Learning General Policies for Classical Planning Domains: Getting Beyond C_2

Simon Ståhlberg[1] , **Blai Bonet**[2] and **Hector Geffner**[3,1]
[1]Linköping University, Sweden
[2]Universitat Pompeu Fabra, Spain
[3]RWTH Aachen University, Germany
simon.stahlberg@liu.se, bonetblai@gmail.com, hector.geffner@ml.rwth-aachen.de

Abstract

GNN-based approaches for learning general policies across planning domains are limited by the expressive power of C_2, namely; first-order logic with two variables and counting. This limitation can be overcomed by transitioning to k-GNNs, for $k = 3$, wherein object embeddings are substituted with triplet embeddings. Yet, while 3-GNNs have the expressive power of C_3, unlike 1- and 2-GNNs that are confined to C_2, they require quartic time for message exchange and cubic space for embeddings, rendering them impractical. In this work, we introduce a parameterized version of relational GNNs. When $t = \infty$, R-GNN[t] approximates 3-GNNs using only quadratic space for embeddings. For lower values of t, such as $t = 1$ and $t = 2$, R-GNN[t] achieves a weaker approximation by exchanging fewer messages, yet interestingly, often yield the C_3 features required in several planning domains. Furthermore, the new R-GNN[t] architecture is the original R-GNN architecture with a suitable transformation applied to the input states only. Experimental results illustrate the clear performance gains of R-GNN[1] and R-GNN[2] over plain R-GNNs, and also over edge transformers that also approximate 3-GNNs.

1 Introduction

General policies are policies that can be used to solve a collection of planning problems reactively [Srivastava *et al.*, 2008; Hu and Giacomo, 2011; Belle and Levesque, 2016; Bonet and Geffner, 2018; Illanes and McIlraith, 2019; Jiménez *et al.*, 2019]. For example, a general policy for solving all Blocksworld problems can place all blocks on the table, and then build up the target towers from the bottom up. Yet while nearly perfect general policies have been learned for many classes of planning domains [Toyer *et al.*, 2020; Rivlin *et al.*, 2020; Ståhlberg *et al.*, 2022a], one key expressive limitation results from the types of features used to classify state transitions or actions. In combinatorial approaches, features are selected from a domain-independent pool, created using a description logic grammar [Baader *et al.*, 2003] based on the given domain predicates [Bonet and Geffner, 2018; Bonet *et al.*, 2019], while in deep learning approaches, the features are learned using relational versions of graph neural networks [Scarselli *et al.*, 2009; Gilmer *et al.*, 2017; Hamilton, 2020]. A shared limitation of *both* approaches, however, is their inability to learn policies requiring complex logical features. This limitation arises in description logics from the C_2 fragment of first-order logic that they capture; namely, first-order logic limited to two variables and counting [Baader *et al.*, 2003], and in GNNs, from the type of message passing that is accommodated, where direct communication involves pairs of objects but no triplets [Grohe, 2021].

This expressive limitation, not always acknowledged, is serious. For example, although these methods can learn general policies for guiding an agent to a specific cell in an $n \times n$ grid containing *obstacles*, with positions and adjacency relations defined in terms of cells and atoms such as $\textsc{At}(c)$ and $\textsc{Adj}(c, c')$, they lack the *expressive capacity* when the relations are represented with atoms like $\textsc{At}(x, y)$, $\textsc{Adj}_1(x, x')$, and $\textsc{Adj}_2(y, y')$. Similarly, these methods are unable to learn policies for classical benchmark domains such as Logistics and Grid, that require composition of binary relations, which is beyond the scope of C_2 [Ståhlberg *et al.*, 2022b; 2023].

In principle, this limitation can be addressed by using richer grammars to generate non-C_2 features or by using k-GNNs, for $k = 3$, where triplets of objects are embedded instead of individual objects [Morris *et al.*, 2019]. It is known that 3-GNNs have the expressive power of C_3 logic, unlike the C_2 expressive power of 1- and 2-GNNs [Grohe, 2021]. Yet 3-GNNs do not scale up as they require cubic number of embeddings, and quartic time for exchanging messages.

In this paper, we introduce an alternative, parameterized version of Relational GNNs (R-GNNs). The architecture for R-GNN[t] mirrors that of plain R-GNNs and differs only in the input. While plain R-GNNs take the set of atoms S representing a planning state as input, R-GNN[t] accepts a transformed set of atoms $A_t(S)$ instead. At $t = 0$, R-GNN[t] approximates 3-GNNs weakly, while at $t = \infty$, it offers a strong approximation. Thus, the parameter t serves to balance expressive power with computational effort. Crucially, for lower values of t, such as $t = 1$ and $t = 2$, R-GNN[t]'s message passing runs in quadratic time in general while capturing the C_3 features that are essential in several planning domains. Our experiments demonstrate that R-GNN[t], even with small values of t, is practically feasible and significantly improves both the coverage and the quality of the learned general plans when compared to two baselines: plain relational GNNs and Edge-Transformers, a different architecture also aimed at approximating 3-GNNs [Bergen *et al.*, 2021].

The rest of the paper is organized as follows. We review first related work and background on planning, generalized planning, GNNs, and relational GNNs. We introduce then the parametric R-GNNs, the learning task, the baselines, the experimental results, and a careful analysis of them.

2 Related Work

General policies from logic. The problem of learning general policies has a long history [Khardon, 1999; Martín and Geffner, 2004; Fern *et al.*, 2006], and general policies have been formulated in terms of logic [Srivastava *et al.*, 2011; Illanes and McIlraith, 2019], regression [Boutilier *et al.*, 2001; Wang *et al.*, 2008; Sanner and Boutilier, 2009], and policy rules [Bonet and Geffner, 2018; Bonet *et al.*, 2019] that can be learned [Francès *et al.*, 2021; Drexler *et al.*, 2022].

General policies from neural nets. Deep learning (DL) and deep reinforcement learning (DRL) [Sutton and Barto, 1998; Bertsekas, 1995; François-Lavet *et al.*, 2018] have been used to learn general policies [Kirk *et al.*, 2023]. In some cases, the planning representation of the domains is used [Toyer *et al.*, 2020; Bajpai *et al.*, 2018; Rivlin *et al.*, 2020]; in most cases, it is not [Groshev *et al.*, 2018; Chevalier-Boisvert *et al.*, 2019], and in practically all cases, the neural networks are GNNs or variants. Closest to our work is the use of GNNs for learning general policies for classical planning [Ståhlberg *et al.*, 2022b; 2023].

