Learning More Expressive General Policies for Classical Planning Domains

Simon Ståhlberg[1], **Blai Bonet**[2], **Hector Geffner**[1]

[1]RWTH Aachen University, Germany
[2]Universitat Pompeu Fabra, Spain
simon.stahlberg@gmail.com, 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 overcame 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 to store embeddings, rendering them infeasible in practice. In this work, we introduce a parameterized version R-GNN[t] (with parameter t) of Relational GNNs. Unlike GNNs, that are designed to perform computation on graphs, Relational GNNs are designed to do computation on relational structures. When $t = \infty$, R-GNN[t] approximates 3-GNNs over graphs, but 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 expressivity 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 inputs only. Experimental results illustrate the clear performance gains of R-GNN[1] over the plain R-GNNs, and also over Edge Transformers that also approximate 3-GNNs.

Introduction

General policies are policies that can be used to solve a collection of planning problems reactively (Srivastava, Immerman, and Zilberstein 2008; Hu and Giacomo 2011; Belle and Levesque 2016; Bonet and Geffner 2018; Illanes and McIlraith 2019; Jiménez, Segovia-Aguas, and Jonsson 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, Hazan, and Karpas 2020; Ståhlberg, Bonet, and Geffner 2022a), one key expressive limitation results from the type 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, Francès, and Geffner 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, Bonet, and Geffner 2022b, 2023).

In principle, this limitation can be addressed by using richer grammars to generate non-C_2 features, in the logical setting, or by using k-GNNs, with $k = 3$, on the neural setting, 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). R-GNNs are designed to perform computation over relational structures, unlike GNNs that can only process graphs. The architecture for R-GNN[t] mirrors that of plain R-GNNs and differs only in the input. While a plain R-GNNs takes 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 four baselines: plain R-GNN, 2-GNN, R-GNN$_2$, and Edge-Transformers (Bergen, O'Donnell, and Bahdanau 2021), where the last two aim to approximate 3-GNNs (Bergen, O'Donnell, and Bahdanau 2021) more accurately than R-GNN[t].

The rest of the paper is organized as follows. We review first related work and background on planning, generalized planning, GNNs and relational GNNs, and the Weisfeiler-Leman coloring algorithms. Then we introduce the parametric R-GNNs, the learning task and baselines, and the experimental results. The paper ends with a discussion on the expressivity of the model, and conclusions.

Related Work

General policies from logic. The problem of learning general policies has a long history (Khardon 1999; Martín and Geffner 2004; Fern, Yoon, and Givan 2006), and general policies have been formulated in terms of logic (Srivastava, Immerman, and Zilberstein 2011; Illanes and McIlraith 2019), regression (Boutilier, Reiter, and Price 2001; Wang, Joshi, and Khardon 2008; Sanner and Boutilier 2009), and policy rules (Bonet and Geffner 2018; Bonet, Francès, and Geffner 2019) that can be learned (Francès, Bonet, and Geffner 2021; Drexler, Seipp, and Geffner 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, Garg et al. 2018; Rivlin, Hazan, and Karpas 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, Bonet, and Geffner, 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 relational structures (Schlichtkrull et al. 2018; Vashishth et al. 2019; Barcelo et al. 2022), and our R-GNNs borrow from those used for max-CSP (Toenshoff et al. 2021) and generalized planning (Ståhlberg, Bonet, and Geffner 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, Fürer, and Immerman 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, Fürer, and Immerman 1992), is more efficient as it has the power of $(k{+}1)$-WL while using $\mathcal{O}(n^k)$ memory. Maron et al. (2019b) define a parameterized family of permutation-invariant neural networks, which for $k = 2$ has the expressiveness of 3-GNNs (Maron et al. 2019a). In the experiments, we consider a baseline based on Edge Transformers that has the same expressiveness (Bergen, O'Donnell, and Bahdanau 2021).

Background

We review planning, generalized planning, GNNs, relational GNNs, and the Weisfeiler-Leman graph coloring algorithms.

Planning and Generalized 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, Nau, and Traverso 2004; Geffner and Bonet 2013; Haslum et al. 2019). The domain D mainly 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 $p(x_1, x_2, \ldots, x_k)$ where p is a predicate symbol (also called domain predicate) 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* $p(o_1, o_2, \ldots, o_k)$, where $o_i \in O$ and p is a predicate of arity k, 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 $S_0, S_1, \ldots, S_n$ 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 policy π can be represented in many forms from formulas or rules to general value functions $V(S)$; in the latter, the state transitions (S, S') in π are those that minimize the value $V(S')$, for the successor states S' of S.

Graph Neural Networks (GNNs)

GNNs are parametric functions that operate on graphs (Scarselli et al. 2009; Gilmer et al. 2017; Hamilton 2020). GNNs maintain and update embeddings $\boldsymbol{f}_i(v) \in \mathbb{R}^k$ for each vertex v in a graph G. The process is iteratively performed over L layers, from initial embeddings $\boldsymbol{f}_0(v)$, and progressing for $i = 0, \ldots, L-1$:

$$\boldsymbol{f}_{i+1}(v) = \mathrm{comb}_i\big(\boldsymbol{f}_i(v), \mathrm{agg}_i\big(\{\!\{\boldsymbol{f}_i(w) \mid w \in N_G(v)\}\!\}\big)\big) \quad (1)$$

