langle p, p_i \rangle_\gamma \,|\, i \in [n_P]\} \ \cup \ \{\langle p_i, o_i \rangle_\gamma \,|\, i \in [n_P]\} \cup \{\langle p, P \rangle_\gamma\} \big)$$

$$E_f = \bigcup_{p=P() \in f(a)} \{\langle P, p_{a,f} \rangle_f, \langle p_{a,f}, a \rangle_f\} \cup$$
$$\bigcup_{p=P(\delta_1,\ldots,\delta_{n_P}) \in f(a), n_P \geq 1} \big(\{\langle P, p_{a,f} \rangle_f\} \ \cup \ \{\langle p_{a,f}, p_{a,f,i} \rangle_f, \langle p_{a,f,i}, a_{\delta_i} \rangle_f \,|\, i \in [n_P]\} \big)$$

for $f \in \{\text{pre}, \text{add}, \text{del}\}$. E_ν connects objects to predicates and schemas to their arguments, as indicated by gray edges in Fig. 1, 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 $\bigcup_{f \in \{\text{pre},\text{add},\text{del}\}} E_f$ provides edges connecting 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 $u \mapsto [u \in \mathcal{P}; u \in \mathcal{O}; u \in \mathcal{A}; u \in s_0; u \in G] \,\|\, \overline{\text{IF}}(u)$ where $\|$ denotes vector concatenation, $\overline{\text{IF}}(u) = \text{IF}(i)$ for u of the form p_i or $p_{a,f,i}$ with $f \in \{\text{pre}, \text{add}, \text{del}\}$ and $\overline{\text{IF}}(u) = \vec{0}$ otherwise, and IF : $\mathbb{N} \to \mathbb{R}^T$ is defined by a fixed randomly chosen injective map from $\mathbb{N}$ to the sphere $\{x \in \mathbb{R}^T \mid \|x\| = 1\}$.

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.

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 et al. [2020], the previous work on learning domain-independent heuristics. Fig. 2 summarises the main theorems of this section via an expressiveness hierarchy. Proofs of theorems are attached in the appendix.

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 et al. [1989].

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 $h \in \{h^{\text{add}}, h^{\max}\}$. 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 $h(s_0) \leq B$, then we have $|h(s_0) - \mathcal{F}_\Theta(\mathcal{G}(\Pi))| < \varepsilon$.*

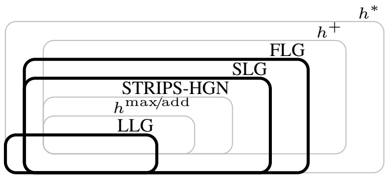


Figure 2: Expressiveness hierarchy of MPNNs on graph representations with respect to STRIPS-HGN and the heuristics $h^{\max}$, h^{add}, h^+ and h^*. Bold outlines represent new graphs.

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 exists 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 $h \in \{h^{\text{add}}, h^{\max}\}$. 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. We formalise this 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 $\bigwedge_{i=1,2} |\mathcal{F}_\Theta(\mathcal{G}(\Pi_i)) - h(\Pi_i)| \leq c$. Also, for any set of parameters we do not have $\bigwedge_{i=1,2} |1 - \mathcal{F}_\Theta(\mathcal{G}(\Pi_i))/h(\Pi_i)| \leq c$.*

One may also ask about the expressiveness of learning a policy. A policy can be learned in one of several ways. A policy can be induced from a learned heuristic where given a state s we take the action a whose successor s' has the lowest heuristic value over all successors from s. To learn a policy directly on grounded graphs, we can take inspiration from ASNets Toyer et al. [2020] and predict confidence values in the range from 0 to 1 on grounded action nodes. To learn a policy directly on lifted graphs, one may take inspiration from the architecture by Karia and Srivastava [2021] which predicts the schema and the corresponding arguments of an action.

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 et al. [2022a] identify domains for which they can formulate the optimal policy with 2-variable counting logics Cai et al. [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, we 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-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.

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 modified RGCN Schlichtkrull et al. [2018] message passing step where we replace the mean with max aggregator over neighbours under different edge labels:

$$\mathbf{h}_u^{(t+1)} = \sigma\Big(\mathbf{W}_0^{(t)}\mathbf{h}_u^{(t)} + \sum_{\iota\in\mathcal{R}} \max_{\langle u,v\rangle_\iota \in \mathcal{N}_\iota(u)} \mathbf{W}_\iota^{(t)}\mathbf{h}_v^{(t)}\Big).$$

GOOSE uses the eager GBFS component of Fast Downward Helmert [2006] for search which calls the trained models for heuristic evaluation and the evaluation of successor states is parallelised on GPUs for each opened node. The learning and evaluation module is built on pytorch-geometric Fey and Lenssen [2019], while the search component is implemented in C++.