GNNs, R-GNNs, and C_k logics. The use of GNNs is common when learning general policies where the number of objects change from instance to instance. This is because GNNs trained with small graphs can be used for dealing with larger graphs [Scarselli *et al.*, 2009; Gilmer *et al.*, 2017; Hamilton, 2020], and because states in classical planning are closely related to graphs: they represent relational structures that become graphs when there is a single non-unary relation that is binary and symmetric. In such a case, the graph vertices stand for the objects and the edges for the relation. Relational GNNs extend GNNs to relation structures [Schlichtkrull *et al.*, 2018; Vashishth *et al.*, 2019; Barcelo *et al.*, 2022], and our R-GNNs borrow from those used for min-CSP [Toenshoff *et al.*, 2021] and generalized planning [Ståhlberg *et al.*, 2022a].

There is a tight correspondence between the classes of graphs that can be distinguished by GNN, the WL procedure [Morris *et al.*, 2019; Xu *et al.*, 2019], and C_2 logic [Cai *et al.*, 1992; Barceló *et al.*, 2020; Grohe, 2021]. The expressive power of GNNs can be extended by replacing graph vertices by tuples of k-vertices. The resulting k-GNNs have the the power of the k-WL coloring algorithm, and hence the expressivity of C_k for $k > 2$. The "folklore" variant of the k-WL algorithm, k-FWL [Cai *et al.*, 1992], is more efficient as it has the power of (k+1)-WL while using $\mathcal{O}(n^k)$ memory. The edge-transformer can be thought of a variant of 2-GNN that aims to approximate the expressive power of C_3 logic using $\mathcal{O}(n^2)$ memory [Bergen *et al.*, 2021].

3 Background

We review planning, generalized planning, k-WL coloring, k-GNNs, and relational GNNs.

3.1 Planning

A classical planning problem is a pair $P = \langle D, I\rangle$, where D represents a first-order *domain* and I contains information specific to the *problem instance* [Ghallab *et al.*, 2004; Geffner and Bonet, 2013; Haslum *et al.*, 2019]. The domain D consists of two components: a set of predicate symbols, and a set of action schemas. The action schemas come with preconditions and effects expressed with atoms $p(x_1, \ldots, x_k)$ where p is a predicate symbol of arity k, and each term x_i is a schema argument. An instance is a tuple $I = \langle O, S_I, G\rangle$, where O represents a set of object names; S_I is the initial state expressed as a set of *ground atoms* $p(o_1, \ldots, o_k)$ where $o_i \in O$ and p is a domain predicate, and G is also a set of ground atoms encoding the goal. A problem P compactly defines a transition system over a finite set of states.

A *generalized policy* π for a class $\mathcal{Q}$ of planning instances over the same domain D represents a collection of state transitions (S, S') in each instance P of $\mathcal{Q}$ that are said to be in π. A π-trajectory is a sequence of states $S_0, \ldots, S_n$ that starts in the initial state of P and whose transitions (S_i, S_{i+1}) are all in π. The trajectory is maximal if S_n is the first goal state of the sequence or there is no transition (S_n, S) in π. The policy π solves P if all maximal π-trajectories in P reach the goal, and it solves $\mathcal{Q}$ if it solves each P in $\mathcal{Q}$. A generalized policy π can be represented in many forms from formulas or rules to general value functions $V(S)$, where the resulting state transitions (S, S') are those with a minimum $V(S')$ value.

3.2 Weisfeiler-Leman Algorithms

The Weisfeiler-Leman (WL) algorithms iteratively color each vertex of a graph, or each k-tuple of vertices, based on the colors of their neighbors, until a fixed point is reached. Colors, represented as natural numbers, are generated with a RELABEL function that maps structures over colors into unique colors. Within GNNs, colors are transformed into real vector embeddings, and multisets of colors are aggregated by permutation-invariant functions like max or sum. We borrow notation and terminology from Morris *et al.* [2023].

1-Dimensional WL (1-WL). For a graph $G = (V, E)$, the node coloring C^1_{i+1} at iteration i is defined as:

$$C^1_{i+1}(v) = \textsc{Relabel}\big(\langle C^1_i(v), \{\!\{C^1_i(u) \mid u \in N(v)\}\!\}\rangle\big)$$

where $N(v)$ denotes the neighborhood of node v in G, and all nodes have the same initial color.

Folklore k-Dimensional WL (k-FWL). For a graph G and tuple $\mathbf{v} \in V(G)^k$, the coloring C^k_{i+1} at iteration i is:

$$C^k_{i+1}(\mathbf{v}) = \textsc{Relabel}\big(\langle C^k_i(\mathbf{v}), M_i(\mathbf{v})\rangle\big)$$

where $M_i(\mathbf{v})$ is the multiset

$$M_i(\mathbf{v}) = \{\!\{\big(C^k_i(\phi_1(\mathbf{v}, w)), \ldots, C^k_i(\phi_k(\mathbf{v}, w))\big) \mid w \in V(G)\}\!\}$$

the function $\phi_j(\mathbf{v}, w)$ replaces the j-th component of the tuple $\mathbf{v}$ with the node w, and the initial color for tuple $\mathbf{v}$ is determined by the structure of the subgraph induced by $\mathbf{v}$.

Oblivious k-Dimensional WL (k-OWL). The coloring C^{k*}_{i+1} for the k-OWL variant at iteration i is defined as:

$$C^{k*}_{i+1}(\mathbf{v}) = \textsc{Relabel}\big(\langle C^{k*}_i(\mathbf{v}), M^*_i(\mathbf{v})\rangle\big)$$

where the multiset $M_i(\mathbf{v})$ of vectors in k-FWL is replaced by a k-dim. vector $M^*_i(\mathbf{v})$ of multisets:

$$\big[M^*_i(\mathbf{v})\big]_j = \{\!\{C^{k*}_i(\phi_j(\mathbf{v}, w)) \mid w \in V(G)\}\!\}$$

for $j = 1, 2, \ldots, k$, where $\phi_j(\mathbf{v}, w)$ is the same function as in k-FWL, and the initial coloring $M^*_0(\mathbf{v})$ is also the same.

For a tuple $\mathbf{v}$ of k vertices, there are potentially $k \cdot n$ neighbor-tuples resulting from replacing each j-th component v_j of $\mathbf{v}$ with each of the n nodes w in the graph, for $j = 1, 2, \ldots, k$. The key difference between k-FWL and k-OWL lies in how these $k \cdot n$ tuples are grouped to determine the new color of $\mathbf{v}$. In k-FWL, the tuple $\mathbf{v}$ "sees" a multiset of n vectors, each with k colors, resulting from replacing v_j with w for each j from 1 to k, providing one such vector or "context" for each node w in the graph. In contrast, in k-OWL, the tuple $\mathbf{v}$ "sees" a k-vector whose elements $\mathbf{v}_j$ are multisets of n colors, with the j component v_j of $\mathbf{v}$ replaced by each of the n nodes in the graph. In k-OWL, the "contexts" mentioned above are broken, hence the method is termed "oblivious" WL, as it is oblivious to such contexts.

