State Encodings for GNN-Based Lifted Planners

Rostislav Horčík, Gustav Šír, Vítězslav Šimek, Tomáš Pevný
Czech Technical University in Prague
xhorcik@fel.cvut.cz, gustav.sir@cvut.cz, simekvi1@fel.cvut.cz, pevnak@protonmail.ch

Abstract

The application of graph neural networks (GNNs) to learn heuristic functions in classical planning is gaining traction. Despite the variety of methods proposed in the literature to encode classical planning tasks for GNNs, a comparative study evaluating their relative performances has been lacking. Moreover, some encodings have been assessed solely for their expressiveness rather than practical effectiveness in planning. This paper provides an extensive comparative analysis of existing encodings. Our results indicate that the smallest encoding based on Gaifman graphs, not yet applied in planning, outperforms the rest due to its fast evaluation times and the informativeness of the resulting heuristic. The overall coverage measured on the IPC almost reaches that of the state-of-the-art planner LAMA while exhibiting rather complementary strengths across different domains.

Code — https://github.com/pevnak/NeuroPlanner.jl

1 Introduction

The use of neural networks (NNs) to learn heuristic functions in classical planning is on the rise. Initial efforts developed neural network architectures based on grounded representations of planning tasks, such as ASNets (Toyer et al. 2018) and STRIPS-HGNs (Shen, Trevizan, and Thiébaux 2020). However, these grounded representations are usually quite large, resulting in significantly large NNs. Consequently, their evaluation time renders them less competitive in typical search algorithms compared to classical planners like LAMA (Richter and Westphal 2010). This has shifted current research focus primarily towards the application of Graph Neural Networks (GNNs) to lifted representations instead (Chen, Thiébaux, and Trevizan 2024; Ståhlberg, Bonet, and Geffner 2022a,b, 2023). To effectively utilize GNNs as heuristic functions for planning tasks, it is necessary to represent planning states as appropriate graph structures referred to as *state encodings*, as illustrated in the general GNN-based pipeline shown in Figure 1.

Existing studies typically fix a particular state encoding and measure the performance of the resulting heuristic function represented by the pipeline. However, no comparative study has investigated the performance effect of different

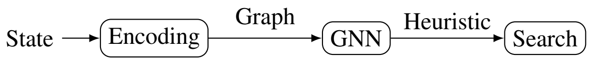

Figure 1: A GNN-based pipeline in classical planning.

kinds of state encodings on the pipeline. The performance measurement depends on the pipeline's application mode. The existing approaches work either in an offline or online setting. The offline setting incorporates the pipeline's heuristic function into a standard heuristic search algorithm like Greedy Best First Search (Chen, Thiébaux, and Trevizan 2024). In constrast, the online setting employs the pipeline's heuristic to construct the corresponding greedy policy (Ståhlberg, Bonet, and Geffner 2022a,b, 2023). In the offline setting, it is crucial to balance evaluation time with informativeness, given the time constraints on the search. Conversely, the online setting must optimize for informativeness and less for evaluation time.

The informativeness is tightly connected to the pipeline's expressiveness, determining the range of heuristic functions that can be learned. This expressiveness was investigated in (Horčík and Šír 2024) where the state encodings for GNNs were compared based on their ability to differentiate between states, as it is assumed that a network capable of distinguishing more states is more powerful and ultimately leads to stronger heuristics. While this is true, the computational complexity is the other side of the problem, impacting the heuristic's overall effectiveness, especially in the offline setting. Consequently, a less powerful but computationally cheaper heuristic may be more advantageous.

This paper, therefore, discusses the computational aspects of different state encodings and compares them across problems from the IPC 2023 learning track. In particular, we study not only the obtained coverage for a given time limit but also the size of the state encodings, evaluation time, the number of expanded states during the search, and plan lengths. To evaluate the state encodings, we apply them offline to simultaneously measure the evaluation time and the informativeness via the number of expanded states. We leverage A* as the search algorithm.

To construct the GNN-based pipeline, we consider all the relevant state encodings from the literature, namely (Chen, Thiébaux, and Trevizan 2024; Ståhlberg, Bonet, and Geffner

2022a; Horčík and Šír 2024). These encodings are reviewed in Section 3. To train the pipeline, we use optimal plans together with a loss function from (Chrestien et al. 2023), as detailed in Section 4. This loss function penalizes deviations from the optimal plan by the A* algorithm. Even though the trained heuristic need not be admissible, we use A* as we train the pipeline to behave optimally and expect to get close to optimal plans. Section 5 summarizes our experimental results. Finally, we discuss related work and present our conclusions.

2 Background

This section introduces our notation and the necessary background. For a natural number $m \geq 1$, the set $\{1, \ldots, m\}$ is denoted as $[m]$. To construct multisets, we use double curly brackets $\{\!\{\ldots\}\!\}$. Given a multiset M of vectors of the same dimension, we define $\mathsf{sum}(M)$ as the sum of all vectors in M and $\mathsf{max}(M)$ as their component-wise maximum.

Classical Lifted Planning

A *relational language* $\mathcal{L}$ consists of predicate symbols together with their respective arities and variables. For simplicity, we assume that $\mathcal{L}$ does not contain constants. For our purposes, they can be simulated by unary predicates. More precisely, for each constant c, one introduces a unary predicate symbol P_c such that $P_c(x)$ holds iff $x = c$.

An *atomic formula* is an expression of the form $R(x_1, \ldots, x_n)$ where R is an n-ary predicate symbol and each x_i is a variable. A *(ground) atom* over a set of objects O is an expression of the form $R(b_1, \ldots, b_n)$ where R is an n-ary predicate symbol and each $b_i \in O$.

A *classical planning task* is a pair $P = \langle D, I\rangle$ where D is a domain and I an instance. The *domain* is a pair $D = \langle \mathcal{L}, \mathcal{A}\rangle$ where $\mathcal{L}$ is a relational language and $\mathcal{A}$ is a collection of *action schemata*. Each action schema $a(\vec{v}) \in \mathcal{A}$ has a tuple of its variable parameters $\vec{v}$ and consists of three sets of atomic formulas built from the parameters $\vec{v}$: preconditions $\mathsf{pre}_a(\vec{v})$, add effects $\mathsf{add}_a(\vec{v})$, and delete effects $\mathsf{del}_a(\vec{v})$. In this paper, we assume that all actions have a unit cost.

The *instance* I is a triple $\langle O, s_0, \psi_G\rangle$ where O is a set of objects, s_0 and ψ_G are sets of ground atoms called *initial state* and *goal*, respectively. A *state* is a set of ground atoms. A *goal state* is a state s such that $\psi_G \subseteq s$.