5.1 Setup and Baselines

For domain-dependent heuristic learning, we train 5 models for each domain on optimal plans with problems specified in Fig. 3. Each plan of length h^* contributes states $s_0, s_1, \dots, s_g$ with corresponding labels $h^*, h^* - 1, \dots, 0$. For domain-independent heuristic learning, we train 5 models on optimal plans in the 1998 to 2018 IPC dataset with unit costs. We remove any training data associated with the domains in the test set in Fig. 3. In both settings only optimal plans from training problems for which scorpion Seipp et al. [2020] solves within 30 minutes are kept. 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. In both settings, following a similar method to Ferber et al. [2020] we then select the best model using the validation set in Fig. 3 by choosing the model which solves the most problems, and breaking ties with the sum of number of expanded nodes, and the training loss. For all models, we choose hidden dimensionality 64 with 4 message passing layers. GOOSE is run with a single NVIDIA GeForce RTX 3090 GPU.

domain	train		validate		test		best
blocks	$b \in [3, 10]$	*40*	$b \in [11]$	*3*	$b \in [15, 100]$	*90*	$b = 60^*$
ferry	$l, c \in [2, 10]$	*125*	$l, c \in [11]$	*3*	$l, c \in [15, 100]$	*90*	$l, c = 90$
gripper	$b \in [1, 10]$	*10*	$b \in [11]$	*1*	$b \in [15, 100]$	*18*	$b = 100$
n-puzzle	$n \in [2, 4]$	*100*	$n \in [5]$	*3*	$n \in [5, 9]$	*50*	$n = 5^*$
sokoban	$n \in [5, 7]$	*60*	$n \in [8]$	*3*	$n \in [8, 12]$	*90*	$n = 12^*$
spanner	$s, n \in [2, 10]$	*75*	$s, n \in [11]$	*3*	$s, n \in [15, 100]$	*90*	$s, n = 65$
visitall	$n \in [3, 10]$	*24*	$n \in [11]$	*3*	$n \in [15, 100]$	*90*	$n = 65$
visitsome	$n \in [3, 10]$	*24*	$n \in [11]$	*3*	$n \in [15, 100]$	*90*	$n = 90$

Figure 3: Problem splits with sizes and number of tasks per domain. Right most column indicates largest size problem solved with GOOSE. An asterisk ($*$) marks whether the problem size is not directly correlated with problem difficulty.

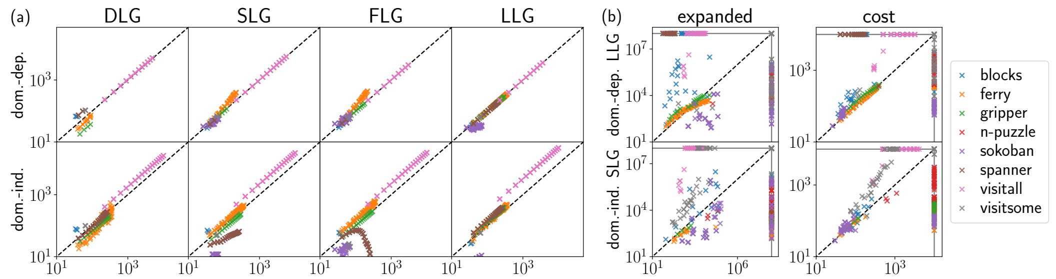


Figure 4: (a) GOOSE learned heuristics (y-axis) vs. h^* (x-axis). (b) h^{FF} (y-axis) vs. GOOSE (x-axis) on number of expanded nodes (left) and plan cost (right). Points on the bottom right triangles favour h^{FF} and on the top left triangles favour GOOSE. Problems unsolved by a configuration get value set to the maximum of the plot's axis.

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. We also consider an additional baseline, the *delete learning graph* (DLG), defined as the SLG graph without delete effect edges. DLG is an alternative to STRIPS-HGN implemented in GOOSE with similar expressivity (Thm. 4.2). The DLG configuration consists of both domain-dependent and domain-independent training and also employs GPUs. The other baselines are run on CPUs only. All GOOSE configurations and baselines are run with a 600 second timeout. 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. LAMA Richter and Westphal [2010] is able to solve almost all problems on every domain except Spanner which it solves none. To focus on the effectiveness of learned heuristics, we left out results for stronger satisficing planners which use techniques beyond heuristic search.

5.2 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 similarly to Ståhlberg et al. [2022a]. Thus, we propose and answer the following question empirically.

How close to the optimal heuristic are our learned heuristics? 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. 4(a). In the domain-dependent training setting, GOOSE with LLG provides the best predictions over most domains except for Sokoban. GOOSE achieves 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 which consider delete effects (SLG and FLG) underestimate on Spanner and in the case of FLG provides lower estimates on larger problems. This suggests GOOSE is overfitting on the training set on the more expressive grounded graphs since it is not possible to compute h^*, and is most true in the case of FLG which 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 Tab. 1 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. We note that several different configurations of MPNN parameters allow LLG to match h^{FF} on coverage in Ferry. However, all

Table 1: Left: coverage of planners and GOOSE over various domains. Cell intensities indicate rank of planner per domain. Right: 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 column-wise are highlighted in bold.

	baseline					domain-dep.			domain-ind.		
	blind	FD-h^{FF}	S-HGN	DLG-DD	DLG-DI	SLG	FLG	LLG	SLG	FLG	LLG
blocks (90)	-	19	-	15	6	10	11	**29**	14	14	16
ferry (90)	-	**90**	-	4	3	33	33	78	15	6	12
gripper (18)	1	**18**	-	3	6	5	9	**18**	4	5	9
n-puzzle (50)	-	**36**	-	4	1	10	10	1	3	3	-
sokoban (90)	74	**90**	-	12	37	52	56	34	48	42	39
spanner (90)	-	-	-	-	-	-	-	**55**	-	-	10
visitall (90)	-	6	-	41	15	**52**	35	39	44	24	38
visitsome (90)	3	26	-	73	25	**78**	23	3	37	14	-

		domain-dep.			domain-ind.		
aggr.	L	SLG	FLG	LLG	SLG	FLG	LLG
mean	4	0.70	**0.71**	0.90	0.39	0.24	0.40
	8	0.71	0.60	**1.00**	0.60	**0.57**	0.12
	12	0.62	0.52	0.93	**0.64**	0.56	0.19
	16	0.49	0.42	0.83	0.56	0.46	0.13
max	4	**0.78**	0.64	0.94	0.53	0.38	**0.46**
	8	0.67	0.53	0.72	0.55	0.41	0.37
	12	0.61	0.53	0.76	0.17	0.33	0.31
	16	0.66	0.49	0.77	0.13	0.33	0.34