Algorithm 1: Relational GNN (R-GNN)

```
**Input:** Set of ground atoms $S$ (state), and objects $O$
**Output:** Embeddings $\boldsymbol{f}_L(o)$ for each object $o \in O$
Initialize $\boldsymbol{f}_0(o) \sim 0^k$ for each object $o \in O$
**for** $i \in \{0, \dots, L-1\}$ **do**
    **for** each atom $q := p(o_1, o_2, \dots, o_m) \in S$ **do**
        $\boldsymbol{m}_{q,o_j} := [\text{MLP}_p(\boldsymbol{f}_i(o_1), \boldsymbol{f}_i(o_2), \dots, \boldsymbol{f}_i(o_m))]_j$
    **end for**
    **for** each object $o \in O$ **do**
        $\boldsymbol{f}_{i+1}(o) := \boldsymbol{f}_i(o)$
            $+ \text{MLP}_U\big(\boldsymbol{f}_i(o), \text{agg}\big(\{\{\boldsymbol{m}_{q,o} \mid o \in q, q \in S\}\}\big)\big)$
    **end for**
**end for**
```

where agg_i and comb_i are aggregation and combination functions, respectively, and $\{\{\boldsymbol{f}_i(w) \mid w \in N_G(v)\}\}$ is the *multiset* of embeddings $\boldsymbol{f}_i(w)$ for the neighboring vertices w of v in the graph G. The aggregation functions agg_i (e.g., max, sum, or smooth-max) condense multiple vectors into a single vector, whereas the combination functions comb_i merge 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. The aggregation and combination functions are parametric, allowing the vertex embeddings $\boldsymbol{f}_i(\cdot)$ to be learnable functions.

Relational GNNs (R-GNNs)

GNNs operate over graphs, whereas planning states are relational structures over predicates of varying arities. The Relational GNN (R-GNN) for processing relational structures (Ståhlberg, Bonet, and Geffner 2022a) is inspired by those used for max-CSPs (Toenshoff et al. 2021). Like GNNs, R-GNNs is a message-passing architecture where messages are exchanged between the objects in the input relational structure $\mathcal{A}$, but the messages are not directly associated with edges. Instead, messages are exchanged according to the atoms that are true in the structure. That is, initial embeddings $\boldsymbol{f}_0(o)$ for each object o in $\mathcal{A}$ are updated as follows:

$$\boldsymbol{f}_{i+1}(o) = \text{comb}_i\big(\boldsymbol{f}_i(o), \text{agg}_i\big(\{\{\boldsymbol{m}_{q,o} \mid o \in q, \mathcal{A} \vDash q\}\}\big)\big)\,, \quad (2)$$

where $\boldsymbol{m}_{q,o}$ is the message that atom q (that is true in the relational structure $\mathcal{A}$, and that mentions object o) sends to object o. For a m-ary predicate p, an atom of the form $q = p(o_1, o_2, \dots, o_m)$ sends m (non-necessarily equal) messages to the objects $o_1, o_2, \dots, o_m$, respectively. All such messages are computed (in parallel) using a *learnable* combination function $\text{comb}_p(\cdot)$, one for each symbol p, that maps m input embeddings into m output messages:

$$\boldsymbol{m}_{q,o_j} = \big[\text{comb}_p\big(\boldsymbol{f}_i(o_1), \boldsymbol{f}_i(o_2), \dots, \boldsymbol{f}_i(o_m)\big)\big]_j\,, \quad (3)$$

where $[\dots]_j$ refers to the j-th embedding of its argument. The $\text{comb}_i(\cdot)$ function in (2) merges two vectors of size k, the current embedding $\boldsymbol{f}_i(o)$ and the aggregation of the messages received at object o.

The relational neural network for planning states S 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. 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 $p(o_1, o_2, \dots, o_m)$ is an atom in G, the atom $p_g(o_1, o_2, \dots, o_m)$ is added to S, where p_g is a new "goal predicate" (Martín and Geffner 2004).

The set of object embeddings $\boldsymbol{f}_L(o)$ at the last layer is the result of the net; i.e., $\text{R-GNN}(S, O) = \{\{\boldsymbol{f}_L(o) \mid o \in O\}\}$. 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 a final MLP:

$$V(S) = \text{MLP}\big(\textstyle\sum_{o \in O} \boldsymbol{f}_L(o)\big)\,. \quad (4)$$

Weisfeiler-Leman Coloring Algorithms

Weisfeiler-Leman (WL) coloring algorithms provide the theory for establishing the expressive limitation of GNNs. These 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. 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:

$$C^1_{i+1}(v) = \text{Relabel}\big(\langle C^1_i(v), \{\{C^1_i(u) \mid u \in N(v)\}\}\rangle\big) \quad (5)$$

where $N(v)$ is the neighborhood of node v, and the initial coloring is determined by the given vertex colors, if any, or uniform otherwise.

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

$$C^k_{i+1}(\langle v\rangle) = \text{Relabel}\big(\langle C^k_i(\langle v\rangle), M_i(\langle v\rangle)\rangle\big) \quad (6)$$

where $M_i(\langle v\rangle)$ is the multiset of tuples

$$M_i(\langle v\rangle) = \{\{\big(C^k_i(\phi_1(\langle v\rangle, w)), \dots, C^k_i(\phi_k(\langle v\rangle, w))\big) \mid w \in V(G)\}\}\,, \quad (7)$$

the function $\phi_j(\langle v\rangle, w)$ replaces the j-th component of the tuple $\langle v\rangle$ with the node w, and the initial color for tuple $\langle v\rangle$ is determined by the structure of the subgraph induced by $\langle v\rangle$, and the order of the vertices in the tuple $\langle v\rangle$.