For an action schema $a(\vec{v})$ and a tuple of objects $\vec{b}$ of the same length as $\vec{v}$, the (ground) *action* $a(\vec{b})$ is created from $a(\vec{v})$ by substituting $\vec{b}$ for $\vec{v}$. We say that a ground action $a(\vec{b})$ is *applicable* in the state s if $\mathsf{pre}_a(\vec{b}) \subseteq s$. The application of $a(\vec{b})$ in s gives us a new state $s' = (s \setminus \mathsf{del}_a(\vec{b})) \cup \mathsf{add}_a(\vec{b})$.

A *plan* is a sequence of ground actions $\pi = a_1(\vec{b}_1), \ldots, a_n(\vec{b}_n)$ transforming the initial state s_0 into a goal state. In other words, applying subsequently the actions in π at s_0 generates a sequence of states $s_0, s_1, \ldots, s_n$ such that $\psi_G \subseteq s_n$. A plan is called *optimal* if it is shortest among all plans.

One way to compute a plan is to apply a heuristic search algorithm like A*. Recall that A* maintains a priority queue Open consisting of states waiting to be expanded in each iteration. At a given iteration, A* selects for expansion the state from Open based on its f-scores, i.e., $f(s) = g(s) + h(s)$ where $g(s)$ is the currently known best cost of getting from s_0 to s and $h(s)$ is a heuristic value that estimates the cost of getting to a goal state.

Our GNN-based pipeline is employed in the A* algorithm, where the GNN predicts the heuristic value $h(s)$ based on its parameters θ, which we learn from optimal plans of the training instances. To provide the GNN with information on the goal, we integrate the goal ψ_G into states by expanding the language $\mathcal{L}$ as in (Ståhlberg, Bonet, and Geffner 2022a). We define an expansion $\mathcal{L}_G$ of $\mathcal{L}$ so that $\mathcal{L}_G$ contains each n-ary predicate R from $\mathcal{L}$ together with its copy R_G. Given a state s and a goal ψ_G, its *enriched state* is $s_G = s \cup \{R_G(\vec{c}) \mid R(\vec{c}) \in \psi_G\}$. From now on, when we refer to a state, we mean the state enriched with the goal information.

Graph Structures

We employ several kinds of labeled graphs. All versions of our graphs are undirected. In other words, an *edge* between vertices u, v is an unordered pair of vertices $\{u, v\}$. A *graph* is a pair $G = \langle V, E\rangle$ where V is a set of vertices and E is a set of edges. A generalization of a graph having k types of edges is called an *edge-typed graph*. It is a tuple $G = \langle V, E_1, \ldots, E_k\rangle$ where V is a set of vertices and each E_i is a set of edges of type i.

Graphs allowing parallel edges among nodes are called multigraphs. Formally, a *multigraph* is a triple $G = \langle V, E, r\rangle$ where V is a set of nodes, E a set of edges and $r\colon E \to 2^V$ a map assigning to every edge $e \in E$ its endpoints $r(e) = \{u, v\} \subseteq V$. Note that edges in multigraphs are not identified with 2-element sets of vertices as in (edge-typed) graphs. Further, observe that any graph $G = \langle V, E\rangle$ can be viewed as a special multigraph $\langle V, E, r\rangle$ where r is the identity map.

We call any of the above graph structures *bipartite* if V can be partitioned into two disjoint subsets V_1, V_2 such that the edges can only connect vertices from V_1 with vertices from V_2, i.e., there are no edges connecting two vertices from the same subset V_i.

In the following text, we will label the vertices by real-valued vectors. Given a vertex $v \in V$, we denote its label by $\boldsymbol{v} \in \mathbb{R}^n$. We assume that the dimension n of all vertex labels is the same unless the underlying graph structure is bipartite. In that case, each partition can have its own dimension.

Further, we define two conversions. The first converts an edge-typed graph into a multigraph. The second converts the resulting multigraph into a graph. Let $G = \langle V, E_1, \ldots, E_k\rangle$ be an edge-typed graph. One can transform G into a multigraph G_m by forgetting the edge types. Formally, $G_m = \langle V, E, r\rangle$ where $E = E_1 \dot\cup \cdots \dot\cup E_k$ is the disjoint union of all typed edges. In other words, the elements of E are pairs $\langle e, i\rangle$ for $e \in E_i$ and $i \in [k]$. Finally, we define $r(e, i) = e$. To recover the type information, we label each edge by the one-hot encoding of its type. More precisely, an edge $\langle e, i\rangle \in E$ of type i is labeled by the vector $\langle 0, \ldots, 1, \ldots, 0\rangle$ consisting of all zeros except the i-th component which is one.

The second conversion takes the multigraph $G_m = \langle V, E, r\rangle$ and converts it into a graph $G_g = \langle V, r(E)\rangle$ by deduplicating the parallel edges; formally $r(E) = \{r(e, i) \mid$

$\langle e, i \rangle \in E\}$. To recover the type information, we label each resulting edge $\{u, v\}$ by the vector whose i-th component is one if there was an edge of type i connecting u and v, and zero otherwise. In machine learning terminology, the labeling vector is the multi-hot encoding of the edge types.

Graph Neural Networks

We recall the message-passing architecture of GNNs for the above graph structures; for an overview see (Zhou et al. 2020). Let G be any of the above graph structures. Independently of its type, a GNN consists of several layers $\ell_1, \ldots, \ell_k$. Each layer ℓ_i for $i < k$ takes G and updates its vertex labeling. More precisely, the layer ℓ_i computes "messages" for each vertex $v \in V$ based on the vertex labels of its neighbors and the labels of the corresponding edges. These messages are aggregated and combined with the vertex label of v to produce the updated vertex label for v. The last layer ℓ_k is called a *read-out* layer, and it aggregates all the vertex labels into a single output; in our case, a real number.

Next, we explicitly define the layer ℓ_i, $i < k$ for particular types of graph structures. Assume that $G = \langle V, E, r \rangle$ is a multigraph or a graph. For a node $v \in V$, we denote the vector label computed by the i-th layer as $\boldsymbol{v}^{(i)}$. Similarly, $\boldsymbol{e}$ represents the vector labeling of the edge $e \in E$. There is no layer index because each layer's edge labels remain the same. We define a multiset of messages for each layer i and vertex $v \in V$ based on all its neighbors:

$$M^{(i)}(v) = \{\!\{\mathsf{msg}(\boldsymbol{u}^{(i-1)}, \boldsymbol{e}) \mid e \in E,\ r(e) = \{v, u\}\}\!\}.$$

The function msg is represented by a multilayer perceptron (MLP) with its parameters tuned by training. The messages are aggregated and combined with the vertex label of v from the previous layer as follows:

$$\boldsymbol{v}^{(i)} = \mathsf{comb}(\boldsymbol{v}^{(i-1)}, \mathsf{agg}(M^{(i)}(v))).$$

Similarly, as for msg, the function comb is represented by an MLP with tunable parameters. Moreover, both functions need not be the same across the GNN layers, i.e., each layer has its tuple of parameters. The function agg is an aggregation function like sum or max applied over the multiset $M^{(i)}(v)$.