GOOSE configurations perform worse than blind search on Sokoban. Even though it expands fewer nodes, 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 no problems due to its slow evaluation on CPUs. Our simplified STRIPS-HGN variant using GNNs, GOOSE with DLG, solves some problems but is not competitive with h^{FF} except on VisitAll and VisitSome. Lastly, we mention that LAMA Richter and Westphal [2010] solves almost all of the test problems except Spanner where it solves none. It remains as future work to extend GOOSE in the planning algorithm side beyond learning heuristics to match the performance of stronger satisficing planners.

Fig. 4(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 in several orders of magnitude. On Gripper, GOOSE has marginally fewer node expansions than h^{FF} until the largest problem with 100 balls, and similarly on Ferry except the size in which GOOSE begins to perform worse is smaller. 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 Tab. 1 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 learned domain-independent heuristics have some informativeness. This is again supported by Fig. 4(a) which shows that in most domains domain-independent heuristics provide an approximation of h^* which generally scales linearly with h^*. Most notably, domain-independent SLG outperforms h^{FF} on VisitAll and VisitSome, and domain-independent LLG is also able to solve some Spanner problems.

The best performing domain-independent GOOSE configuration 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. With the exception of n-puzzle, domain-independent GOOSE with SLG generally returns better quality plans than h^{FF}, and expands fewer nodes on VisitAll, VisitSome, and more than half the Blocksworld instances which both planners were able to solve. In terms of overall coverage, domain-independent SLG also outperforms *domain-dependent* DLG, the delete free version of SLG and optimised version of STRIPS-HGN. 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 number of message passing layers in $\{4, 8, 12, 16\}$ and both the mean and max aggregator in Tab. 1. We omitted results with the sum aggregator as it yields 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 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.

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, it is also possible to improve the expressivity of learned heuristics by leveraging stronger graph representation learning techniques.

Acknowledgements

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

References

Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

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

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

Augusto B. Corrêa, Florian Pommerening, Malte Helmert, and Guillem Francès. Lifted successor generation using query optimization techniques. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 80–89, 2020.

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

Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. In *International Conference on Learning Representations (ICLR)*, 2019.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *International Conference on Machine Learning (ICML)*, 2022.

Patrick Ferber, Malte Helmert, and Jörg Hoffmann. Neural network heuristics for classical planning: A study of hyperparameter space. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *European Conference on Artificial Intelligence (ECAI)*, volume 325, pages 2346–2353, 2020.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *CoRR*, abs/1903.02428, 2019.

Sankalp Garg, Aniket Bajpai, and Mausam. Symbolic network: Generalized neural policies for relational mdps. In *International Conference on Machine Learning (ICML)*, volume 119, pages 3397–3407, 2020.

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.

Clement Gehring, Masataro Asai, Rohan Chitnis, Tom Silver, Leslie Pack Kaelbling, Shirin Sohrabi, and Michael Katz. Reinforcement learning for classical planning: Viewing heuristics as dense reward generators. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 588–596, 2022.

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

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

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Rushang Karia and Siddharth Srivastava. Learning generalized relational heuristic networks for model-agnostic planning. In *AAAI*, pages 8064–8073, 2021.

Michael Katz, Shirin Sohrabi, Horst Samulowitz, and Silvan Sievers. Delfi: Online planner selection for cost-optimal planning. *International Planning Competition*, pages 57–64, 2018.

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

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

Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:4465–4478, 2020.

Tengfei Ma, Patrick Ferber, Siyu Huo, Jie Chen, and Michael Katz. Online planner selection with graph neural networks and adaptive scheduling. In *AAAI*, pages 5077–5084, 2020.

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

Silvia Richter and Matthias Westphal. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.*, 39:127–177, 2010.

Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *Extended Semantic Web Conference (ESWC)*, Lecture Notes in Computer Science, pages 593–607, 2018.

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

Vishal Sharma, Daman Arora, Florian Geißer, Mausam, and Parag Singla. Symnet 2.0: Effectively handling non-fluents and actions in generalized neural policies for RDDL relational mdps. In James Cussens and Kun Zhang, editors, *Uncertainty in Artificial Intelligence, (UAI)*, volume 180, pages 1771–1781, 2022.

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

Alexander Shleyfman, Michael Katz, Malte Helmert, Silvan Sievers, and Martin Wehrle. Heuristics and symmetries in classical planning. In *AAAI*, pages 3371–3377, 2015.

Silvan Sievers, Gabriele Röger, Martin Wehrle, and Michael Katz. Theoretical Foundations for Structural Symmetries of Lifted PDDL Tasks. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 29, pages 446–454, 2019.

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

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning generalized policies without supervision using gnns. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2022b.

Florent Teichteil-Königsbuch, Guillaume Povéda, Tim Luchterhand Guillermo Gonzalez de Garibay Barba, and Sylvie Thiébaux. Fast and robust resource-constrained scheduling with graph neural networks. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2023.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. Equivariant and stable positional encoding for more powerful graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2022.