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

$$C^{k*}_{i+1}(\langle v\rangle) = \text{Relabel}\big(\langle C^{k*}_i(\langle v\rangle), M^*_i(\langle v\rangle)\rangle\big) \quad (8)$$

where the multiset $M_i(\langle v\rangle)$ of vectors in k-FWL is replaced by a k-dimensional vector $M^*_i(\langle v\rangle)$ of multisets:

$$\big[M^*_i(\langle v\rangle)\big]_j = \{\{C^{k*}_i(\phi_j(\langle v\rangle, w)) \mid w \in V(G)\}\} \quad (9)$$

for $j = 1, 2, \ldots, k$, where $\phi_j(\langle v\rangle, w)$ is the same function as in k-FWL, and the initial coloring $M_0^*(\langle v\rangle)$ is also the same.

For a tuple $\langle v\rangle$ of k vertices, there are potentially $k \cdot n$ neighbor-tuples resulting from replacing each j-th component v_j of $\langle v\rangle$ with each of the n nodes 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 $\langle v\rangle$. In k-FWL, the tuple $\langle v\rangle$ "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 $\langle v\rangle$ "sees" a k-vector whose elements $\langle v\rangle_j$ are multisets of n colors, with the j component v_j of $\langle v\rangle$ 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" as it is oblivious to such contexts.

2-FWL vs. 2-OWL. The case for $k = 2$ clearly illustrates the difference between the two algorithms. For a graph $G = (V, E)$ and coloring C, the context for the pair $\langle u, v\rangle$ considered by 2-FWL is

$$M(\langle u, v\rangle) = \{\!\!\{ \big(C(\langle w, v\rangle), C(\langle u, w\rangle)\big) \mid w \in V \}\!\!\} \,.$$

It considers pairs of tuples $\langle u, w\rangle$ and $\langle w, v\rangle$ whose "join" results in $\langle u, v\rangle$. On the other hand, the context considered by 2-OWL is

$$M^*(\langle u, v\rangle) = \big(\{\!\!\{C(\langle w, v\rangle) \mid w \in V\}\!\!\}, \{\!\!\{C(\langle u, w\rangle) \mid w \in V\}\!\!\}\big).$$

It is clear the "loss of information" suffered by 2-OWL with respect to 2-FWL as the context $M^*(\langle u, v\rangle)$ can be recovered from $M(\langle u, v\rangle)$, but not the other way around.

Expressive power. Cai, Fürer, and Immerman (1992) established that two graphs are indistinguishable by k-FWL if and only if they satisfy the same set of formulas in the logic C_{k+1}. In terms of expressive power, 1-OWL is equivalent to 2-OWL, k-OWL is strictly more expressive than $(k-1)$-OWL, for $k \geq 3$, and 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.

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. Indeed, the function computed by the new architecture, R-GNN[t](S, O), where t is a non-negative integer parameter, is defined as:

$$\text{R-GNN}[t](S, O) = \text{R-GNN}(A_t(S), O^2) \qquad (10)$$

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 the parameter t. Specifically, if $w = \langle o_1, o_2, \ldots, o_m\rangle$ is a tuple of objects, $\langle w\rangle^2$ refers to the tuple of m^2 pairs obtained from w as:

$$\langle w\rangle^2 = \langle (o_1, o_1), \ldots, (o_1, o_m), \ldots, (o_m, o_1), \ldots, (o_m, o_m)\rangle \qquad (11)$$

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

$$A_0(S) = \{p(\langle w\rangle^2) \mid p(w) \in S\}\,. \qquad (12)$$

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 $A_0(S)$ with atoms for a new ternary predicate $\triangle$ as: $A_t(S) = A_0(S) \cup \Delta_t(S)$ where

$$\Delta_t(S) = \big\{\triangle(\langle o, o'\rangle, \langle o', o''\rangle, \langle o, o''\rangle) \mid \langle o, o'\rangle, \langle o', o''\rangle \in R_t\big\}\,, \qquad (13)$$

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

$$\langle o, o'\rangle \in R_t \text{ iff } \begin{cases} o \text{ and } o' \text{ are both in an atom in } S & \text{if } t = 1, \\ \exists o''[\{\langle o, o''\rangle, \langle o'', o'\rangle\} \subseteq R_{t-1}] & \text{if } t > 1. \end{cases} \qquad (14)$$

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 $A_0(S)$ whose arguments 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 the Edge Transformer. But, 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 by simply changing the input from $\langle S, O\rangle$ to $\langle A_t(S), O^2\rangle$.

The final embeddings produced by R-GNN[t] are then used to define a general value function $V(S)$ as

$$V(S) = \text{MLP}\big(\textstyle\sum_{o \in O} \boldsymbol{f}_L(\langle o, o\rangle)\big)\,, \qquad (15)$$

where 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 $A_0(S)$ that mark the pairs $\langle o, o\rangle$ as different from other pairs $\langle o, o'\rangle$.

Learning Task

We aim at extending the expressivity of R-GNNs for finding policies for generalized planning. For this purpose, for each domain in the benchmark, we learn a general value function V in a supervised manner from the optimal values $V^*(S)$ over a small collection of training states S. Such states S belong to planning instances with small state spaces which makes the computation of V^* straightforward with a standard breadth-first search. The loss function that is minimized during training is:

$$L(S) = |V^*(S) - V(S)|\,. \tag{16}$$

Additionally, batches for training are created containing as many distinct V^* values as possible. This can be done as the V^* values for the states in the training set are pre-computed.