In case when G is bipartite, i.e., $V = V_1 \cup V_2$, we use two message functions (MLPs) msg^1 and msg^2 for each part since their labels can be of different dimensions. For instance, if $v \in V_2$, we apply msg^1 as neighbors of v can be only from V_1:

$$M^{(i)}(v) = \{\!\{\mathsf{msg}^1(\boldsymbol{u}^{(i-1)}, \boldsymbol{e}) \mid e \in E,\ r(e) = \{v, u\}\}\!\}.$$

Now, assume that $G = \langle V, E_1, \ldots, E_n \rangle$ is an edge-typed graph. In that case, the messages are created for each edge type separately. Further, we do not use edge labels for edge-typed graphs. For $v \in V$ and $j \in \{1, \ldots, n\}$ we define:

$$M_j^{(i)}(v) = \{\!\{\mathsf{msg}_j(\boldsymbol{u}^{(i-1)}) \mid \{u, v\} \in E_j\}\!\}.$$

where msg_j is an MLP corresponding to E_j. All the multisets $M_j^{(i)}(v)$ are aggregated, and the results are combined with the vertex label of v from the previous layer:

$$\boldsymbol{v}^{(i)} = \mathsf{comb}(\boldsymbol{v}^{(i-1)}, \mathsf{agg}(M_1^{(i)}(v)), \ldots, \mathsf{agg}(M_n^{(i)}(v))),$$

In case when G is bipartite, i.e., $V = V_1 \cup V_2$, we use two message functions msg_j^1 and msg_j^2 for each type j instead of a single function msg_j.

The final read-out layer ℓ_k for any graph structure G takes the multiset of all vertex labels from the layer ℓ_{k-1} and aggregates them into a single output value $\mathsf{ro}(\mathsf{agg}(M))$ where $M = \{\!\{\boldsymbol{v}^{(k-1)} \mid v \in V\}\!\}$. The function ro is again represented utilizing an MLP.

Further, we consider an encoding (denoted **Rel**) that conceptually follows (Ståhlberg, Bonet, and Geffner 2022a) which defines the message-passing schema directly over a state s of a planning task. Recall that s is a set of ground atoms over a set of objects O. Suppose that $P_1, \ldots, P_m$ are the unary predicates in the corresponding relational language.[1] For each object $o \in O$, we introduce its 0-th vector label $\boldsymbol{o}^{(0)} \in \mathbb{R}^m$ such that its j-th component is 1 if $P_j(o) \in s$; and 0 otherwise. For each object $o \in O$ and an n-ary predicate R of arity at least 2, we define the corresponding message multiset:

$$M_R^{(i)}(o) = \{\!\{\mathsf{msg}_R(\boldsymbol{o}_1^{(i-1)}, \ldots, \boldsymbol{o}_n^{(i-1)}) \mid R(o_1, \ldots, o_n) \in s, o \in \{o_1, \ldots, o_n\}\}\!\},$$

where msg_R is an MLP corresponding to the predicate R. Let $R_1, \ldots, R_n$ be the predicates of arity at least 2. The combination of messages is defined as follows:

$$\boldsymbol{o}^{(i)} = \mathsf{comb}(\boldsymbol{o}^{(i-1)}, \mathsf{agg}(M_{R_1}^{(i)}(o)), \ldots, \mathsf{agg}(M_{R_n}^{(i)}(o))).$$

The read-out layer for this encoding is the same as for the above graph structures. We simply aggregate all the object labels by means of agg and ro.

3 State Encodings

We consider all the existing encodings of states applied in learning for lifted planning. Each of them represents a state of a planning task as an edge-typed graph G. In the experimental section, we apply a corresponding type of GNN respectively to G, G_m, and G_g. Throughout this section, we fix a planning task $P = \langle D, I \rangle$ with a domain $D = \langle \mathcal{L}, \mathcal{A} \rangle$, an instance $I = \langle O, s_0, \psi_G \rangle$, and s a state in P (i.e., a set of ground atoms). As a running example illustrating each encoding, we use the state s in Example 1 from (Horčík and Šír 2024).

Example 1. Consider a language $\mathcal{L}$ consisting of a nullary predicate symbol N, two unary predicates T, L, a binary A, and ternary R. Recall that the language $\mathcal{L}_G$ is the extension of $\mathcal{L}$ by the goal copies of all the predicates. The state s over the set of objects $O = \{t, l_1, l_2\}$ is the following set of ground atoms: $\{N, T(t), L(l_1), L(l_2), A(t, l_1), A_G(t, l_2), R(t, l_1, l_2)\}$. The example is a modified version of the IPC domain *transport* where t is a truck and l_1, l_2 are locations. The unary predicates T, L represent types of objects. The ground atom $A(t, l_1)$ expresses that t is located at l_1. The road connections are captured by the predicate R. We consider R to be ternary to show the encodings' impact on predicates of higher arity. For the same reason, we add a dummy nullary predicate N.

[1] We understand nullary predicates as unary ones that either hold for all objects or none.

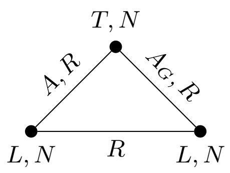

Figure 2: The object edge-typed graph $G(s)$ built from the state s introduced in Example 1.

The first encoding is called *object encoding* as it uses objects O as graph vertices and connects them by edges. It was introduced in (Horčík and Šír 2024) and constructs the state encoding as a Gaifman graph (Libkin 2004, Chapter 4) for the relational structure represented by the state s.

Definition 1. Let $R_1, \ldots, R_k \in \mathcal{L}_G$ be the predicates of arity at least two. We define an *object edge-typed graph* of s as $G(s) = \langle V, E_1, \ldots, E_k \rangle$ where $V = O$ and $\{c_1, c_2\} \in E_i$ iff c_1, c_2 occur in $\vec{c}$ for some $R_i(\vec{c}) \in s$. Unary atoms in s are encoded as vertex labels via the multi-hot encoding. More precisely, let $P_1, \ldots, P_m \in \mathcal{L}_G$ be unary predicates. For each $v \in V$, we define its label $\boldsymbol{v} \in \mathbb{R}^m$ as a vector whose i-th component is 1 if $P_i(v) \in s$; and 0 otherwise.