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

A Proofs of Theorems

This appendix includes the proofs of all theorems. For ease of reference, theorem statements are also repeated here.

Theorem 4.1 (MPNNs can learn h^{add} and $h^{\max}$ on grounded graphs)**.** Let $L, B \in \mathbb{N}$, $\mathcal{G} \in \{\text{SLG}, \text{FLG}\}$, $\varepsilon > 0$ and $h \in \{h^{\text{add}}, h^{\max}\}$. 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 (Alg. 1) converges within L iterations for Π, and $h(s_0) \leq B$, then we have $|h(s_0) - \mathcal{F}_\Theta(\mathcal{G}(\Pi))| < \varepsilon$.

Proof. The main idea of the proof is that we can encode Alg. 1 for computing h into an MPNN using a correct choice of continuous bounded functions and aggregation operators and using the approximation theorem to find parameters in order to achieve the desired function. We will assume unitary cost actions and note that the below proof can be generalised to account for general cost actions. We first deal with the case where $h = h^{\max}$ and $\mathcal{G} = \text{DLG}$, where DLG is the graph SLG but without delete edges. The proof generalises to SLG as an MPNN can learn to ignore delete edges.

Let $x^{(u)} \in \mathbb{R}^3$ be the feature of node u. By definition of DLG as the graph SLG (Def. 3.1) with no delete edges, it is defined by $x_0^{(u)} = 1$ if u corresponds to a proposition node, else $x_0^{(u)} = 0$ when u

Algorithm 1: Naive dynamic programming for computing h^{add} and $h^{\max}$

```
Data: Propositional STRIPS planning task $\Pi = \langle P, A, s_0, G \rangle$, desired heuristic
      $h \in \{h^{\text{add}}, h^{\max}\}$
Result: $h(s) \in \mathbb{N}$
1  if $h = h^{\text{add}}$ then $\oplus \leftarrow \sum$;
2  else if $h = \max$ then $\oplus \leftarrow \max$;
3  $h^{(0)}[p] \leftarrow 0, \quad \forall p \in s_0$
4  $h^{(0)}[p] \leftarrow \infty, \quad \forall p \in P \setminus s_0$
5  for $i = 1, \ldots$ do
6      for $a \in A$ do
7          $h^{(i)}[a] \leftarrow \oplus_{p \in \text{pre}(a)} h^{(i-1)}[p]$
8      for $p \in P$ do
9          $h^{(i)}[p] \leftarrow \min\big(h^{(i-1)}[p], \min_{a \in A, p \in \text{add}(a)} h^{(i)}[a] + c(a)\big)$
10     if $h^{(i)} = h^{(i-1)}$ then
11         return $\oplus_{p \in g} h^{(i)}[p]$
```

corresponds to an action node a. Furthermore, $x_1^{(u)} = 1$ if u is a proposition in the initial state and $x_2^{(u)} = 1$ if u is a goal proposition. Note that it is possible that $x_1^{(u)} = x_2^{(u)} = 1$ when a proposition is both a goal condition and in the initial state. If not mentioned, we have that $x_i^{(u)} = 0$ everywhere else.

Then we will construct a MPNN with $2L + 2$ layers. For the first layer we have an embedding layer which ignores neighbourhood nodes with $\text{agg}^{(0)} = \vec{0}$ and $\varphi^{(0)}(\mathbf{h}_u, \mathbf{h}_N) = f_{\text{emb}}(\mathbf{h}_u)$. Let K be the finite set of possible node features in a DLG representation of a planning task. Then $f_{\text{emb}} : K \to \mathbb{R}^3$ is defined by

$$f_{\text{emb}}([0,0,0]^\top) = [0,0,0]^\top \tag{1}$$
$$f_{\text{emb}}([1,0,0]^\top) = [B,0,1]^\top \tag{2}$$
$$f_{\text{emb}}([1,0,1]^\top) = [B,1,1]^\top \tag{3}$$
$$f_{\text{emb}}([1,1,0]^\top) = [0,0,1]^\top \tag{4}$$
$$f_{\text{emb}}([1,1,1]^\top) = [0,1,1]^\top. \tag{5}$$

This first round of message passing updates corresponds to the initialisation step of the heuristic algorithm with B representing infinity values. We also note that after applying $\text{agg}^{(0)}$ and $\varphi^{(0)}$ and throughout the remaining forward pass of the MPNN, node embeddings will have the form $[x_0, x_1, x_2]$ which encode information about their corresponding proposition or action during the execution of the $h^{\max}$ algorithm where

- x_0 corresponds to the intermediate h values computed in the $h^{\max}$ algorithm,
- x_1 signifies whether the node corresponds to a goal node, and
- x_2 determines if the node is a proposition or action node.