Baselines

We compare the new "architecture" R-GNN[t] with four baseline architectures: (plain) R-GNN, Edge Transformers (ETs), R-GNN$_2$, and 2-GNNs. While ETs and R-GNN$_2$ aim to match the expressive power of 2-FWL and hence C_3, 2-GNNs match the expressive power of 2-OWL and hence C_2, but embedding also pairs of objects. 3-GNNs are unfeasible in practice given the large number of objects; e.g., with 50 objects, there are 125,000 object triplets.

Edge Transformers (ETs)

Briefly, ETs are designed to operate on a *complete graph* with n nodes and n^2 directed edges (Bergen, O'Donnell, and Bahdanau 2021). Each edge is embedded as a k-dimensional feature vector. The core computation of an ET layer is described by the equation:

$$\boldsymbol{f}_{i+1}(u,v) = \text{MLP}(\text{LN}(\boldsymbol{f}_i(u,v) + \text{Tri.Att.}(\text{LN}(\boldsymbol{f}_i(u,v))))) \tag{17}$$

where LN represents layer normalization, and $\boldsymbol{f}_i(u,v)$ is the feature vector for pair $\langle u,v\rangle$ at layer i. A fundamental aspect of ETs is the triangular attention mechanism. This mechanism functions by aggregating information from all pairs of edges that share a common vertex; i.e., for $\langle u,v\rangle$, it aggregates information from the pairs $\{(\langle u,w\rangle,\langle w,v\rangle) \mid w \in V\}$, like 2-FWL, using a self-attention-based combination of contributions. The expressive power of ETs is the same as for 2-FWL (Müller et al. 2024).

Note that, ETs requires that information about true atoms in a state S and goal G to be included in the initial embeddings. This restricts ET to binary relations, and unary predicates are mapped to binary ones by repeating the first term. To encode the state and the goal, we learn two k-dimensional vectors, $\boldsymbol{e}_p$ and $\boldsymbol{e}_{p_g}$, for each predicate p. The initial embedding $\boldsymbol{e}_{o,o'}$ for the pair $\langle o,o'\rangle$ is:

$$\boldsymbol{e}_{o,o'} = \textstyle\sum_p \big(\boldsymbol{e}_p \cdot \llbracket p(o,o') \in S \rrbracket + \boldsymbol{e}_{p_g} \cdot \llbracket p(o,o') \in G \rrbracket\big), \tag{18}$$

where $\llbracket\cdot\rrbracket$ is the Iverson bracket.

These initial embeddings encode the state and the goal in a way that is suitable for the network. For the value function, we use the same readout function as for R-GNN[t] in all baselines, which aggregates the final embeddings for pairs of identical objects and feeds the resulting vector into an MLP; cf. (15). Hence, the number of embeddings aggregated in the final readout is the same for R-GNN, R-GNN[t], and ET.

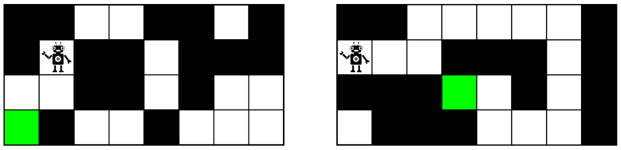

Figure 1: Two $8{\times}4$ test instances of the Navig-xy domain where the robot has to reach the green cell in a grid with obstacles, and where the objects are the values of each one of the two coordinates, and not the cells themselves. The problem is not in C_2 in this representation, and indeed, after 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.

R-GNN$_2$

The final two baselines are our own. Rather than selectively adding $\triangle$ atoms, as in R-GNN[t], we introduce these atoms based on the object pairs rather than the state and the goal. For this baseline, we aim to emulate 2-FWL by augmenting the input with all possible atoms of the form:

$$\triangle(\langle o,o'\rangle, \langle o',o''\rangle, \langle o,o''\rangle), \tag{19}$$

where each atom encodes a composition of two object pairs. This approach is similar to R-GNN[t], but also ET. In all three, expressivity beyond C_2 is achieved through a "triangulation" mechanism specifically designed to mirror the 2-FWL procedure. In WL terms, both ETs and R-GNN$_2$ maintain the "context" of triplets, thereby achieving greater expressiveness.

2-GNNs

The other baseline is designed to emulate 2-OWL instead of 2-FWL. This allows us to test whether the performance improvement is due to the object pairs themselves rather than to the additional atoms. In 2-OWL, the color $C_{i+1}(\langle u,v\rangle)$ of a pair $\langle u,v\rangle$ is determined based on the color $C_i(\langle u,v\rangle)$ and the multisets $\{\!\{C_i(\langle w,v\rangle) \mid w \in V\}\!\}$ and $\{\!\{C_i(\langle u,w\rangle) \mid w \in V\}\!\}$. This is emulated by introducing two binary predicates, p_1 and p_2 to represent these multisets. The ground atoms that determine which pair communicates are all possible atoms with one of the following forms:

$$p_1(\langle w,v\rangle, \langle u,v\rangle) \tag{20}$$

$$p_2(\langle u,w\rangle, \langle u,v\rangle) \tag{21}$$

The initial embeddings encode the state and goal using the same approach described for the ET baseline.