The object edge-typed graph $G(s)$ for the state s defined in Example 1 is shown in Fig. 2. The unary predicates are depicted as node labels. If several edges of different types connect two nodes, we list the types along a single edge.

The second encoding is called *atom encoding* as it uses ground atoms as graph vertices and connects them by edges (Horčík and Šír 2024).

Definition 2. Let m be the maximum arity of all the predicates in $\mathcal{L}$. We define an *atom edge-typed graph* of s as $G(s) = \langle V, \langle E_{i,j} \mid i, j \in [m] \rangle \rangle$ whose vertices are ground atoms, i.e., $V = s$. For atoms $\alpha, \beta \in V$, we define

$$\{\alpha, \beta\} \in E_{i,j} \text{ iff } \alpha = R(\vec{b}),\ \beta = R'(\vec{c}) \text{ and } b_i = c_j.$$

In other words, we connect α and β via $E_{i,j}$ iff the i-th object of the atom α is the same as the j-th object in the atom β.[2] In addition, each vertex is labeled by one-hot encoding of its predicate symbol.

The atom edge-typed graph for the running example is depicted in Figure 3. Similarly to Figure 2, the edge types connecting the same vertices are listed along a single edge.

Finally, the last state encoding is called *object-atom encoding* for its use of both objects and atoms as the graph vertices (Chen, Thiébaux, and Trevizan 2024; Horčík and Šír 2024). This encoding creates a bipartite graph. One partition is formed by the objects and the other one by the atoms.

Definition 3. Let m be the maximum arity of all the predicates in $\mathcal{L}$. We define *object-atom edge-typed graph* $G(s) = \langle V, E_1, \ldots, E_m \rangle$ where $V = O \cup s$. For an object $o \in O$ and an atom $\alpha \in s$ we define

$$\{o, \alpha\} \in E_i \text{ iff } \alpha = R(\vec{c}) \text{ and } o = c_i.$$

In other words, we relate the object o with the atom α via E_i iff o occurs at the i-th position within the atom α. The atoms are labeled by the one-hot encoding of their respective predicates, whereas objects are labeled by the multi-hot encoding reflecting the unary predicates they satisfy.

The object-atom edge-typed graph for the state s from Example 1 is shown in Figure 4. The object-atom encoding was applied in (Chen, Thiébaux, and Trevizan 2024; Chen, Trevizan, and Thiébaux 2024), with the difference that they did not use the enriched language $\mathcal{L}_G$ to represent the goal ψ_G but encoded ψ_G into the vertex labels instead.

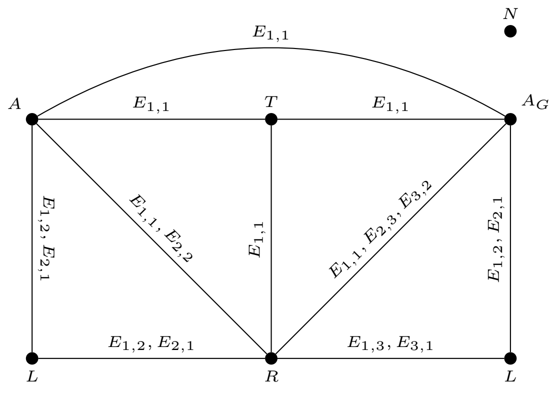

Figure 3: The atom edge-typed graph corresponding to the state from Example 1.

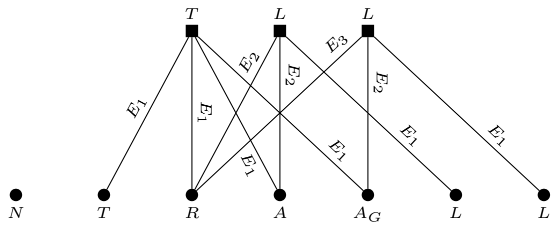

Figure 4: The object-atom edge-typed graph corresponding to the state from Example 1.

4 Loss Function

To train the GNN-based pipeline, we applied the ranking loss function L^* introduced in (Chrestien et al. 2023).[3] It does not optimize the GNN prediction to be close to the perfect heuristic. Instead, L^* penalizes if the learned heuristic generates a suboptimal search by the A* algorithm. It is easier for the GNN to learn which state from *Open* to expand next than the perfect heuristic.

We assume that our GNN-based pipeline predicts a heuristic value $h(s, \theta)$ for each state s depending on the GNN's

[2]To simplify the original encoding from (Horčík and Šír 2024), we use undirected edges instead of directed ones. Further, we also remove the edges that are defined based on the symmetric difference of the objects occurring in the atoms α and β.

[3]Ablation experiments with the more common regression ("cost-to-go") loss are provided in Appendix B.

	Object Encoding			Atom Encoding			Object-Atom Encoding						
Domain	**Gr**	**MGr**	**EGr**	**Gr**	**MGr**	**EGr**	**BipGr**	**BipMGr**	**BipEGr**	**Rel**	**ASNet**	**HGN**	**LAMA**
blocksworld	**58.5**	31.1	27.4	36.3	20.7	18.5	26.3	26.3	25.6	25.2	25.9	26.3	49.5
ferry	**65.9**	44.1	40.7	61.9	33.3	36.3	40.4	40.4	39.6	39.3	52.6	51.9	66.7
floortile	30.4	20.0	**31.9**	24.8	0.7	5.6	3.3	3.3	7.8	23.7	—	—	12.2
miconic	**67.8**	67.4	67.4	32.6	33.3	33.3	51.1	51.1	63.7	65.6	52.6	50.0	86.7
rovers	30.4	**32.6**	28.9	—	—	—	—	25.9	29.3	29.3	—	—	51.1
satellite	**39.6**	32.6	28.9	29.3	18.1	20.4	23.0	23.0	26.3	26.7	—	—	72.2
sokoban	28.5	**30.0**	27.0	—	—	21.9	21.9	22.2	26.3	**30.0**	—	—	38.9
spanner	57.8	**58.9**	56.7	48.1	49.3	50.7	57.0	57.4	58.1	57.8	38.5	38.9	33.3
transport	**38.5**	20.7	17.4	31.5	11.9	12.2	13.0	13.7	18.1	16.7	—	—	52.2
total	**46.4**	37.5	36.3	29.4	18.6	22.1	26.2	29.3	32.8	34.9	18.8	18.6	51.4

Table 1: The average coverage (in percent) on testing instances from three repetitions of the experiments. Each domain has 90 testing instances. **Gr**, **MGr**, **EGr** denote respectively graph $G_g(s)$, multigraph $G_m(s)$, and edge-typed graph $G(s)$. The prefix **Bip** denotes bipartite graph structure. **Rel** stands for the message-passing architecture built directly over the set of ground atoms (Section 2). The symbol "—" indicates that the training did not finish in 72 hours (due to excessive encoding size).