The next $2L$ layers use the component wise max aggregation function $\text{agg} = \max$ and alternates between setting $\varphi^{(l)}(\mathbf{h}_u, \mathbf{h}_N) = f_a([\mathbf{h}_u \,\|\, \mathbf{h}_N])$ and $\varphi^{(l+1)}(\mathbf{h}_u, \mathbf{h}_N) = f_p([\mathbf{h}_u \,\|\, \mathbf{h}_N])$ where f_a :

$\mathbb{R}^6 \to \mathbb{R}^3$ and $f_p : \mathbb{R}^6 \to \mathbb{R}^3$ are defined by

$$f_a\left(\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ y_0 \\ y_1 \\ y_2 \end{bmatrix}\right) = \begin{bmatrix} x_0 x_2 - (1 - x_2) y_0 \\ x_1 x_2 \\ x_2^2 \end{bmatrix}, \tag{6}$$

$$f_p\left(\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ y_0 \\ y_1 \\ y_2 \end{bmatrix}\right) = \begin{bmatrix} \min(x_0, -y_0 + 1) x_2 \\ x_1 x_2 \\ x_2^2 \end{bmatrix}. \tag{7}$$

These functions correspond to the iterative updates of $h^{(l)}[a]$ and $h^{(l)}[p]$ in Alg. 1, recalling that L is the number of iterations it takes for the algorithm converges. More specifically, suppose we have a node u with embedding $\mathbf{h}_u = [x_0, x_1, x_2]$ and aggregated embedding from its neighbours $\mathbf{h}_N = [y_0, y_1, y_2]$. Then we have two cases.

- If $x_2 = 0$, indicating that the node u corresponds to a n action, then we get

$$f_a([\mathbf{h}_u \,\|\, \mathbf{h}_N]) = [-y_0, 0, 0] \tag{8}$$

$$f_p([\mathbf{h}_u \,\|\, \mathbf{h}_N]) = [0, 0, 0]. \tag{9}$$

  Eq. 8 corresponds to Line 7 in Alg. 1 where $-y_0$ contains the negative of $h[a]$. We take the negative since we are restricted to using $\max$ aggregators only[1] which in turn means we require taking maximums of negatives in order to mimic the minimum aggregator later in Line 9 of the same algorithm. Eq. 9 corresponds to Line 9 but since this line only affects propositions and $h[a]$ values do not need to be stored after execution of this line, we set $\mathbf{h}_u$ to zero.

- If $x_2 = 1$, indicating that the node u corresponds to a proposition, then we get

$$f_a([\mathbf{h}_u \,\|\, \mathbf{h}_N]) = [x_0, x_1, x_2] \tag{10}$$

$$f_p([\mathbf{h}_u \,\|\, \mathbf{h}_N]) = [\min(x_0, -y_0 + 1), x_1, x_2]. \tag{11}$$

  We recall f_a corresponds to Line 7 which only affects $h[a]$ values. Given that we require storing $h[p]$ values throughout the whole algorithm, f_a acts as the identity function on $\mathbf{h}_u$ for proposition nodes as seen in Eq. 10. This is in contrast to f_p which acts as the zero function on $\mathbf{h}_u$ for action nodes. Eq. 11 corresponds to Line 9 where $-y_0$ is equivalent to the $\min_{a \in A, p \in \mathrm{add}(a)} h[a] = \max_{a \in A, p \in \mathrm{add}(a)} -h[a]$ term by definition of DLG, agg and f_a acting on action node embeddings.

We append a final layer to the network where we ignore neighbourhood nodes with $\mathrm{agg}^{(2L+1)} = \vec{0}$ and $\varphi^{(2L+1)}([x_0, x_1, x_2]^\top, \mathbf{h}_N) = x_0 x_1$. In combination with a max readout function Φ, this corresponds to computing the final heuristic value. The above encoding of Alg. 1 has also been experimentally verified to be correct.

In order to satisfy the neural network component of the MPNN, we replace the $\varphi^{(i)}$ for $i = 0, \ldots, 2L+1$ with feedforward networks. Noting that we have finitely many layers we can choose small enough fractions of ε for the universal approximation theorem for neural networks [Hornik et al., 1989, Cybenko, 1989] to approximate the continuous functions $\varphi^{(i)}$ whose domain is bounded in the ball of radius B in order to achieve our result.

The encoding for h^{add} is the same except we use a sum aggregator $\mathrm{agg} = \sum$ and readout.

[1]As min aggregators conflict with ReLU activation functions commonly seen in neural networks.

For the case of the other FLG, we note that the $h^{\max}$ and h^{add} algorithm for FDR problems and hence FLG graph representations work in the obvious way by compiling FDR planning tasks into propositional STRIPS planning task by treating variable-value pairs in FDR problems as propositional facts. $\square$

Theorem 4.2 (MPNNs on grounded graphs are strictly more expressive than STRIPS-HGN)**.** Let $\mathcal{G} \in \{\text{SLG}, \text{FLG}\}$. 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.

Proof sketch. To show the first part of the theorem, we describe how to construct an MPNN $\mathcal{F}_\Phi$ acting on SLG and FLG corresponding to a given STRIPS-HGN Shen et al. [2020] model $\mathcal{S}_\Theta$ 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 $\mathcal{F}_\Phi(\mathcal{G}(\Pi_1)) \neq \mathcal{F}_\Phi(\mathcal{G}(\Pi_2))$. First we note that each STRIPS-HGN hypergraph message passing layer can be emulated by two MPNN message passing layers. We note that the STRIPS-HGN aggregation function is not permutation invariant as it requires ordering the messages it receives before concatenating them and updating the aggregated feature. This can similarly be done for a MPNN. Another difference with STRIPS-HGN and MPNNs is the usage of global features that are updated with each message passing layers. The MPNN framework can also be extended to make use of global features, for example by appending a virtual node to the whole input graph, and using different weights for the message passing functions associated with the virtual node. Lastly, STRIPS-HGN uses the same weights for each message passing layer and this may also be done for an MPNN.