Cai *et al.* [1992] established that two graphs are indistinguishable by k-FWL if and only if they satisfy the same formulas in the logic C_{k+1}. In terms of expressive power, 1-OWL is equivalent to 2-OWL, and for $k \geq 3$, k-OWL is strictly more expressive than $(k-1)$-OWL. Furthermore, k-OWL has the same expressiveness as $(k-1)$-FWL for $k \geq 2$. More importantly, the discriminative power of C_3 can be achieved either by using 3-OWL over triplets in cubic space, or by using 2-FWL over pairs in quadratic space.

3.3 Graph Neural Networks

GNNs are parametric functions that operate on graphs through aggregate and combination functions, agg_i and comb_i respectively [Scarselli *et al.*, 2009; Gilmer *et al.*, 2017; Hamilton, 2020]. GNNs maintain and update embeddings $f_i(v) \in \mathbb{R}^k$ for each vertex v in a graph G. The process is iteratively performed over L layers, starting with $i = 0$ and initial embeddings $f_0(v)$, while progressing to $f_{i+1}(v)$ as:

$$f_{i+1}(v) = \text{comb}_i\big(f_i(v), \text{agg}_i\big(\{\!\{f_i(w) \mid w \in N_G(v)\}\!\}\big)\big) \quad (1)$$

where $N_G(v)$ is the set of neighboring vertices of v in the graph G. The aggregation function agg_i (e.g., max, sum, or smooth-max) condenses multiple vectors into a single vectore, whereas the combination function comb_i merges pairs of vectors. The function implemented by GNNs is well defined for graphs of any size, and invariant, under (graph) isomorphisms, for permutation-invariant aggregation functions.

GNNs resemble the 1-WL coloring algorithm, with the difference that vertex colors are represented by real vectors (vertex embedding) instead of natural numbers. Additionally, the aggregation and combination functions are parametric, allowing the vertex embeddings $f_i(v)$ to be learnable functions.

Algorithm 1 Relational GNN (R-GNN)

```
1: Input: Set of ground atoms $S$ (state), and set of objects $O$
2: Output: Embeddings $f_L(o)$ for each object $o \in O$
3: Initialize $f_0(o) \sim 0^k$ for each object $o \in O$
4: for $i \in \{0, \ldots, L-1\}$ do
5:     for each atom $q := p(o_1, \ldots, o_m) \in S$ do
6:         $m_{q,o_j} := [\text{MLP}_p(f_i(o_1), \ldots, f_i(o_m))]_j$
7:     end for
8:     for each object $o \in O$ do
9:         $f_{i+1}(o) := f_i(o) + \text{MLP}_U\big(f_i(o), \text{agg}(\{\!\{m_{q,o} \mid o \in q\}\!\})\big)$
10:    end for
11: end for
```

3.4 Relational GNNs (R-GNNs)

GNNs operate over graphs, whereas planning states are relational structures over predicates of varying arities. Our relational GNN (R-GNN) for processing relational structures is inspired by those used for min-CSPs [Toenshoff *et al.*, 2021], and closely follow the ones used for learning general policies [Ståhlberg *et al.*, 2022a]. In our case, vertices correspond to objects, and the objects communicate via the atoms that are true in the state where they are mentioned. This means that instead of passing messages along graph edges as described by (1), messages are passed via atoms as follows:

$$f_{i+1}(o) = \text{comb}_i\big(f_i(o), \text{agg}_i\big(\{\!\{m_{q,o} \mid o \in q, q \in S\}\!\}\big)\big)\,. \quad (2)$$

Here, $m_{q,o}$ for a predicate $q = p(o_1, \ldots, o_m)$ and object $o = o_j$ is the message that atom q sends to object o, defined as

$$m_{q,o_j} = \big[\text{comb}_p\big(f_i(o_1), \ldots, f_i(o_m)\big)\big]_j\,,$$

where $\text{comb}_p(\cdot)$ is the combination function for predicate p that creates m (possibly different) messages, one for each object o_j, from their embeddings $f_i(o_j)$. The $\text{comb}_i(\cdot)$ function merges two vectors of size k, the current embedding $f_i(o)$ and the aggregation of the messages $m_{q,o}$ received at o.

The relational neural network is detailed in Algorithm 1, where the update for the embeddings in (2) is implemented via residual connections. In our implementation, the aggregation function $\text{agg}(\cdot)$ is *smooth maximum* that approximates the (component-wise) maximum. The combination functions are implemented using MLPs. All the functions $\text{comb}_i(\cdot)$ correspond to the same MLP_U that maps two real vectors in $\mathbb{R}^k$ into a vector in $\mathbb{R}^k$, while $\text{comb}_p(\cdot)$, for a predicate p of arity m, is an MLP_p that maps m vectors in $\mathbb{R}^k$ into m vectors in $\mathbb{R}^k$. In all cases, each MLP has three parts: first, a linear layer; next, the Mish activation function [Misra, 2020]; and then another linear layer. The architecture in Algorithm 1 requires two inputs: a set of atoms denoted as S, and a set of objects denoted as O. The goal G is encoded by *goal atoms* that are assumed to be in S: if $p(o_1, \ldots, o_m)$ is an atom in G, the atom $p_g(o_1, \ldots, o_m)$ is added to S, where p_g is a new "goal predicate" [Martín and Geffner, 2004].