	Object econding			Atom encoding			Object-Atom encoding		Rel	
Domain	V	**Gr**	**MGr/EGr**	V	**Gr**	**MGr/EGr**	V	**Gr/MGr/EGr**	O	$\|s\|$
blocksworld	**17.7**	**39.9**	50.6	45.4	359.3	418.1	43.0	101.3	**17.7**	50.6
childsnack	**26.5**	**16.7**	**16.7**	28.1	107.3	113.8	34.9	33.5	**26.5**	**16.7**
ferry	**23.8**	**36.9**	48.9	26.4	132.2	166.9	48.3	97.7	**23.8**	48.9
floortile	**34.0**	**253.3**	278.0	156.7	5065.3	5529.1	173.0	556.0	**34.0**	278.0
miconic	**29.8**	**401.8**	**401.8**	217.8	14246.0	14620.4	230.7	803.7	**29.8**	**401.8**
rovers	**35.0**	**325.1**	918.4	261.1	37815.5	43241.8	273.7	1175.3	**35.0**	587.6
satellite	**42.0**	**193.6**	204.9	112.9	2688.0	2850.6	144.5	409.8	**42.0**	204.9
sokoban	**177.0**	1135.1	1520.0	341.5	37274.4	39529.2	435.7	1536.0	**177.0**	**768.0**
spanner	**28.0**	**54.0**	**54.0**	46.3	277.8	310.3	55.0	108.0	**28.0**	**54.0**
transport	**25.2**	**139.5**	146.0	73.0	2579.2	2783.2	98.2	292.0	**25.2**	146.0

Table 2: The average number of vertices V of $G(s)$ for states from the optimal plans for the five largest training instances. The average number of edges of $G_g(s)$, $G_m(s)$, and $G(s)$ are denoted respectively **Gr**, **MGr**, and **EGr**. For **Rel**, O denotes the average number of objects, and $|s|$ is the number of ground atoms. We omit the sizes for ASNets and STRIPS-HGN, noting that they are 1–2 orders of magnitude higher.

parameters θ. Let $\pi = a_1(\vec{b}_1), \ldots, a_n(\vec{b}_n)$ be an optimal plan with the corresponding sequence of states $s_0, s_1, \ldots, s_n$. Suppose that the A* algorithm expands only states $s_0, \ldots, s_{n-1}$. Let $\mathcal{O}_0 = \{s_0\}$ and let $\mathcal{O}_i$ be the set of states contained in $Open$ after expanding s_{i-1}. Thus $s_i \in \mathcal{O}_i$ and $s_j \notin \mathcal{O}_i$ for all $j < i$. The loss function L^* is evaluated for the whole plan π as follows:

$$L^*(\pi, \theta) = \sum_{i=1}^{n} \sum_{t \in \mathcal{O}_i \setminus \{s_i\}} [\![\varrho(s_i, t, \theta) > 0]\!],$$

where $\varrho(s_i, t, \theta) = g(s_i) - g(t) + h(s_i, \theta) - h(t, \theta)$ and $[\![\ldots]\!]$ denotes the Iverson bracket, that is 1 if its inner expression is true and 0 otherwise. Thus L^* penalizes if the f-scores in $Open$ at the i-iteration prefers t rather than s_i. To make L^* differentiable, we replace the Iverson bracket with the function $\log(1 + \exp(x))$ that behaves like a smooth ReLU. Thus, the resulting loss function is the following:

$$L^*(\pi, \theta) = \sum_{i=1}^{n} \sum_{t \in \mathcal{O}_i \setminus \{s_i\}} \log(1 + \exp(\varrho(s_i, t, \theta))).$$

5 Experiments

In our experiments, we use the GNN architectures described in Section 2. The concrete functions comb and msg are implemented as a single linear layer followed by the ReLU activation function. The read-out function ro is formed by two linear layers interleaved with ReLU. For the aggregation function we use, based on our experience, both sum and max, i.e., $\mathsf{agg}(M) = \langle \mathsf{sum}(M), \mathsf{max}(M) \rangle$. Regarding the hyperparameters, there is only a single hidden dimension of vertex labels $\boldsymbol{v}^{(i)}$ for $i > 0$ that is taken from $\{4, 16, 32\}$. The number of GNN layers ranges from 1 to 3.

The GNN-based pipeline was tested on the IPC 2023 learning track (Seipp and Segovia-Aguas 2023). To train the GNNs, we used the optimal plans for the training instances that we were able to solve by an optimal planner. To evaluate the loss function L^*, each optimal plan was "replayed" to construct the content $\mathcal{O}_i$ of the priority queue $Open$ for each iteration i of the A* search.

The overall coverage (i.e., the amount of solved problems for each planning domain) is shown in Table 1. Each domain was evaluated on AMD EPYC 7543 using eight cores and 64GB of memory limit allocated by the Slurm scheduler. The maximum running time was 72 hours. To assess the advantages of lifted representation, we measured the coverage with

	Object encoding			Atom encoding			Object-Atom encoding					
Domain	**Gr**	**MGr**	**EGr**	**Gr**	**MGr**	**EGr**	**BipGr**	**BipMGr**	**BipEGr**	**Rel**	**ASNet**	**HGNN**
blocksworld	28.5	**27.9**	29.3	115.4	117.8	115.6	53.2	48.0	46.2	29.8	588.3	567.7
childsnack	27.2	**26.0**	60.8	54.1	58.9	67.8	35.5	35.1	40.5	66.2	450.6	501.6
ferry	**28.5**	30.0	29.3	54.7	56.6	63.6	51.3	48.0	49.8	32.5	249.0	279.0
floortile	**92.3**	94.0	178.5	1235.9	1271.4	767.9	206.7	188.0	174.9	211.5	13666.0	13889.2
miconic	119.3	106.5	**103.6**	2802.4	2776.0	1557.9	278.4	243.1	216.9	142.3	1238.1	1325.4
rovers	**147.2**	386.6	492.9	7866.5	8623.5	5080.5	566.9	518.4	462.1	439.8	15888.2	15766.4
satellite	**77.2**	82.2	147.8	746.6	733.4	436.7	177.6	151.6	134.6	170.9	5857.7	5900.5
sokoban	379.2	453.8	363.4	7022.8	7133.6	3893.3	644.6	559.4	479.6	**325.0**	—	—
spanner	**33.8**	36.9	62.2	100.7	104.4	111.5	62.5	54.8	62.8	77.5	631.0	664.8
transport	57.9	**54.6**	100.9	639.5	646.0	339.0	114.0	110.9	96.6	122.0	4453.3	4175.4

Table 3: The average time of the heuristic computation, comprising of constructing the state encoding and the GNN inference, as evaluated over states from the optimal plans for the five largest training instances. The times are reported in microseconds.