For the second part of the theorem, we note that for any planning problem Π by definition of STRIPS-HGN, $\mathcal{S}_\Theta(\Pi) = \mathcal{S}_\Theta(\Pi^+)$ for all parameters Θ where Π^+ is the delete relaxation of Π. Now consider the STRIPS problem $\Pi = \langle P, A, s_0, G\rangle$ with $P = G = \{p_0, p_1\}$, $s_0 = \{p_0\}$, and $A = \{a_0, a_1\}$ where both a_0 and a_1 have empty precondition and

$$\text{add}(a_0) = \{p_1\}, \qquad \text{del}(a_0) = \{a_0\}$$
$$\text{add}(a_1) = \{a_1\}, \qquad \text{del}(a_1) = \emptyset.$$

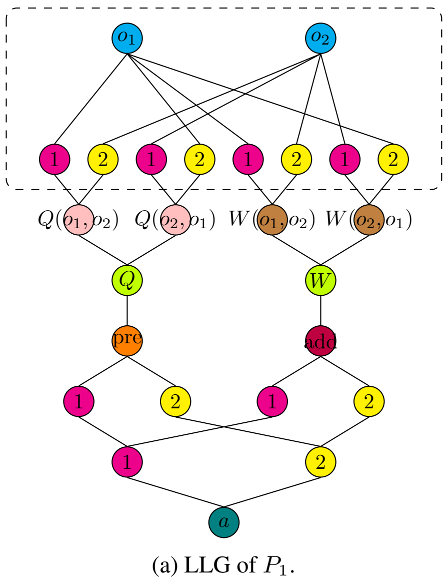
Then the optimal plan for Π has cost 2, while the optimal plan cost of its delete relaxation Π^+ is 1. There exists a set of parameters for an MPNN $\mathcal{F}_\Phi$ acting on $\mathcal{G} \in \{\text{SLG}, \text{FLG}\}$ such that $\mathcal{F}_\Phi(\mathcal{G}(\Pi)) = 2 \neq 1 = \mathcal{F}_\Phi(\mathcal{G}(\Pi^+))$. $\square$

Theorem 4.3 (MPNNs cannot learn h^{add} and $h^{\max}$ on lifted graphs)**.** Let $h \in \{h^{\text{add}}, h^{\max}\}$. 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 $\mathcal{F}_\Theta(\text{LLG}(\Pi_1)) = \mathcal{F}_\Theta(\text{LLG}(\Pi_2))$.

Proof. Consider the two (delete free) lifted problems $P_1 = \langle \mathcal{P}, \mathcal{O}, \mathcal{A}, s_0^{(1)}, G\rangle$ and $P_2 = \langle \mathcal{P}, \mathcal{O}, \mathcal{A}, s_0^{(2)}, G\rangle$ with $\mathcal{P} = \{Q(x_1, x_2), W(x_1, x_2)\}$, $\mathcal{O} = \{o_1, o_2\}$, $s_0^{(1)} = \{Q(o_1, o_2), Q(o_2, o_1)\}$, $s_0^{(2)} = \{Q(o_1, o_1), Q(o_2, o_2)\}$, $G = \{W(o_1, o_2), W(o_2, o_1)\}$ and one action schema $\mathcal{A} = \{a\}$ with $\Delta(a) = \{\delta_1, \delta_2\}$, $\text{pre}(a) = \{Q(\delta_1, \delta_2)\}$, $\text{add}(a) = \{W(\delta_1, \delta_2)\}$ and $\text{del}(a) = \emptyset$.

By definition P_1 can be solved with a plan consisting of $a(o_1, o_2)$ and $a(o_2, o_1)$ in either order and the corresponding heuristic values are $h^{\max}(s_0^{(1)}) = h^{\text{add}}(s_0^{(1)}) = 1$. On the other hand P_2 is unsolvable in which case we have $h^{\max}(s_0^{(2)}) = h^{\text{add}}(s_0^{(2)}) = \infty$.

The graphs are indistinguishable by the WL algorithm where we colour nodes by mapping their features to the set of natural numbers for the LLG graphs, given that the set of possible node features is countable. Note that it is possible to extend the WL algorithm to deal with edge labelled graphs by replacing each labelled edge with a coloured node connected to the edge's endpoints. Fig. 5 illustrates the graph representations for LLG. Then the result follows by the contrapositive of [Xu et al., 2019, Lem. 2] as WL assigns the same output for both graphs, and hence any MPNN also assigns the same output. $\square$



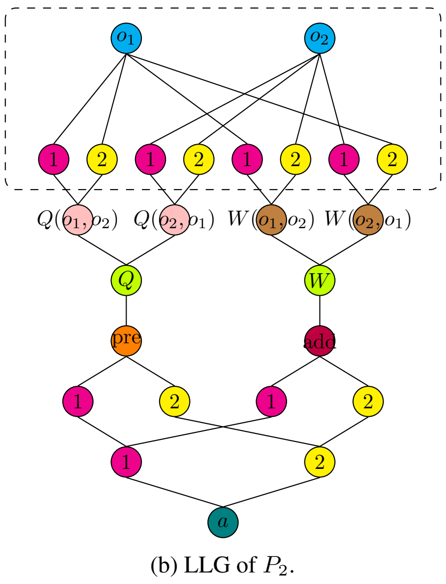
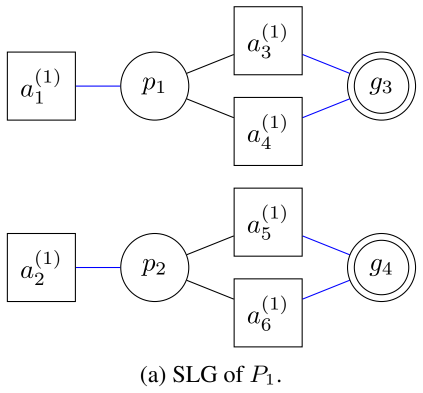


Figure 5: LLG of problems used in Thm. 4.3 (edges between objects and predicates omitted). Only colours are known to the WL algorithm, not the node descriptions in the figure. The only difference between the two graphs lies in the different edges between the top two layers of the graph as highlighted by the dashed regions. However, they are indistinguishable by the WL algorithm.