The set of object embeddings $f_L(o)$ at the last layer of the R-GNN is the result of the net; i.e., $\text{R-GNN}(S, O) = \{f_L(o) \mid o \in O\}$. Such embeddings are used to encode general value functions, policies, or both. In this work, we encode a learnable value function $V(S)$ through a simple additive readout that feeds the embeddings into an MLP:

$$V(S) = \text{MLP}\big(\textstyle\sum_{o \in O} f_L(o)\big)\,.$$

4 Parametric Extended R-GNN: R-GNN[t]

The new architecture extends the expressive power of R-GNNs beyond C_2 by capitalizing the relational component of R-GNNs, outlined in Algorithm 1. The function computed by the new architecture, R-GNN[t](S, O), where t is a non-negative integer parameter, is defined as:

$$\text{R-GNN}[t](S, O) = \text{R-GNN}(A_t(S), O^2) \tag{3}$$

where $O^2 = O \times O$ stands for the pairs $\langle o, o' \rangle$ of objects in O, and $A_t(S)$ stands for a transformation of the atoms in S that depends on t. Specifically, if $w = \langle o_1, \ldots, o_m \rangle$ is a tuple, $\langle w \rangle^2$ refers to the tuple of m^2 pairs obtained from w as:

$$\langle w \rangle^2 = \langle (o_1, o_1), \ldots, (o_1, o_m), \ldots, (o_m, o_1), \ldots, (o_m, o_m) \rangle$$

Then, for $t = 0$, the set of atoms $A_t(S)$ is:

$$A_0(S) = \{ p(\langle w \rangle^2) \mid p(w) \in S \} .$$

That is, predicates p of arity m in S transform into predicates p of arity m^2 in $A_0(S)$, and each atom $p(w)$ in S is mapped to the atom $p(\langle w \rangle^2)$.

For $t > 0$, the set of atoms $A_t(S)$ extends the atoms in $A_0(S)$ with atoms for a new ternary "compose" predicate $\triangle$ as: $A_t(S) = A_0(S) \cup \Delta_t(S)$ where

$$\Delta_t(S) = \left\{ \triangle(\langle o, o' \rangle, \langle o', o'' \rangle, \langle o, o'' \rangle) \mid \langle o, o' \rangle, \langle o', o'' \rangle \in R_t \right\} ,$$

and the binary relation R_t is defined from S and G as:

$$\langle o, o' \rangle \in R_t \text{ iff } \begin{cases} o \text{ and } o' \text{ are both in an atom in } S & t = 1 , \\ \langle o, o'' \rangle \text{ and } \langle o'', o' \rangle \text{ are in } R_{t-1} & t > 1 . \end{cases}$$

In words, for the R-GNN to emulate a relational version of 2-FWL, two things are needed. First, object pairs need to be embedded. This is achieved by replacing the atoms in S with the atoms in $A_0(S)$, which are object pairs. Second, object pairs $\langle o, o' \rangle$ need to receive and aggregate messages from object triplets, the "contexts" in 2-FWL, which are formed by vectors of pairs $\langle o, o'' \rangle$ and $\langle o'', o' \rangle$. This interaction is captured through the new atoms $\triangle(\langle o, o'' \rangle, \langle o'', o' \rangle, \langle o, o' \rangle)$ and the associated MLP$_\triangle$. The relational GNN architecture in Algorithm 1 allows each argument to communicate with every other argument in the context of a third one, with messages that depend on all the arguments. This is similar to the triangulation found in edge transformers [Bergen *et al.*, 2021]. Rather than adding all possible $\triangle(\langle o, o'' \rangle, \langle o'', o' \rangle, \langle o, o' \rangle)$ atoms to $A_0(S)$, the atoms are added in a controlled manner using the parameter t to avoid a cubic number of messages to be exchanged. The parameter t controls the maximum number of sequential compositions that can be captured. In problems that require a single composition, a value of $t = 1$ suffices to yield the necessary C_3 features without having to specify which relations need to be composed. Moreover, all this is achieved without altering the plain R-GNN architecture, by simply changing the inputs from S and O to $A_t(S)$ and O^2, respectively, as expressed in (3).

As before, the final embeddings are used to define a general value function $V(S)$:

$$V(S) = \text{MLP}\big(\textstyle\sum_{o \in O} f_L(\langle o, o \rangle)\big) .$$

However, the readout only takes the final embeddings for object pairs $\langle o, o \rangle$ that represent single objects, and passes their sum to an MLP that outputs the scalar $V(s)$. The reason is to avoid summing the embeddings for all pairs $\langle o, o' \rangle$ as it leads to high variance, and a more difficult learning.

Since R-GNN[t] is a regular R-GNN over a transformed input, the objects in a R-GNN are indistinguishable a priori, and the readout function only considers pairs of type $\langle o, o \rangle$, some way to make such pairs different from others must be incorporated so that the message passing mechanism learns where to send the information for the readout. This is automatically achieved by adding static atoms $\textsc{Obj}(o)$, for each OBJect o, for a new static unary predicate $\textsc{Obj}$. Such atoms are then transformed into atoms $\textsc{Obj}(\langle o, o \rangle)$ in $A_0(S)$ which then mark the pairs $\langle o, o \rangle$ as different from other pairs $\langle o, o' \rangle$.

5 Learning Task

Since the aim of this paper is to extend the expressivity of R-GNNs, the general value function V is learned in a supervised manner from the optimal value function V^* using a small collection of training instances from given domains. These instances have small state spaces, so computing V^* is simple. The loss function, which is minimized to train V, is:

$$L(S) = |V^*(S) - V(S)| .$$

When we create a batch, we try to sample batches with as many distinct V^* values as possible. The state spaces of the training instances are fully expanded, thus we know the optimal cost from every state.

6 Baselines: R-GNN and Edge Transformers

The baselines for evaluation are the basic R-GNN architecture and edge transformers [Bergen *et al.*, 2021]. ETs and R-GNN[t] share a close similarity. In both, expressivity beyond C_2 is achieved through a "triangulation" designed to mirror the 2-FWL procedure. In ETs, the triangulation occurs via a version of self-attention [Vaswani *et al.*, 2017], whereas in R-GNN[t], it occurs via the composition atoms in Δ_t.

In ETs, like in 2-GNNs, each object pair communicates with every overlapping pair. Thus, information about true atoms in a state S must be included in the initial embeddings. For this reason, we restrict the use of ETs to domains where all predicates are *binary*, interpreting unary predicates as binary by repeating the atom's first term. In addition, for each predicate p, two *learnable* d-dimensional vectors, E_p and E_{p_g}, are used to define the initial embeddings $E_{o,o'}$:

$$E_{o,o'} = \textstyle\sum_p \big(E_p \cdot [\![p(o, o') \in S]\!] + E_{p_g} \cdot [\![p(o, o') \in G]\!] \big)$$

where $[\![\cdot]\!]$ denotes the Iverson bracket. The initial embeddings depend on the learned vectors accurately representing the state and the goal in a way that is suitable for self-attention. For the value function, we use the same readout function as for R-GNN[t] that aggregates the final embeddings of pairs with identical objects and feeds the resultant vector into an MLP. Notice that the number of embeddings aggregated in the final readout is the same for R-GNN, R-GNN[t], and ET.

7 Example: Grid Navigation with Obstacles

We illustrate the expressivity demands on a simple example and how these demands are met by the different architectures.