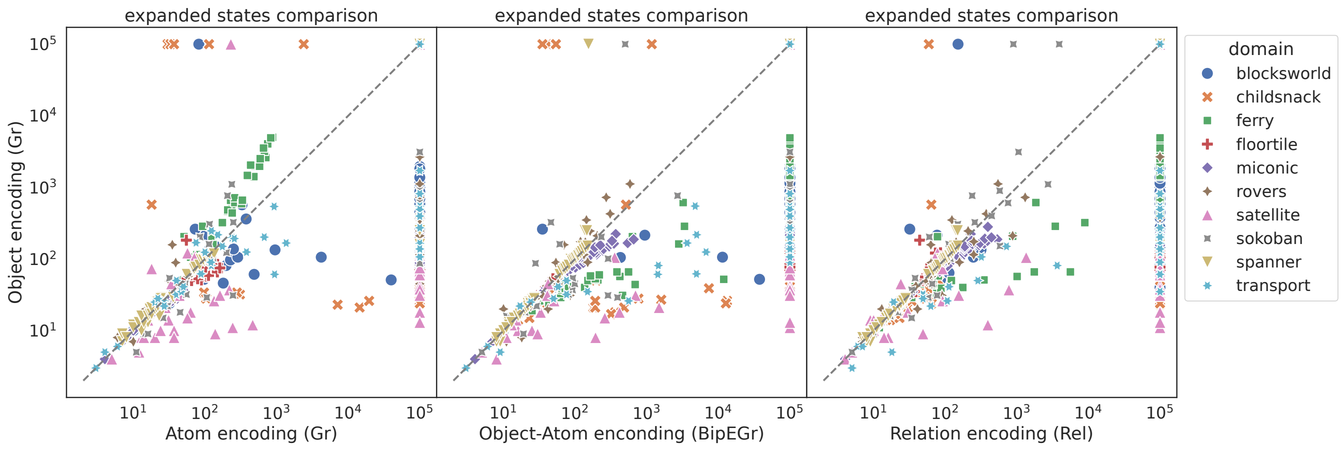

Figure 5: Numbers of expanded states across the encodings' best-performing variants, i.e., **Gr** for the object encoding, **Gr** for the atom encoding, **BipEGr** for object-atom encoding, and the message-passing architecture **Rel** built directly from states.

a short time limit per instance, namely 30 seconds. Due to the non-deterministic nature of the training procedure, we repeated the evaluation experiment three times with seeds 1, 2, and 3 and reported the average coverage. The GNN-based pipeline was trained for all combinations of the hyperparameters. To identify the optimal setting, we employed a cross-validation procedure. Testing instances were randomly split into two parts: one part was used to select the best hyperparameters based on coverage, while the second part's coverage was measured. The roles of the parts were then swapped, and coverage was measured for the first part instead.

For each state encoding (Section 3), we tested each variant of the graph structure, i.e., the edge-typed graph $G(s)$, the multigraph $G_m(s)$, and the graph $G_g(s)$. As shown in Table 1, object encoding is the most performant state encoding, especially its graph variant $G_g(s)$ that uses multi-hot labeling to represent edge types. Note that the object encoding is the smallest of all the discussed state encodings; see Table 2 for the sizes of particular state encodings. Also, its evaluation time is the fastest among all the tested approaches; see Table 3. Apparently, the shorter evaluation time helps the search algorithm explore more states within the given time limit. However, object encoding also provides a more informative heuristic in most cases. Figure 5 compares the numbers of expanded states across the encodings (their best-performing variants). It shows that the object encoding generally expanded a fewer, or similar, number of states compared to the other state encodings. It is worth noting that the atom encoding generates more informative heuristic in some cases, especially for the ferry domain.

We also compared the resulting coverage with neural networks based on grounded representations, namely ASNets (Toyer et al. 2018) and STRIPS-HGN (Shen, Trevizan, and Thiébaux 2020) denoted in Table 1 respectively as **ASNet** and **HGN**. Their evaluation times are 1-2 orders larger than those of the state encodings using lifted representation; see Table 3. Finally, we measured the coverage of the satisficing planner LAMA (Richter and Westphal 2010). As shown in Table 1, our best-performing state encoding solved nearly the same amount of testing instances. However, the domain-wise coverage of our best state encoding is, to a certain extent, complementary to that of LAMA. Figure 6 compares the lengths of plans generated by our best state encoding and LAMA. It also shows how close we get to the lengths of optimal plans found by the Fast Downward planner (Helmert 2006) using the LMCut heuristic (Helmert and Domshlak