Example: Grid Navigation with Obstacles

We illustrate the expressivity demands on a simple example and how these demands are met by the different architectures. 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 in $X \cup Y$, where $X = \{x_1,\ldots,x_n\}$ and $Y = \{y_1,\ldots,y_m\}$, and successor relations SUCC-X and SUCC-Y. The domain also includes the binary relations AT(x,y) to specify the initial and goal cells,

$\textsc{Blocked}(x,y)$ to specify the cells that are blocked, and a dummy $\textsc{Cell}(x,y)$ to identify the cells in the grid, for $x \in X$ and $y \in Y$.

Under the experimental settings described below 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 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 R-GNN can only compute "Manhattan distances", that in some cases (e.g., the left grid in Fig. 1) are good or perfect approximations of $V^*(s)$.

Experiments

A learned value function V, for a 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 with no unvisited successors. Reaching a goal is counted as a success, else as a failure.

For learning 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.[1] For each 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 embedding dimension $k = 64$, $L = 30$ layers for R-GNN, R-GNN[t] and the ETs. For R-GNN$_2$, we used $k = 32$ to avoid running out of memory during training. In all approaches, all layers share weights, and the ETs have 8 self-attention heads.

Inference time depends on the size of the net, but it is typically in the order of tens of milliseconds. The time to decide which successor to take is the number of successor states multiplied by this time.

Domains

Brief descriptions of the domains used in the experiments, mostly taken from Ståhlberg, Bonet, and Geffner (2022a; 2022b; 2023), follow. In all cases, the instances in the training set are small, while those in the test set are significantly larger as they contain more objects.

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 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 travel 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, up to 100 cells.

Results

Tables 1 and 2 show the results. We anticipate that the improved expressiveness of the networks will result in:

- Maintaining performance levels on C_2 domains; and
- Achieving broader coverage on C_{3+} domains, or
- Generating plans of superior quality.

These expections are mostly confirmed by the experiments. The coverage results for R-GNN[$t = 0, 1$] on C_2 domains, as seen in Table 1, remain consistent with R-GNN, with plan quality largely unchanged. Just in one case, Gripper, increasing the parameter t from 0 to 1 leads to a decline in coverage for R-GNN[t]. We attribute this to the high volume

[1] Code, data and models: https://zenodo.org/records/14505092.

			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
	R-GNN$_2$	14 / 17 (82 %)	922	35	65
	2-GNN	**17 / 17 (100 %)**	678	36	39
	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
	R-GNN$_2$	**22 / 22 (100 %)**	936	39	42
	2-GNN	20 / 22 (91 %)	750	40	37
	ET	18 / 22 (82 %)	966	39	53
Gripper	R-GNN	**18 / 18 (100 %)**	4,800	231	266
	R-GNN[0]	**18 / 18 (100 %)**	1,764	98	98
	R-GNN[1]	11 / 18 (61 %)	847	77	77
	R-GNN$_2$	**18 / 18 (100 %)**	1,764	98	98
	2-GNN	1 / 18 (6 %)	53	53	53
	ET	4 / 18 (22 %)	246	61	61
Miconic	R-GNN	**20 / 20 (100 %)**	1,342	67	67
	R-GNN[0]	**20 / 20 (100 %)**	1,566	71	78
	R-GNN[1]	**20 / 20 (100 %)**	2,576	71	128
	R-GNN$_2$	**20 / 20 (100 %)**	1,342	67	67
	2-GNN	12 / 20 (60 %)	649	54.5	54
	ET	**20 / 20 (100 %)**	1,368	68	68
Visitall	R-GNN	18 / 22 (82 %)	636	29	35
	R-GNN[0]	21 / 22 (95 %)	1,128	35	53
	R-GNN[1]	**22 / 22 (100 %)**	886	35	40
	R-GNN$_2$	20 / 22 (91 %)	739	33	36
	2-GNN	18 / 22 (82 %)	626	32	34
	ET	18 / 22 (82 %)	670	29	37

Table 1: Coverage and plan lengths for C_2 domains. In these domains, R-GNNs performs best, and R-GNN[1] is competitive, except in Gripper.

			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
	R-GNN$_2$	10 / 20 (50 %)	124	11.5	12
	2-GNN	6 / 20 (30 %)	82	11.5	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 %)**	1,057	52	52
	R-GNN$_2$	15 / 20 (75 %)	799	52	53
	2-GNN	0 / 20 (0 %)	–	–	–
	ET	0 / 20 (0 %)	–	–	–
Rovers	R-GNN	9 / 20 (45 %)	2,599	280	288
	R-GNN[0]	**14 / 20 (70 %)**	2,418	153	172
	R-GNN[1]	**14 / 20 (70 %)**	1,654	55	118
	R-GNN$_2$	11 / 20 (55 %)	2,225	239	202
	2-GNN	Unsuitable domain: ternary predicates			
	ET				
Vacuum	R-GNN	**20 / 20 (100 %)**	4,317	141	215
	R-GNN[0]	**20 / 20 (100 %)**	183	9	9
	R-GNN[1]	**20 / 20 (100 %)**	192	9	9
	R-GNN$_2$	**20 / 20 (100 %)**	226	9	11
	2-GNN	Unsuitable domain: ternary predicates			
	ET				
Visitall-xy	R-GNN	5 / 20 (25 %)	893	166	178
	R-GNN[0]	15 / 20 (75 %)	1,461	84	97
	R-GNN[1]	**20 / 20 (100 %)**	1,829	83	91
	R-GNN$_2$	19 / 20 (95 %)	2,428	116	127
	2-GNN	12 / 20 (60 %)	1,435	115	119
	ET	3 / 20 (15 %)	455	138	151

Table 2: Coverage and plan lengths for C_3 domains. In these domains, R-GNN[1] performs best, but both R-GNN[0] and R-GNN$_2$ outperform R-GNN and ET.

of messages that are exchanged, which slows down training and may result in incomplete convergence. The plan quality across all approaches is comparable for the C_2 domains, except in the Gripper domain, where R-GNN produces longer plans. This is not due to a lack of expressiveness; rather, we believe that the additional expressiveness provided by the other approaches result in a more stable general policy.