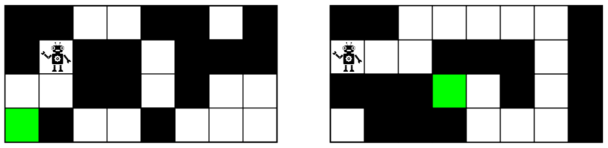

Figure 1: Two 8×4 test instances of the Navig-xy domain where the robot has to reach the green cell in a grid with blocked cells, and where the objects are the values of each one of the two coordinates, and not the cells themselves. The problem is not in C_2 in this representation, and indeed, after training, the baseline R-GNN solves the instance on the left but not the one on the right, while R-GNN[t], for $t = 1$, solves all the instances in the test set.

We call the domain Navig-xy and two instances are shown in Figure 1. In this domain, a robot has to reach the goal (green) cell in a $n \times m$ grid with blocked cells, that is represented with $n + m$ objects $X \cup Y$, where $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_m\}$, and successor relations SUCC-X and SUCC-Y. The domain also includes the binary relations $\text{AT}(x, y)$ to specify the initial and goal cells, $\text{BLOCKED}(x, y)$ to specify the cells that are blocked, and a dummy $\text{CELL}(x, y)$ to identify the cells in the grid; $x \in X$ and $y \in Y$.

Under the settings described in the next section and 12 hours of training over 105 random $n \times m$ solvable instances, with $nm < 30$, policies strictly greedy in the learned value function V achieve coverages of 59.72%, 80.55%, and 100% for the baseline R-GNN, R-GNN[0], and R-GNN[1], respectively, on instances with different sets of blocked cells and up to a slightly larger size $nm \leq 32$. The ET performs poorly and achieves 4.16% of coverage. The instances are not difficult as the there is just one free path to the goal on which the robot just has to keep moving forward, but when the value function is wrong, it can drive the robot backwards creating a cycle.

The explanation for the coverage results is simple. For computing the true distance to the goal in these grids, emulating a shortest path algorithm, each cell (x, y) in the grid must be able to communicate with each of its neighbor cells (x, y') and (x', y). In the R-GNN architecture captured by Alg. 1, this means that there must be atoms involving the three objects x, y, and y', and similarly, x, x', and y. There are no such atoms in the state S, except in R-GNN[t], for $t \geq 1$, where $A_t(S)$ includes the composition atoms $\triangle((x, x'), (x', y), (x, y))$ and $\triangle((x, y), (y, y'), (x, y'))$. As a result, R-GNN[1] and R-GNN[2] can compute the true distances, while R-GNN[0] and (perhaps) R-GNN can only compute "Manhattan distances", in general, that in some cases are good approximations of $V^*(s)$.

8 Domains

Brief descriptions of the domains for the experiments, mostly taken from Ståhlberg *et al.* [2022a; 2022b; 2023], follow:

Blocks. In Blocks-s (resp. Blocks-m), a single tower (resp. multiple towers) must be built. Both have training and validation sets with 4 to 9 blocks. The test set for Blocks-s (resp. Blocks-m) has 10 to 17 blocks (resp. up to 20 blocks).

Grid. The goal is to fetch keys and unlock doors to reach a cell. A generator creates random instances with given (room) layouts. Test instances usually have more keys and locks than those for training and validation, have different layouts, and their state spaces are too big to be fully expanded.

Gripper. A robot with two grippers must move balls from one to another room. The training and validation instances have up to 14 balls, while test instances have 16-50 balls.

Logistics. Transportation domain with packages, cities, trucks, and one airplane. Training and validation instances have 2-5 cities and 3-5 packages, while testing instances have 15-19 cities and 8-11 packages.

Miconic. An elevator must pick and deliver passengers at different floors. Training and validation instances involve 2-20 floors and 1-10 passengers, while those for testing contain 11-30 floors and 22-60 passengers.

Rovers. The domain simulates planetary missions where a rover must navigate among waypoints to collect soil/rock samples, take pictures, and send information back to base. Training and validation instances use 2-3 rovers and 3-8 waypoints; those for testing have 3 rovers and 21-39 waypoints.

Vacuum. Robot vacuum cleaners that move around and clean different locations. The robots have their own traversal map, so some robots can go between two locations while others cannot. In our version, there is a single dirty location in the middle. The training and validation sets involve 8-38 locations and 1-6 robots. The test set includes 40-93 locations and 6-10 robots.

Visitall. A robot must visit multiple cells in a grid without obstacles. In Visitall-xy, the grid is described with coordinates as in the Navig-xy domain, while in Visitall there is an object for each cell in the grid. Both versions come with training and validation sets with up to 21 locations, while the test set includes strictly more, with a maximum of 100 cells.

9 Experiments

A learned value function V, for a given domain, defines a general policy π_V that at state s selects an *unvisited* successor state s' with lowest $V(s')$ value. We test such policies on instances until reaching a goal state, executing 1000 steps, or reaching a state has no unvisited successors; where reaching a goal is counted as a success, or else as a failure.

For learning the value functions, we implemented the architectures in PyTorch, and trained the models on NVIDIA A10 GPUs with 24 GB of memory over 12 hours, using Adam [Kingma and Ba, 2015] with a learning rate of 0.0002, batches of size 16, and without applying any regularization loss. For each specific domain, a total of three models were trained, and the model with the lowest loss on the validation set was selected as the final model. We used 30 layers for both the relational NNs and the ETs, all layers share weights, and the ETs have 8 self-attention heads.

Tables 1 and 2 show the results. We anticipate that the improved expressivity of the networks will result in one or more of the following outcomes:

- Maintaining performance levels on C_2 domains; and

			Plan Length		
Domain	Model	Coverage (%)	Total	Median	Mean
Blocks-s	R-GNN	**17 / 17 (100 %)**	674	38	39
	R-GNN[0]	**17 / 17 (100 %)**	670	36	39
	R-GNN[1]	**17 / 17 (100 %)**	684	36	40
	ET	16 / 17 (94 %)	826	38	51
Blocks-m	R-GNN	**22 / 22 (100 %)**	868	40	39
	R-GNN[0]	**22 / 22 (100 %)**	830	39	37
	R-GNN[1]	**22 / 22 (100 %)**	834	39	37
	ET	18 / 22 (82 %)	966	39	53
Gripper	R-GNN	**18 / 18 (100 %)**	4800	231	266
	R-GNN[0]	**18 / 18 (100 %)**	1764	98	98
	R-GNN[1]	11 / 18 (61 %)	847	77	77
	ET	4 / 18 (22 %)	246	61	61
Miconic	R-GNN	**20 / 20 (100 %)**	1342	67	67
	R-GNN[0]	**20 / 20 (100 %)**	1566	71	78
	R-GNN[1]	**20 / 20 (100 %)**	2576	71	128
	ET	**20 / 20 (100 %)**	1368	68	68
Visitall	R-GNN	18 / 22 (82 %)	636	29	35
	R-GNN[0]	21 / 22 (95 %)	1128	35	53
	R-GNN[1]	**22 / 22 (100 %)**	886	35	40
	ET	18 / 22 (82 %)	670	29	37

Table 1: Coverage and plan lengths for C_2 domains.