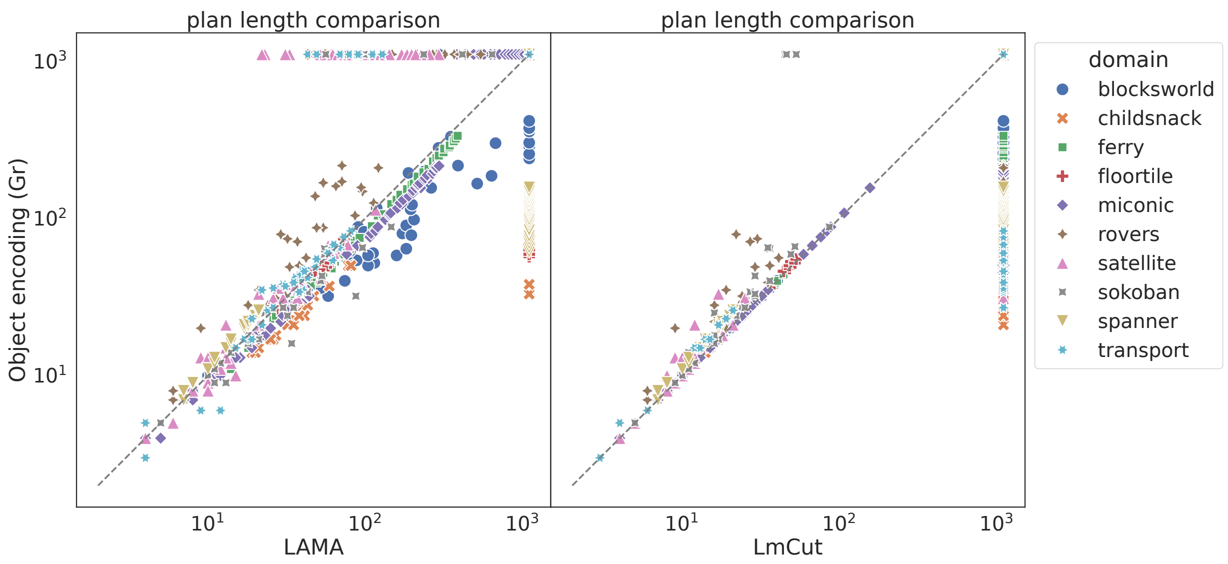

Figure 6: (Left) The comparison of the lengths of plans between LAMA (the first returned plan) and the graph variant of the object encoding. (Right) The comparison w.r.t. the lengths of the optimal plans found by LMCut.

	Object encoding			**Atom encoding**			**Object-Atom encoding**					
Domain	**Gr**	**MGr**	**EGr**	**Gr**	**MGr**	**EGr**	**BipGr**	**BipMGr**	**BipEGr**	**Rel**	**ASNet**	**HGN**
blocksworld	16 / 2	16 / 3	32 / 2	4 / 2	16 / 2	16 / 2	32 / 3	32 / 3	32 / 3	4 / 3	16 / 2	4 / 1
childsnack	4 / 1	4 / 1	4 / 1	4 / 3	4 / 3	32 / 1	16 / 1	16 / 1	4 / 2	4 / 1	4 / 2	4 / 3
ferry	4 / 1	32 / 3	16 / 2	4 / 3	16 / 3	16 / 3	16 / 3	16 / 3	16 / 3	32 / 2	4 / 1	4 / 1
floortile	16 / 3	32 / 2	4 / 3	32 / 1	32 / 1	16 / 2	32 / 2	32 / 2	32 / 3	32 / 2	32 / 1	4 / 1
miconic	4 / 1	4 / 1	4 / 1	4 / 1	4 / 1	4 / 1	4 / 1	4 / 1	4 / 2	4 / 1	4 / 1	4 / 1
rovers	4 / 1	4 / 2	4 / 1	32 / 1	4 / 1	16 / 1	4 / 3	4 / 3	4 / 1	32 / 2	4 / 1	4 / 3
satellite	16 / 2	16 / 2	16 / 2	4 / 1	4 / 2	4 / 2	16 / 2	16 / 2	16 / 3	32 / 2	4 / 1	16 / 3
sokoban	4 / 1	4 / 3	4 / 2	16 / 1	16 / 1	32 / 3	4 / 3	4 / 2	16 / 3	4 / 3	—	—
spanner	4 / 1	16 / 2	4 / 1	4 / 1	4 / 1	4 / 1	4 / 1	4 / 1	4 / 1	4 / 1	4 / 1	4 / 2
transport	4 / 1	16 / 3	32 / 2	4 / 1	32 / 2	16 / 3	16 / 3	16 / 3	4 / 3	32 / 3	4 / 1	4 / 1

Table 4: The best hyperparameter values (hidden dimension/number of GNN layers) as measured by average coverage.

2009).

Finally, Table 4 reports a post-hoc analysis of the optimal hyperparameter values, i.e., the hidden dimensionality of the MLPs and the number of GNN layers, for each state encoding and domain based on their average test coverage. Surprisingly, a single message-passing layer and a hidden dimensionality of 4 would often yield the best performance, confirming that smaller models work better in the offline setting.

6 Related Work

The first applications of deep neural networks in classical planning (Toyer et al. 2018; Shen, Trevizan, and Thiébaux 2020; Chrestien et al. 2023) were based on the grounded representation of the planning task, i.e., either STRIPS (Fikes and Nilsson 1971) or SAS+ (Bäckström and Nebel 1995). Recently, the research has shifted towards the lifted representation based on first-order logic, as given by the PDDL language (McDermott 2000). The first work based on lifted representation applied in the online setting was (Ståhlberg, Bonet, and Geffner 2022a), which introduced a message-passing architecture **Rel** built directly on states; see Section 2. A follow-up paper (Ståhlberg, Bonet, and Geffner 2022b) using the same architecture changed the loss function to allow the neural model to train without strict optimality constraints. The same authors further applied reinforcement learning over the same architecture (Ståhlberg, Bonet, and Geffner 2023). Another stream of research applied the lifted representation to learn a heuristic for a search algorithm. The object-atom encoding together with GNNs was investigated in (Chen, Thiébaux, and Trevizan 2024). The same state encoding was also applied in (Chen, Trevizan, and Thiébaux 2024). However, it did not use a GNN to process the underlying graph structures but employed the Weisfeiler-Lehman graph kernels (Shervashidze et al. 2011) together with a linear classifier. Finally, (Horčík and Šír 2024) introduced several state encodings (particularly the object and atom encodings) and studied their expressiveness in combination with different

	Object encoding			Atom encoding			Object-Atom encoding					
Domain	**Gr**	**MGr**	**EGr**	**Gr**	**MGr**	**EGr**	**BipGr**	**BipMGr**	**BipEGr**	**Rel**	**ASNet**	**HGN**
blocksworld	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.33	0.33	0.33	0.0	0.17	0.33
childsnack	159.0	159.0	159.0	**150.33**	**150.33**	**150.33**	159.0	159.0	159.0	321.5	159.0	165.17
ferry	0.83	0.83	0.83	**0.17**	**0.17**	**0.17**	6.67	6.67	6.67	0.83	0.83	7.17
floortile	2.33	2.33	2.33	**0.0**	**0.0**	**0.0**	6.83	6.83	6.83	1.0	**0.0**	15.17
miconic	8.0	8.0	8.0	**2.0**	**2.0**	**2.0**	**2.0**	**2.0**	**2.0**	**2.0**	—	—
rovers	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	13.5
satellite	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.33	0.33	0.33	**0.0**	**0.0**	67.67
sokoban	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	—	—
spanner	52.83	52.83	52.83	**26.0**	**26.0**	**26.0**	55.83	55.83	55.83	52.83	52.83	54.17
transport	1.67	1.67	1.67	**0.33**	**0.33**	**0.33**	2.17	2.17	2.17	1.67	1.67	1.83

Table 5: The average number of problematic indistinguishable state pairs encountered during the A* search (states that are in $\mathcal{O}_i$ with one belonging to the optimal plan).