For the C_3 domains, shown in Table 2, we observe that R-GNN has limited coverage across all domains except Vacuum, where it generates very long plans. The Vacuum domain requires C_3 expressiveness, as each robot has its own traversal capabilities, necessitating the network to determine which agent is closest relative to their capabilities. While R-GNN achieves high coverage, actions are executed without clear intention, and goal states are reached incidentally. This is reflected in the long plan lengths for R-GNN, whereas other approaches produce optimal or near-optimal plans.

In the other C_3 domains, R-GNN[1] consistently outperforms R-GNN in coverage due to its increased expressiveness. The performance difference between R-GNN[0] and R-GNN[1] depends on the need for composition. In Grid, Logistics, and Visitall-xy, at least one level of composition is required, and by including these atoms, we observe improved coverage. In Rovers, although the necessity for composition is unclear, the plan quality is significantly improved. Optimal planning in Grid (Helmert 2003) is NP-hard, and it seems to be challenging in Rovers as well, and this appears to be the reason why less than 100 % coverage was achieved.

The baseline R-GNN$_2$ surpasses ET in all domains except Blocks-s. We believe this is due to the aggregation function: the output of the softmax in the attention mechanism depend on the number of objects, leading to value magnitudes that differ from those encountered during training. This is not an issue in R-GNN$_2$, where a smooth maximum is used as an aggregation function. While R-GNN$_2$ performs better than R-GNN in C_3 domains, it underperforms compared to R-GNN[1]. This discrepancy is not due to expressiveness, as R-GNN$_2$ is theoretically more expressive. Rather, it may be easier to identify the relevant compositions since R-GNN[1] has far fewer compositions in its input.

The baseline 2-GNN consistently performs worse than R-GNN[0] in our experiments, even though both models use object pairs and do not derive compositions. This disparity is likely due to the reduced volume of messages passed in R-GNN[0], which allows for clearer messages. Additionally, each message in R-GNN[0] is computed using MLPs tailored to the predicate symbols of the atoms, leading to more inductive bias and thus better generalization.

On the Expressivity of the Model

We establish next that the R-GNN[t] model has the capability to capture compositions of binary relations that can be expressed in C_3. This capability is critical in many domains, and can be achieved by adding derived predicates (Ståhlberg, Bonet, and Geffner 2022b; Haslum et al. 2019; Thiébaux, Hoffmann, and Nebel 2005). In particular, we are interested in derived predicates that correspond to relational joins in C_3:

Definition 1 (C_3-Relational Joins)**.** *Let σ be a relational language. The class $\mathcal{J}_3 = \mathcal{J}_3[\sigma]$ of relational joins over the language σ is the* ***smallest*** *class of formulas that satisfy the following properties:*

1. $\{R(x,y), \neg R(x,y)\} \subseteq \mathcal{J}_3$ *for binary relation R in σ,*
2. $\varphi(x,y) \wedge \phi(x,y) \in \mathcal{J}_3$ *if* $\{\varphi(x,y), \phi(x,y)\} \subseteq \mathcal{J}_3$*,*
3. $\varphi(x,y) \vee \phi(x,y) \in \mathcal{J}_3$ *if* $\{\varphi(x,y), \phi(x,y)\} \subseteq \mathcal{J}_3$*, and*
4. $\exists z[\varphi(x,z) \wedge \phi(y,z)] \in \mathcal{J}_3$ *if* $\{\varphi(x,y), \phi(y,z)\} \subseteq \mathcal{J}_3$*.*

The notation $\varphi(x,y)$ means that φ is a formula whose free variables are among $\{x,y\}$.

For example, if σ contains the relations $\textsc{Key}(k,s)$ and $\textsc{Lock}(\ell,t)$ to express that the key k has shape s, and the lock ℓ has shape t, respectively, then $\mathcal{J}_3$ contains $\varphi(k,\ell) = \exists s[\textsc{Key}(k,s) \wedge \textsc{Lock}(\ell,s)]$ that is true for the pair $\langle k,\ell\rangle$ when the key k opens the lock ℓ.

Let $\mathsf{STRUC} = \mathsf{STRUC}[\sigma]$ be the class of finite structures for language σ. A network that maps structures $\mathcal{A}$ in STRUC into embeddings for all the k-tuples $\langle u_1, u_2, \ldots, u_k\rangle$ of objects in $\mathcal{A}$ is called a **k-embedding network,** and a collection of such networks is a **k-embedding architecture.** For example, the class of all nets in R-GNN[t] for σ is a 2-embedding architecture. The architecture R-GNN[σ, t, k, L] is the collection of networks in R-GNN[t] for the language σ, embedding dimension k, and L layers.