			Plan Length		
Domain	Model	Coverage (%)	Total	Median	Mean
Grid	R-GNN	9 / 20 (45 %)	109	11	12
	R-GNN[0]	12 / 20 (60 %)	177	11	14
	R-GNN[1]	**15 / 20 (75 %)**	209	13	13
	ET	1 / 20 (5 %)	15	15	15
Logistics	R-GNN	10 / 20 (50 %)	510	51	51
	R-GNN[0]	9 / 20 (45 %)	439	48	48
	R-GNN[1]	**20 / 20 (100 %)**	1057	52	52
	ET	0 / 20 (0 %)	-	-	-
Rovers	R-GNN	9 / 20 (45 %)	2599	280	288
	R-GNN[0]	**14 / 20 (70 %)**	2418	153	172
	R-GNN[1]	**14 / 20 (70 %)**	1654	55	118
	ET	Unsuitable domain: ternary predicates			
Vacuum	R-GNN	**20 / 20 (100 %)**	4317	141	215
	R-GNN[0]	**20 / 20 (100 %)**	183	9	9
	R-GNN[1]	**20 / 20 (100 %)**	192	9	9
	ET	Unsuitable domain: ternary predicates			
Visitall-xy	R-GNN	5 / 20 (25 %)	893	166	178
	R-GNN[0]	15 / 20 (75 %)	1461	84	97
	R-GNN[1]	**20 / 20 (100 %)**	1829	83	91
	ET	3 / 20 (15 %)	455	138	151

Table 2: Coverage and plan lengths for C_3 domains.

- Achieving broader coverage on C_{3+} domains, or
- Generating plans of superior quality.

As shown in tables, our anticipations were mostly matched. The coverage results for C_2 domains, shown in Table 1, are maintained, with the plan quality remaining largely unchanged. However, increasing t in R-GNN[t] sometimes results in a drop in coverage. We attribute this outcome to two factors: firstly, such increased values slow down the model, leading to less training and potentially incomplete convergence; secondly, the volume of messages may hinder the ability of the smooth maximum to approximate the maximum. Overall, R-GNN[t] is on par with R-GNN over C_2 domains.

Regarding C_{3+} domains, Table 2, we observe a significant improvement across the board. In all such domains, except Vacuum, there is a significant increase in coverage, which we discuss next in detail. For Vacuum, we observe a significant improvement in plan quality.

9.1 Analysis for $\mathbf{C_3}$ Domains

We now take a closer look at the C_3 domains.

Grid

In the Grid domain, the objective is to go to a specific cell on a grid. However, access to this cell is obstructed by locked doors, each requiring a key of a specific shape. Moreover, one locked door may be behind another, creating a sequence of barriers. The placement of keys within the rooms is such that it ensures the overall instance is solvable.

This domain includes, among others, the binary predicates KEY-SHAPE$/2$ and LOCK-SHAPE$/2$. For example, consider a key k with shape s, as denoted by KEY-SHAPE(k, s), and a lock l, also with shape s, indicated by LOCK-SHAPE(l, s). In this case, they match, allowing the key to be used to open the lock. We believe that this domain requires C_3 expressiveness, as the feature CAN-UNLOCK(k, l), indicating that k can open l, is likely a necessary composition (which requires C_3). Additionally, this feature should be associated with and ”stored” in the embedding for the pair $\langle k, l\rangle$. It cannot be associated solely with k, as there are an arbitrary number of locks, nor can it be associated with l for the same reason. Thus, with an embedding for every pair, the capacity for storing the required features scales appropriately with the number of objects.

We believe these observations explain our experimental results: R-GNN[0] outperforms B because it can appropriately store an approximation of the composition. Similarly, R-GNN[1] outperforms R-GNN[0] by actually computing the composition instead of approximating it. Nonetheless, we did not achieve 100% coverage. Upon inspecting the 5 unsolved instances, we observed that the lengths of the *optimal* plans varied between 25 and 44 steps. As noted by [Ståhlberg *et al.*, 2022a], the network's capability to compute distances is bounded by the number of layers. The plans primarily involve moving to specific cells to collect keys and unlock doors. In our experiments, we used 30 layers, and the lengths of the optimal plans likely exceeded the network's reasoning capacity (rather than expressive capacity). We note that, when a plan is found, it tends to be near-optimal. The mean and median values for optimal plans of the solved instances are both 13, mirroring those for R-GNN[1].

Logistics

In Logistics, as noted by [Ståhlberg *et al.*, 2022b; 2023], the delivery of multiple packages requires C_3 expressiveness to determine if an airplane is in the correct city for a package in its cargo. In their experiments, they achieved 100% coverage by introducing *derived predicates*, which are defined by *composing* binary relations. More specifically, in [Ståhlberg *et al.*, 2022b], they provided derived predicates for the compositions: $\textsc{At} \circ \textsc{In-City}$, $\textsc{At}_g \circ \textsc{In-City}$, $\textsc{In} \circ \textsc{At}$, and $\textsc{In} \circ \textsc{At} \circ \textsc{In-City}$. The hyperparameter t in R-GNN[t], corresponds to the number of compositions performed recursively through the new ternary predicate. In our experiments, it seems that only one composition is needed to learn a value function, while they used two. We emphasize that our compositions were learned automatically, in contrast to their handcrafted approach, which is a significant difference.

Vacuum

Vacuum is a simplified version of Rovers, and was designed to illustrate that C_2 expressiveness is insufficient for learning an optimal value function for Rovers [Ståhlberg *et al.*, 2022a]. The domain has a ternary predicate $\textsc{Adjacent}(r, x, y)$, signifying that robot r can move from location x to location y in a single action. Each robot operates within its unique traversal map, and the objective is to clean a specific dirty location g. The optimal value function involves computing the distance from each robot's current location to g and directing the nearest robot towards g. While the baseline achieved 100% coverage, it did so with excessively long plan lengths. This suggests that the model struggles to reliably identify good successor states, and it seems more coincidental than intentional when the model directs a robot to the goal location that requires cleaning.