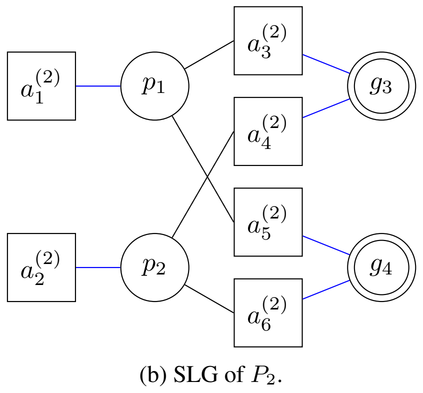


Figure 6: SLG of problems used in the proof of Thm. 4.4. Black edges indicate preconditions and blue edges indicate add effects.

Theorem 4.4 (MPNNs cannot learn h^+ or h^* with our graphs)**.** Let $h \in \{h^+, h^*\}$ and $\mathcal{G} \in \{\text{SLG}, \text{FLG}, \text{LLG}\}$. 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 $\mathcal{F}_\Theta(\mathcal{G}(\Pi_1)) = \mathcal{F}_\Theta(\mathcal{G}(\Pi_2))$.

Proof. Consider the two (delete free) planning problems $P_1 = \langle P, A_1, s_0, G\rangle$ and $P_2 = \langle P, A_2, s_0, G\rangle$ with $P = \{p_1, p_2, g_3, g_4\}$, $G = \{g_3, g_4\}$, $s_0 = \emptyset$ and action sets $A_1 = \{a_i^{(1)} \mid$

$i = 1, \dots, 6\}$, $A_2 = \{a_i^{(2)} \mid i = 1, \dots, 6\}$ where actions have no delete effects and are defined by

$$\begin{aligned}
\text{pre}(a_1^{(1)}) &= \emptyset, & \text{add}(a_1^{(1)}) &= \{p_1\}, \\
\text{pre}(a_1^{(2)}) &= \emptyset, & \text{add}(a_1^{(2)}) &= \{p_1\}, \\
\text{pre}(a_2^{(1)}) &= \emptyset, & \text{add}(a_2^{(1)}) &= \{p_2\}, \\
\text{pre}(a_2^{(2)}) &= \emptyset, & \text{add}(a_2^{(2)}) &= \{p_2\}, \\
\text{pre}(a_3^{(1)}) &= \{p_1\}, & \text{add}(a_3^{(1)}) &= \{g_3\}, \\
\text{pre}(a_3^{(2)}) &= \{p_1\}, & \text{add}(a_3^{(2)}) &= \{g_3\}, \\
\text{pre}(a_4^{(1)}) &= \{p_1\}, & \text{add}(a_4^{(1)}) &= \{g_3\}, \\
\text{pre}(a_4^{(2)}) &= \{p_1\}, & \text{add}(a_4^{(2)}) &= \{g_4\}, \\
\text{pre}(a_5^{(1)}) &= \{p_2\}, & \text{add}(a_5^{(1)}) &= \{g_4\}, \\
\text{pre}(a_5^{(2)}) &= \{p_2\}, & \text{add}(a_5^{(2)}) &= \{g_3\}, \\
\text{pre}(a_6^{(1)}) &= \{p_2\}, & \text{add}(a_6^{(1)}) &= \{g_4\}, \\
\text{pre}(a_6^{(2)}) &= \{p_2\}, & \text{add}(a_6^{(2)}) &= \{g_4\}.
\end{aligned}$$

We have that the minimum plan cost for P_1 is 4 by applying actions $a_1^{(1)}, a_2^{(1)}, a_3^{(1)}, a_5^{(1)}$ whereas the minimum plan cost for P_2 is 3 with actions $a_1^{(1)}, a_3^{(1)}, a_5^{(1)}$, as seen in Fig. 6. Both h^+ and h^* return 4 for P_1 and 3 for P_2.

Colour refinement assigns the same invariant to the graph representations of P_1 and P_2 and thus by the contrapositive of [Xu et al., 2019, Lem. 2], any MPNN assigns the same embedding to both graphs. $\square$

Theorem 4.5 (MPNNs cannot learn any approximation of h^*)**.** Let $\mathcal{G} \in \{\text{SLG}, \text{FLG}, \text{LLG}\}$ 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 $|\mathcal{F}_\Theta(\mathcal{G}(\Pi_1)) - h(\Pi_1)| \leq c \wedge |\mathcal{F}_\Theta(\mathcal{G}(\Pi_2)) - h(\Pi_2)| \leq c$. Furthermore, for any set of parameters we do not have $\left|1 - \frac{\mathcal{F}_\Theta(\mathcal{G}(\Pi_1))}{h(\Pi_1)}\right| \leq c \wedge \left|1 - \frac{\mathcal{F}_\Theta(\mathcal{G}(\Pi_2))}{h(\Pi_2)}\right| \leq c$.