Let $\varphi(x,y)$ be a relational join, and let $\mathcal{A}$ be a structure with universe U. The denotation of $\varphi(x,y)$ over $\mathcal{A}$, denoted by $\mathcal{A}^\varphi$, is the set of pairs $\{\langle u,v\rangle \in U^2 \mid \mathcal{A} \vDash \varphi(u,v)\}$. A 2-embedding network N with embedding dimension k computes $\varphi(x,y)$ if there is an index $0 \leq j < k$ such that the pair $\langle u,v\rangle \in \mathcal{A}^\varphi$ iff $\boldsymbol{f}(\langle u,v\rangle)_j = 1$, where $\boldsymbol{f}(\langle u,v\rangle)$ is the embedding for $\langle u,v\rangle$ produced by N on input $\mathcal{A}$. Likewise, such a network N computes a collection $\mathcal{D}$ of relational joins if N computes each join $\varphi(x,y)$ in $\mathcal{D}$.

Theorem 2 (Computation of C_3-Relational Joins)**.** *Let σ be a relational language, and let $\mathcal{D}$ be a* ***finite collection*** *of C_3-relational joins. Then, there is a tuple of parameters $\langle t,k,L\rangle$ and network N in R-GNN$[\sigma,t,k,L]$ that computes $\mathcal{D}$.*

The parameters $\langle t,k,L\rangle$ are determined by the joins in $\mathcal{D}$. The index t that defines the nesting depth of the Δ atoms is the maximum quantifier depth over (the joins in) $\mathcal{D}$. On the other hand, the embedding dimension k and the number of layers L are bounded by the sum and maximum, respectively, of the number of subformulas of the joins in $\mathcal{D}$.

To illustrate the capabilities of the proposed architecture, let us consider the Navig-xy domain from above for which σ contains the relations $\textsc{Succ-x}$, $\textsc{Succ-y}$, $\textsc{At}$, $\textsc{At}_g$, $\textsc{Blocked}$, and $\textsc{Cell}$. We want to show that $\mathcal{J}_3$ contains the predicate $\varphi_k(x,y)$ that tells when the cell $\langle x,y\rangle$ is at k steps from the goal cell. Indeed, $\varphi_0(x,y) = \textsc{At}_g(x,y)$ is in $\mathcal{J}_3$. Likewise,

$$\begin{aligned}\textsc{Adj-x}(x,x') &= \textsc{Succ-x}(x,x') \vee \textsc{Succ-x}(x',x)\,,\\ \textsc{Adj-y}(y,y') &= \textsc{Succ-y}(y,y') \vee \textsc{Succ-y}(y',y)\end{aligned}$$

belong to $\mathcal{J}_3$. Then, the following also belong to $\mathcal{J}_3$:

$$\begin{aligned}\phi^{\text{X}}_{k+1}(x,y) &= \exists z[\varphi_k(z,y) \wedge \textsc{Adj-x}(x,z)]\,,\\ \phi^{\text{Y}}_{k+1}(x,y) &= \exists z[\varphi_k(x,z) \wedge \textsc{Adj-y}(y,z)]\,,\\ \phi_{k+1}(x,y) &= \phi^{\text{X}}_{k+1}(x,y) \vee \phi^{\text{Y}}_{k+1}(x,y)\,,\\ \varphi_{k+1}(x,y) &= \neg\textsc{Blocked}(x,y) \wedge \phi_{k+1}(x,y)\,.\end{aligned}$$

Hence, for any state S in an instance of Navig-xy, $V^*(S) = k$ iff $S \vDash \textsc{Dist}_k$ where $\textsc{Dist}_k = \exists xy[\textsc{At}(x,y) \wedge \varphi_k(x,y)]$ is a sentence in $\mathcal{J}_3$. Therefore, for a class of instances of bounded x and y dimensions, there is an embedding dimension k, a number L of layers, and a network N in R-GNN[$\sigma, 1, k, L$] that computes V^* for the states S in such instances.

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 ability than Edge Transformers, which have the expressiveness of 3-GNNs, but with much less overhead. Interestingly, for all of the domains we considered, we did not see any improvement using $t > 1$. This is exploited by R-GNN[t], as far fewer compositions need to be considered, resulting in much faster inference and models that generalize better than the baselines that consider all possible compositions.

It remains to be studied whether the full power of C_3 is needed to handle most planning domains. The expressivity results from Horcík and Sír (2024) and Drexler et al. (2024) suggest that the expressive power of C_3 is sufficient but not necessary. Indeed, there is ample room in the middle, between C_2 and C_3, which the R-GNN[t] architecture exploits, in contrast with R-GNN$_2$ and ETs. Approaches that aim to extend representations with new unary predicates by considering cycles in the state graphs may yield an acceptable tradeoff between expressivity and efficiency, without the need to embed pairs or higher tuples of objects at all. This is left for future work.

Acknowledgments

The research has been supported by the Alexander von Humboldt Foundation with funds from the Federal Ministry for Education and Research, Germany, by the European Research Council (ERC), Grant agreement No. 885107, by the Excellence Strategy of the Federal Government and the NRW Länder, Germany, and by the Knut and Alice Wallenberg (KAW) Foundation under the WASP program.

References

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Bajpai, A. N.; Garg, S.; et al. 2018. Transfer of deep reactive policies for MDP planning. In *Proc. NeurIPS*, 10965–10975.

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

Barceló, P.; Kostylev, E.; Monet, M.; Pérez, J.; Reutter, J.; and Silva, J.-P. 2020. The Logical Expressiveness of Graph Neural Networks. In *Proc. ICLR*.

Belle, V.; and Levesque, H. J. 2016. Foundations for Generalized Planning in Unbounded Stochastic Domains. In *Proc. KR*, 380–389.

Bergen, L.; O'Donnell, T. J.; and Bahdanau, D. 2021. Systematic Generalization with Edge Transformers. In *Proc. NeurIPS*, 1390–1402.