The existence of a path of length k from a robot r at a location x to the dirty goal location is defined by the equations:

$$\mathrm{P}_0(r, x) = \textsc{Dirty}(x)$$
$$\mathrm{P}_k(r, x) = \exists y : [\textsc{Adjacent}(r, x, y) \land \mathrm{P}_{k-1}(r, y)]$$

Here, we need three variables in the second equation, placing it in C_3. For the baseline, a practical limitation arises from the inability to store computed distances from a location x to g for robot r. We need to store not only the final distance, but also intermediate distances to compute the final value. However, storing these distances in the embedding for each robot is problematic due to an arbitrary number of locations. Similarly, storing distances in the embedding for each location is also not possible due to an arbitrary number of robots. In contrast, the other methods can store the distances at the robot-location pair $\langle r, x\rangle$. Furthermore, this pair, together with $\langle r, y\rangle$, are specifically referenced in the extended predicate of $\textsc{Adjacent}$. This extended predicate and the pair embeddings are sufficient to compute all shortest paths, and we believe this explains the performance difference between the baseline and extended versions.

Rovers

Vacuum is essentially a subproblem of Rovers. Each rover operates within its own traversal map, defined by the predicate $\textsc{Can-Traverse}/3$, necessitating at least C_3 expressiveness. However, after data collection, it must also be wirelessly communicated back to the lander, provided the lander is visible from the current location. That is, the existence of a path with total length $k_1 + k_2$ from a rover r at location x to a point of interest (POI), where an action is performed, and then back to a location for data transmission, is expressed as:

$$\mathrm{P}^1_{0,k_2}(r, x) = [\mathrm{POI}(x) \lor \textsc{Has-Sample}(r)] \land \mathrm{P}^2_{k_2}(r, x)$$
$$\mathrm{P}^1_{k_1,k_2}(r, x) = \exists y : [\textsc{Can-Trav.}(r, x, y) \land \mathrm{P}_{k-1,k_2}(r, y)]$$
$$\mathrm{P}^2_0(r, x) = \exists y : [\textsc{Lander-Loc.}(y) \land \textsc{Visible}(x, y)]$$
$$\mathrm{P}^2_{k_2}(r, x) = \exists y : \left[\textsc{Can-Trav.}(r, x, y) \land \mathrm{P}^2_{k_2-1}(r, y)\right]$$

Note that in the base case for the first path, P^1, there are two criteria: the point of interest, or if the rover already has the sample. In the latter case, the rover, having visited the POI and performed the necessary action, only needs to reach a location for data transmission to the lander. If predicates not part of the state, i.e., POI, $\textsc{Has-Sample}$, and $\textsc{Lander-Location}$, fall within C_3, then this subproblem also remains in C_3.

Rovers may need to sample at most 3 POIs. Given the fixed number of POIs, it is possible to derive equations that follow the same pattern, dividing the path into up to 4 segments. This feature is quite complex and presents a challenging optimization problem: directing rovers to minimize the total length of the plan. To test whether the poor quality of plans is due to the optimization problem, we tried a version with just one rover. This resulted in 95% coverage with plans that were very close to optimal. Therefore, we believe the difficulty for the model lies in addressing the optimization problem, not in the complexity of the feature itself.

Visitall-xy

In Section 7, a similar domain was explored. The key difference in this domain is the absence of obstacles, with the added goal of visiting multiple locations. Nevertheless, we believe that the analysis from Section 7 is also applicable here. Specifically, for $t \geq 1$, the $\triangle$ relation allows for direct communication between adjacent cells.

10 Conclusions

The paper presents a novel approach for extending the expressive power of Relational Graph Neural Networks (R-GNNs) in the classical planning setting by just adding a set of atoms A_t to the state, in a domain-independent manner that depends on the t parameter and the pairs of objects that interact in an atom true in the state. The resulting "architecture" R-GNN[t] appears to produce the necessary C_3 features in a practical manner without the memory and time overhead of 3-GNNs and with much better generalization than edge-transformers, as shown in the experimental results and in the analysis. It is not clear if the same architecture can yield similar payoffs in standard GNN benchmarks where the state does not change and where there is a single relation that is binary. Looking ahead, we plan to explore whether learning *suboptimal* policies [Ståhlberg *et al.*, 2023] could be beneficial in domains such as Grid and Rovers. In the Grid domain, the value function has to consider the entire plan and is sometimes limited by the number of layers. Nonetheless, this limitation might be circumvented to some extent if the *next* subgoal can be identified, for instance, locating a key with a specific shape. This approach might also solve the issue in Rovers, where the main challenge seems to be optimizing what each rover should do, by greedily using the closest rover.

Acknowledgements

The research of H. Geffner has been supported by the Alexander von Humboldt Foundation with funds from the Federal Ministry for Education and Research. The research has also received funding from the European Research Council (ERC), Grant agreement No. 885107, and Project TAILOR, Grant agreement No. 952215, under EU Horizon 2020 research and innovation programme, the Excellence Strategy of the Federal Government and the NRW Länder, and the Knut and Alice Wallenberg (KAW) Foundation under the WASP program. The computations were enabled by the supercomputing resource Berzelius provided by National Supercomputer Centre at Linköping University and the Knut and Alice Wallenberg foundation.

References

[Baader *et al.*, 2003] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[Bajpai *et al.*, 2018] A. N. Bajpai, S. Garg, et al. Transfer of deep reactive policies for mdp planning. In *Proc. of the 32nd Annual Conference on Neural Information Processing Systems (NeurIPS 2018)*, pages 10965–10975, 2018.

[Barceló *et al.*, 2020] P. Barceló, E. Kostylev, M. Monet, J. Pérez, J. Reutter, and J.-P. Silva. The logical expressiveness of graph neural networks. In *Proc. of the 8th International Conference on Learning Representations (ICLR 2020)*. OpenReview.net, 2020.

[Barcelo *et al.*, 2022] P. Barcelo, M. Galkin, C. Morris, and M. R. Orth. Weisfeiler and leman go relational. In *Learning on Graphs Conference*, pages 46–1, 2022.

[Belle and Levesque, 2016] V. Belle and H. J. Levesque. Foundations for generalized planning in unbounded stochastic domains. In *Proc. of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR 2016)*, pages 380–389, 2016.

[Bergen *et al.*, 2021] L. Bergen, T. J. O'Donnell, and D. Bahdanau. Systematic generalization with edge transformers. In *Proc. of the 35th Annual Conference on Neural Information Processing Systems (NeurIPS 2021)*, pages 1390–1402, 2021.