	Object encoding			Atom encoding			Object-Atom encoding					
Domain	**Gr**	**MGr**	**EGr**	**Gr**	**MGr**	**EGr**	**BipGr**	**BipMGr**	**BipEGr**	**Rel**	**ASNet**	**HGN**
blocksworld	14.4	13.0	13.3	14.4	9.6	12.6	11.9	11.9	12.6	**15.2**	14.1	11.1
childsnack	5.2	5.7	7.3	5.8	6.9	6.3	6.3	6.3	5.8	8.4	**9.3**	8.2
ferry	**47.4**	18.9	23.0	35.2	18.9	21.9	15.9	15.9	15.6	18.9	34.1	28.9
floortile	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
miconic	34.1	34.1	**34.8**	20.4	22.2	27.4	31.5	32.2	33.7	32.2	28.5	26.7
rovers	7.0	5.6	4.8	7.0	**8.7**	**8.7**	6.7	5.2	5.3	8.5	7.2	5.2
satellite	**20.7**	16.3	15.2	12.2	10.0	11.1	12.2	11.5	15.2	14.1	17.9	9.0
sokoban	**26.7**	20.4	20.4	25.0	21.5	17.3	21.1	20.7	24.1	24.1	—	—
spanner	33.0	33.0	33.3	38.5	32.6	33.3	46.3	**47.0**	33.3	33.0	33.3	33.3
transport	**15.2**	11.5	10.0	10.7	8.6	9.8	9.3	9.3	10.0	10.4	14.0	13.0

Table 6: The same as Table 1 but for the loss function L_2.

GNN architectures. More precisely, the paper investigated how many problematic pairs of states occur in planning domains, i.e., pairs of states that cannot be distinguished by a given GNN-based pipeline, no matter the choice of training parameters.

7 Conclusions

We presented a comparative study of all existing state encodings used in learning GNNs as heuristic functions for classical planning. Our findings indicate that smaller representations generally yield better performance in search, particularly the object encoding that has not yet been tested in planning. The object encoding naturally aligns with the binary predicates most common in planning domains, effectively shortening the distances between the object representations compared to the more common object-atom encoding, which requires more GNN passes through the atom node representations.

A Indistinguishable State Pairs

Following (Horčík and Šír 2024), we have measured the number of pairs of states that our architectures (independently of the learned/chosen parameters) could not distinguish even though they need to be distinguished. Specifically, and a bit differently from (Horčík and Šír 2024), we have counted how many times a state on the optimal path cannot be distinguished from other states in the priority queue *Open* maintained by A*, which is a good approximation of how many times the actual search can be degraded. Table 5 shows the average numbers over the five most difficult problem instances from the training set for which there was an optimal plan.

As can be seen, the number of indistinguishable pairs is the lowest for the atom encoding which is, however, expensive to compute. The object encoding has only a slightly higher number of indistinguishable pairs but is much cheaper to compute, which might explain its best overall performance. Surprisingly, HGN displays very high numbers despite the large size of the graph encoding. These results also hint at why ASNets tend to work better than HGNs.

B Optimizing Cost-to-Goal

To demonstrate that the results are not an artifact of the L^* loss (Chrestien et al. 2023), we have repeated the experiment while optimizing the heuristic model to estimate the cost-to-goal, more commonly employed in the prior art. More precisely, we considered the usual L_2 loss function that minimizes the sum of squares between predicted heuristic value $h(s_i, \theta)$ and the perfect heuristic value $h^*(s_i)$:

$$L_2(\pi, \theta) = \sum_{i=1}^{n} (h(s_i, \theta) - h^*(s_i))^2.$$

Table 6 presents the coverage across all the compared architectures. The results show a pattern similar to that observed with the L^* loss, although with a generally lower overall coverage. Specifically, the object encoding is the best in 6 out of 10 cases, with a significant performance drop occurring merely on the *spanner* domain.

Acknowledgments

The first two authors have received funding from the European Union's Horizon Europe Research and Innovation program under the grant agreement TUPLES No 101070149. The last author was supported by the Czech Science Foundation grant—no. 22-32620S.

References

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS$^+$ Planning. *Computational Intelligence*, 11(4): 625–655.

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. 2024. Learning Domain-Independent Heuristics for Grounded and Lifted Planning. In Wooldridge, M. J.; Dy, J. G.; and Natarajan, S., eds., *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI'24)* , 20078–20086. AAAI Press.

Chen, D. Z.; Trevizan, F.; and Thiébaux, S. 2024. Return to Tradition: Learning Reliable Heuristics with Classical Machine Learning. In Bernardini, S.; and Muise, C., eds., *Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS'24)*, 68–76. AAAI Press.

Chrestien, L.; Edelkamp, S.; Komenda, A.; and Pevný, T. 2023. Optimize Planning Heuristics to Rank, not to Estimate Cost-to-Goal. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Fikes, R. E.; and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *AI*, 2: 189–208.

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

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 162–169. AAAI Press.

Horčík, R.; and Šír, G. 2024. Expressiveness of Graph Neural Networks in Planning Domains. In Bernardini, S.; and Muise, C., eds., *Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS'24)*, 281–289. AAAI Press.

Libkin, L. 2004. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer. ISBN 978-3-540-21202-7.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *aim*, 21(2): 35–55.

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

Seipp, J.; and Segovia-Aguas, J. 2023. International Planning Competition 2023 - Learning Track. https://ipc2023-learning.github.io/.

Shen, W.; Trevizan, F. W.; and Thiébaux, S. 2020. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In Beck, J. C.; Buffet, O.; Hoffmann, J.; Karpas, E.; and Sohrabi, S., eds., *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS'20)*, 574–584. AAAI Press.

Shervashidze, N.; Schweitzer, P.; Van Leeuwen, E. J.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12: 2539–2561.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022a. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS'22)*, 629–637. AAAI Press.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022b. Learning Generalized Policies without Supervision Using GNNs. In Kern-Isberner, G.; Lakemeyer, G.; and Meyer, T., eds., *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel, July 31 - August 5, 2022*.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2023. Learning General Policies with Policy Gradient Methods. In Marquis, P.; Son, T. C.; and Kern-Isberner, G., eds., *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023*, 647–657.

Toyer, S.; Trevizan, F. W.; Thiébaux, S.; and Xie, L. 2018. Action Schema Networks: Generalised Policies With Deep Learning. In McIlraith, S. A.; and Weinberger, K. Q., eds., *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18)*, 6294–6301. AAAI Press.

Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; and Sun, M. 2020. Graph neural networks: A review of methods and applications. *AI open*, 1: 57–81.