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

Bonet, B.; Francès, G.; and Geffner, H. 2019. Learning Features and Abstract Actions for Computing Generalized Plans. In *Proc. AAAI*, 2703–2710.

Bonet, B.; and Geffner, H. 2018. Features, Projections, and Representation Change for Generalized Planning. In *Proc. IJCAI*, 4667–4673.

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic Dynamic Programming for First-Order MDPs. In *Proc. IJCAI*, 690–700. Morgan Kaufmann.

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

Chevalier-Boisvert, M.; Bahdanau, D.; Lahlou, S.; Willems, L.; Saharia, C.; Nguyen, T. H.; and Bengio, Y. 2019. BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning. In *Proc. ICLR*.

Drexler, D.; Seipp, J.; and Geffner, H. 2022. Learning Sketches for Decomposing Planning Problems into Subproblems of Bounded Width. In *Proc. ICAPS*, 62–70.

Drexler, D.; Ståhlberg, S.; Bonet, B.; and Geffner, H. 2024. Symmetries and Expressive Requirements for Learning General Policies. In *Proc. KR*.

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

Francès, G.; Bonet, B.; and Geffner, H. 2021. Learning General Planning Policies from Small Examples Without Supervision. In *Proc. AAAI*, 11801–11808.

François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M. G.; and Pineau, J. 2018. An Introduction to Deep Reinforcement Learning. *Foundations and Trends in Machine Learning*.

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

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *Proc. ICML*, 1263–1272.

Grohe, M. 2021. The Logic of Graph Neural Networks. In *Proc. LICS*, 1–17.

Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2018. Learning Generalized Reactive Policies Using Deep Neural Networks. In *Proc. ICAPS*, 408–416.

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

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

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2): 219–262.

Horcík, R.; and Sír, G. 2024. Expressiveness of Graph Neural Networks in Planning Domains. In *Proc. of the 34th International Conference on Automated Planning and Scheduling (ICAPS 2024)*. AAAI Press.

Hu, Y.; and Giacomo, G. D. 2011. Generalized Planning: Synthesizing Plans that Work for Multiple Environments. In *Proc. IJCAI*, 918–923.

Illanes, L.; and McIlraith, S. A. 2019. Generalized Planning via Abstraction: Arbitrary Numbers of Objects. In *Proc. AAAI*, 7610–7618.

Jiménez, S.; Segovia-Aguas, J.; and Jonsson, A. 2019. A Review of Generalized Planning. *The Knowledge Engineering Review*, 34: e5.

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

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *Proc. ICLR*.

Kirk, R.; Zhang, A.; Grefenstette, E.; and Rocktäschel, T. 2023. A Survey of Zero-shot Generalisation in Deep Reinforcement Learning. *Journal of Artificial Intelligence Research*, 76: 201–264.

Maron, H.; Ben-Hamu, H.; Serviansky, H.; and Lipman, Y. 2019a. Provably Powerful Graph Networks. In *Proc. NeurIPS*.

Maron, H.; Ben-Hamu, H.; Shamir, N.; and Lipman, Y. 2019b. Invariant and Equivariant Graph Networks. In *Proc. ICLR*.

Martín, M.; and Geffner, H. 2004. Learning Generalized Policies from Planning Examples Using Concept Languages. *Applied Intelligence*, 20(1): 9–19.

Misra, D. 2020. Mish: A Self Regularized Non-Monotonic Activation Function. In *Proceedings of the 31st British Machine Vision Conference (BMVC 2020)*. BMVA Press.

Morris, C.; Lipman, Y.; Maron, H.; Rieck, B.; Kriege, N. M.; Grohe, M.; Fey, M.; and Borgwardt, K. 2023. Weisfeiler and Leman go Machine Learning: The Story so far. *Journal of Machine Learning Research*, 24(333): 1–59.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proc. AAAI*, 4602–4609.

Müller, L.; Kusuma, D.; Bonet, B.; and Morris, C. 2024. Towards Principled Graph Transformers. In *Proc. NeurIPS*.

Rivlin, O.; Hazan, T.; and Karpas, E. 2020. Generalized Planning With Deep Reinforcement Learning. In *ICAPS 2020 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*, 16–24.

Sanner, S.; and Boutilier, C. 2009. Practical Solution Techniques for First-Order MDPs. *Artificial Intelligence*, 173(5-6): 748–788.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1): 61–80.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Van Den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *Proc. ESWC*, 593–607.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2008. Learning Generalized Plans Using Abstract Counting. In *Proc. AAAI*, 991–997.

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

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022a. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In *Proc. ICAPS*, 629–637.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022b. Learning Generalized Policies without Supervision Using GNNs. In *Proc. KR*, 474–483.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2023. Learning General Policies with Policy Gradient Methods. In *Proc. KR*.

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

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In Defense of PDDL Axioms. *Artificial Intelligence*, 168(1–2): 38–69.

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

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *Journal of Artificial Intelligence Research*, 68: 1–68.

Vashishth, S.; Sanyal, S.; Nitin, V.; and Talukdar, P. 2019. Composition-based Multi-Relational Graph Convolutional Networks. In *Proc. ICLR*.

Wang, C.; Joshi, S.; and Khardon, R. 2008. First Order Decision Diagrams for Relational MDPs. *Journal of Artificial Intelligence Research*, 31: 431–472.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How Powerful are Graph Neural Networks? In *Proc. ICLR*.