Learning Generalized Policies without Supervision Using GNNs

Simon Ståhlberg,[1] Blai Bonet,[2] Hector Geffner [3,2,1]

[1]Linköping University, Sweden
[2]Universitat Pompeu Fabra, Spain
[3]Institució Catalana de Recerca i Estudis Avançats (ICREA), Barcelona, Spain
simon.stahlberg@liu.se, bonetblai@gmail.com, hector.geffner@upf.edu

Abstract

We consider the problem of learning generalized policies for classical planning domains using graph neural networks from small instances represented in lifted STRIPS. The problem has been considered before but the proposed neural architectures are complex and the results are often mixed. In this work, we use a simple and general GNN architecture and aim at obtaining crisp experimental results and a deeper understanding: either the policy greedy in the learned value function achieves close to 100% generalization over instances larger than those used in training, or the failure must be understood, and possibly fixed, logically. For this, we exploit the relation established between the expressive power of GNNs and the C_2 fragment of first-order logic (namely, FOL with 2 variables and counting quantifiers). We find for example that domains with general policies that require more expressive features can be solved with GNNs once the states are extended with suitable "derived atoms" encoding role compositions and transitive closures that do not fit into C_2. The work follows an existing approach based on GNNs for learning optimal general policies in a supervised fashion, but the learned policies are no longer required to be optimal (which expands the scope, as many planning domains do not have general optimal policies) and are learned without supervision. Interestingly, value-based reinforcement learning methods that aim to produce optimal policies, do not always yield policies that generalize, as the goals of optimality and generality are in conflict in domains where optimal planning is NP-hard.

Introduction

Generalized planning is concerned with the computation of general policies for families of planning instances over the same domain that span different state spaces. For example, a general policy for solving Blocks problems can place all blocks on the table and stack then the desired towers, bottom up, one at at time. The formulation and the computation of general policies is particularly interesting since it involves ideas from planning, knowledge representation, and learning. Indeed, the language for representing the general policies is key, especially in domains where the set of ground actions is not fixed (Bonet and Geffner 2018). Also learning policies from examples has been found to be simpler than synthesizing them from specifications (Khardon 1999; Srivastava, Immerman, and Zilberstein 2008; Bonet, Palacios, and Geffner 2009; Hu and De Giacomo 2011; Belle and Levesque 2016; Segovia, Jiménez, and Jonsson 2016). In planning, it is common to assume that domain predicates are known, while some deep learning and deep reinforcement learning approaches assume no domain knowledge, representing the states, e.g., as 2D images (Chevalier-Boisvert et al. 2019; Campero et al. 2021; Cobbe et al. 2020).

In this paper, we consider the problem of learning generalized policies for classical planning domains using *graph neural networks* (Scarselli et al. 2008; Hamilton 2020) from small instances represented in lifted STRIPS. The problem has been considered before but using neural architectures that are more complex and with results that are often less crisp, involving in certain cases heuristic information or search (Toyer et al. 2020; Garg, Bajpai, and Mausam 2020; Rivlin, Hazan, and Karpas 2020; Karia and Srivastava 2021; Shen, Trevizan, and Thiébaux 2020). We use a simple and general GNN architecture and aim at obtaining crisp experimental results and a deeper understanding: either the policy greedy in the learned value function achieves close to 100% generalization over instances larger than those used in training, or the failure must be understood and, possibly fixed, using logical methods. For this, we exploit the relation between the expressive power of GNNs and the two-variable fragment of first-order logic with counting, C_2, that includes the standard description logics (Barceló et al. 2020; Grohe 2020). Description logic features have been used indeed for expressing general policies and general value functions (Martín and Geffner 2004; Fern, Yoon, and Givan 2006; Bonet, Francès, and Geffner 2019; Francès et al. 2019; Francès, Bonet, and Geffner 2021). We find for example that domains with general policies that require more expressive features can be solved with GNNs once the states are extended with suitable "derived atoms" for encoding role compositions and transitive closures that do not fit into C_2.