[Bertsekas, 1995] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[Bonet and Geffner, 2018] B. Bonet and H. Geffner. Features, projections, and representation change for generalized planning. In *Proc. of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 4667–4673. IJCAI, 2018.

[Bonet *et al.*, 2019] B. Bonet, G. Francès, and H. Geffner. Learning features and abstract actions for computing generalized plans. In *Proc. of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, pages 2703–2710. AAAI Press, 2019.

[Boutilier *et al.*, 2001] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 690–700. Morgan Kaufmann, 2001.

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

[Chevalier-Boisvert *et al.*, 2019] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *Proc. of the 7th International Conference on Learning Representations (ICLR 2019)*. OpenReview.net, 2019.

[Drexler *et al.*, 2022] D. Drexler, J. Seipp, and H. Geffner. Learning sketches for decomposing planning problems into subproblems of bounded width. In *Proc. of the 32nd International Conference on Automated Planning and Scheduling (ICAPS 2022)*, pages 62–70. AAAI Press, 2022.

[Fern *et al.*, 2006] A. Fern, S. Yoon, and R. Givan. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of Artificial Intelligence Research*, 25:75–118, 2006.

[Francès *et al.*, 2021] G. Francès, B. Bonet, and H. Geffner. Learning general planning policies from small examples without supervision. In *Proc. of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021)*, pages 11801–11808. AAAI Press, 2021.

[François-Lavet *et al.*, 2018] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 2018.

[Geffner and Bonet, 2013] H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*, volume 7 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool, 2013.

[Ghallab *et al.*, 2004] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[Gilmer *et al.*, 2017] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *Proc. of the 34th International Conference on Machine Learning (ICML 2017)*, pages 1263–1272. JMLR.org, 2017.

[Grohe, 2021] M. Grohe. The logic of graph neural networks. In *Proceedings of the Thirty-Sixth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021)*, pages 1–17, 2021.

[Groshev *et al.*, 2018] E. Groshev, M. Goldstein, A. Tamar, S. Srivastava, and P. Abbeel. Learning generalized reactive policies using deep neural networks. In *Proc. of the 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pages 408–416. AAAI Press, 2018.

[Hamilton, 2020] W. Hamilton. *Graph Representation Learning*, volume 14 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool, 2020.

[Haslum *et al.*, 2019] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise. *An Introduction to the Planning Domain Definition Language*, volume 13 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool, 2019.

[Hu and Giacomo, 2011] Y. Hu and G. D. Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *Proc. of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 918–923. AAAI Press, 2011.

[Illanes and McIlraith, 2019] L. Illanes and S. A. McIlraith. Generalized planning via abstraction: Arbitrary numbers of objects. In *Proc. of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, pages 7610–7618. AAAI Press, 2019.

[Jiménez *et al.*, 2019] S. Jiménez, J. Segovia-Aguas, and A. Jonsson. A review of generalized planning. *The Knowledge Engineering Review*, 34:e5, 2019.

[Khardon, 1999] R. Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113:125–148, 1999.

[Kingma and Ba, 2015] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. of the 3rd International Conference on Learning Representations (ICLR 2015)*, 2015.

[Kirk *et al.*, 2023] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023.

[Martín and Geffner, 2004] M. Martín and H. Geffner. Learning generalized policies from planning examples using concept languages. *Applied Intelligence*, 20(1):9–19, 2004.

[Misra, 2020] D. Misra. Mish: A self regularized non-monotonic activation function. In *Proceedings of the 31st British Machine Vision Conference (BMVC 2020)*. BMVA Press, 2020.

[Morris *et al.*, 2019] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proc. of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, pages 4602–4609. AAAI Press, 2019.

[Morris *et al.*, 2023] C. Morris, Y. Lipman, H. Maron, B. Rieck, N. M. Kriege, M. Grohe, M. Fey, and K. Borgwardt. Weisfeiler and leman go machine learning: The story so far. *Journal of Machine Learning Research*, 24(333):1–59, 2023.

[Rivlin *et al.*, 2020] O. Rivlin, T. Hazan, and E. Karpas. Generalized planning with deep reinforcement learning. In *ICAPS 2020 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*, pages 16–24, 2020.

[Sanner and Boutilier, 2009] S. Sanner and C. Boutilier. Practical solution techniques for first-order MDPs. *Artificial Intelligence*, 173(5-6):748–788, 2009.

[Scarselli *et al.*, 2009] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[Schlichtkrull *et al.*, 2018] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, pages 593–607. Springer, 2018.

[Srivastava *et al.*, 2008] S. Srivastava, N. Immerman, and S. Zilberstein. Learning generalized plans using abstract counting. In *Proc. of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 991–997. AAAI Press, 2008.

[Srivastava *et al.*, 2011] S. Srivastava, N. Immerman, and S. Zilberstein. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2):393–401, 2011.

[Ståhlberg *et al.*, 2022a] S. Ståhlberg, B. Bonet, and H. Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *Proc. of the 32nd International Conference on Automated Planning and Scheduling (ICAPS 2022)*, pages 629–637. AAAI Press, 2022.

[Ståhlberg *et al.*, 2022b] S. Ståhlberg, B. Bonet, and H. Geffner. Learning generalized policies without supervision using GNNs. In *Proc. of the 19th International Conference on Principles of Knowledge Representation and Reasoning (KR 2022)*, pages 474–483. IJCAI Organization, 2022.

[Ståhlberg *et al.*, 2023] S. Ståhlberg, B. Bonet, and H. Geffner. Learning general policies with policy gradient methods. In *Proc. of the 20th International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*. IJCAI Organization, 2023.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[Toenshoff *et al.*, 2021] J. Toenshoff, M. Ritzert, H. Wolf, and M. Grohe. Graph neural networks for maximum constraint satisfaction. *Frontiers in Artificial Intelligence and Applications*, 3:580607, 2021.

[Toyer *et al.*, 2020] S. Toyer, S. Thiébaux, F. Trevizan, and L. Xie. ASNets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68:1–68, 2020.

[Vashishth *et al.*, 2019] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar. Composition-based multi-relational graph convolutional networks. In *Proc. of the 7th International Conference on Learning Representations (ICLR 2019)*. OpenReview.net, 2019.

[Vaswani *et al.*, 2017] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proc. of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*. Curran Associates, Inc., 2017.

[Wang *et al.*, 2008] C. Wang, S. Joshi, and R. Khardon. First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research*, 31:431–472, 2008.

[Xu *et al.*, 2019] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *Proc. of the 7th International Conference on Learning Representations (ICLR 2019)*. OpenReview.net, 2019.