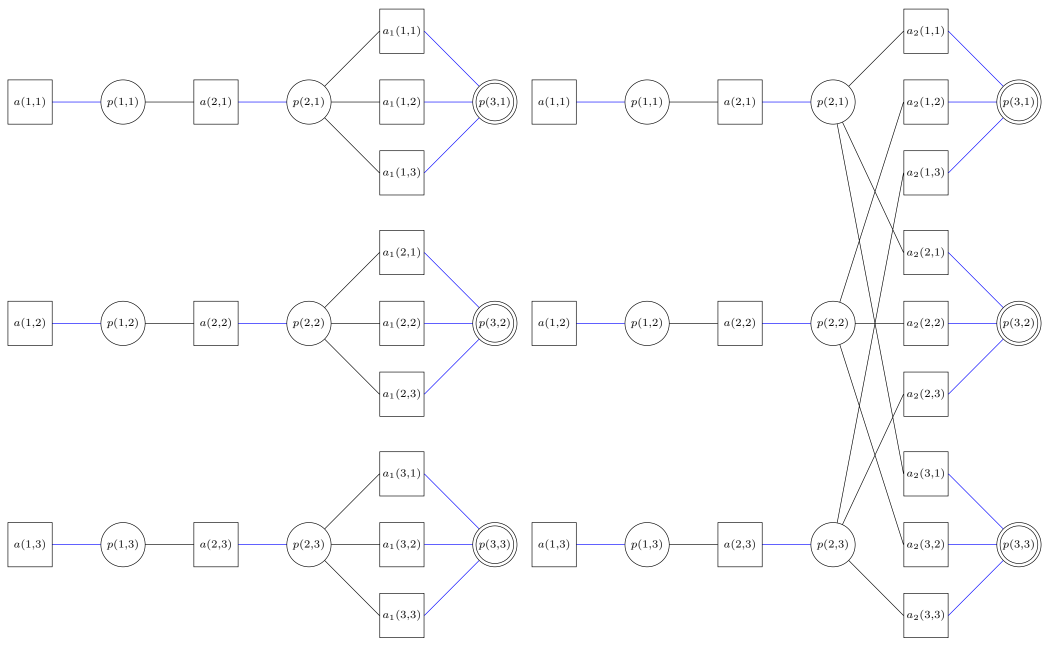
Proof. Let us fix $n \in \mathbb{N}$ with $n > 2$. Then we will construct a pair of planning problems whose optimal plan costs are $2n - 1$ and n^2 respectively but are indistinguishable by MPNNs by any graph representations $\mathcal{G} \in \{\text{SLG}, \text{FLG}, \text{LLG}\}$ of the problems. Thus, we can make our absolute and relative errors, given by $n^2 - 2n + 1$ and $\frac{n^2}{2n-1}$ respectively, arbitrary large.

Consider the two (delete free) planning problems given by $P_1 = \langle P, A_1, s_0, G\rangle$ and $P_2 = \langle O, A_2, s_0, G\rangle$ with $P = \{p(x, y) \mid x, y \in [n]\}$, $G = \{p(n, y) \mid y \in [n]\} \subset P$, $s_0 = \emptyset$ and actions $A_1 = \{a_1(y, z) \mid y, z \in [n]\} \cup A$ and $A_2 = \{a_2(y, z) \mid y, z \in [n]\} \cup A$ where $A = \{a(x, y) \mid x \in [n - 1], y \in [n]\}$. All actions have no delete effects and their preconditions and add effects are given as follows

$$\begin{aligned}
\text{pre}(a(1, y)) &= \emptyset, & \text{add}(a(1, y)) &= \{p(1, y)\}, & &\forall y \in [n] \\
\text{pre}(a(x, y)) &= \{p(x - 1, y)\}, & \text{add}(a(x, y)) &= \{p(x, y)\}, & &\forall x \in [2..n - 1], y \in [n] \\
\text{pre}(a_1(y, z)) &= \{p(n - 1, y)\}, & \text{add}(a_1(y, z)) &= \{p(n, y)\}, & &\forall y, z \in [n] \\
\text{pre}(a_2(y, z)) &= \{p(n - 1, z)\}, & \text{add}(a_2(y, z)) &= \{p(n, y)\}, & &\forall y, z \in [n]
\end{aligned}$$

where we note that the case $n = 2$ is given in the proof of Thm. 4.4. We refer to Fig. 7 for the case of $n = 3$. An optimal plan for P_1 consists of executing all actions $a \in A$ and $a_1(y, 1)$ for $y \in [n]$. On the other hand, an optimal plan for P_2 consists only of executing $a(x, 1)$ for $x \in [n - 1]$ followed by $a_2(y, 1)$ for all $y \in [n]$. Thus, the optimal plan costs for P_1 and P_2 are n^2 and $2n - 1$ respectively.

As in the previous proof, any graph representations of the pair of problems for any n are indistinguishable by colour refinement and hence by MPNNs [Xu et al., 2019, Lem. 2]. $\square$



(a) SLG of P_1 for $n = 3$.

(b) SLG of P_2 for $n = 3$.

Figure 7: SLGs of problems used in the proof of Thm. 4.5 where black edges indicate preconditions and blue edges indicate add effects.