The work follows the GNN approach for learning optimal general policies in a supervised fashion (Ståhlberg, Bonet, and Geffner 2022) but the learned policies are no longer required to be optimal, which expands the scope of the approach, as many planning domains do not admit general optimal policies, and are learned without supervision. The learning problem becomes the problem of learning a value function V that can be applied to the states s of any domain instance, such that the greedy policy in V solves the training instances. Versions of this idea have been used in

combinatorial settings (Francès et al. 2019; Francès, Bonet, and Geffner 2021). Interestingly, value-based reinforcement learning methods that aim to produce optimal value functions $V = V^*$ are shown not to generalize as well in domains that admit (non-optimal) general policies but where optimal planning is NP-hard.

The rest of the paper is organized as follows. First we discuss related research, cover classical planning, general policies and value functions, GNNs, and the actual GNN architecture and loss functions used for learning. This is followed by the experimental section, analyses, and a summary.

Related Work

Some related research threads are the following.

Generalized planning (GP). Formulations of generalized planning differ in the way in which general policies are represented; most often, as logic programs, finite-state controllers, or programs with loops (Khardon 1999; Srivastava, Immerman, and Zilberstein 2008; Bonet, Palacios, and Geffner 2009; Hu and De Giacomo 2011; Belle and Levesque 2016; Segovia, Jiménez, and Jonsson 2016). In all cases, the most compact policies that manage to solve a family of examples are sought, and the key question is how the space of possible programs or controllers is defined.

GP with logical features. An alternative approach is to define the general policies as collection of rules over a set of logical features (Bonet and Geffner 2018), often derived from the domain predicates using a description logic grammar (Martín and Geffner 2004; Fern, Yoon, and Givan 2006). Recent methods learn such policies from pools of such features (Bonet, Francès, and Geffner 2019; Francès, Bonet, and Geffner 2021).

Generalized policies using deep learning. Deep learning and deep reinforcement learning methods have been used to compute general policies from sampled problems without having to predefine the space of possible features. In some cases, the planning representation of the domains is used (Toyer et al. 2020; Garg, Bajpai, and Mausam 2020; Rivlin, Hazan, and Karpas 2020); in other cases, it is not (Groshev et al. 2018; Chevalier-Boisvert et al. 2019; Campero et al. 2021; Cobbe et al. 2020). Also in some cases, the learning is supervised; in others, it is based on reinforcement learning (Bertsekas 1995; Sutton and Barto 2018; François-Lavet et al. 2018). The neural networks learn to map states into a feature representation that is mapped into a value or policy associated to the state.

GNNs and logic. A graph neural network learns to map vertices of a graph into feature representations that can be aggregated and fed into a feedforward neural network for classifying graphs, and more generally, for computing functions over graphs independently of their size (Scarselli et al. 2008; Hamilton 2020). Since the computational model is based on message passing, GNNs cannot distinguish all pairs of graphs that are not isomorphic but can distinguish those that are distinguished by the WL coloring procedure (Morris et al. 2019; Xu et al. 2019). These correspond in turn to those that can be distinguished by formulas in the two-variable fragment of first-order logic with counting quantifiers, C_2, which includes the standard description logics (Barceló et al. 2020; Grohe 2020).

GNNs and optimal general policies. Ståhlberg, Bonet, and Geffner (2022) use GNNs to learn optimal general policies in a supervised fashion from targets $V^*(s)$ and sampled states s, taking advantange of a GNN architecture introduced for learning to solve Max-CSPs (Toenshoff et al. 2021), extended to the more general relational structures underlying planning states where objects define the universe, predicates define the relations, and atoms define their denotations. In this work, we build on these results to learn general policies that are not necessarily optimal (and which hence cover more domains) without supervision and without having to predefine a pool of features (Francès et al. 2019).

Classical Planning

A classical planning problem is a pair $P = \langle D, I \rangle$ where D is a first-order *domain* and I contains information about the instance (Geffner and Bonet 2013; Ghallab, Nau, and Traverso 2016; Haslum et al. 2019a). The domain D contains a set of predicate symbols p and a set of action schemas with preconditions and effects given by atoms $p(x_1, \ldots, x_k)$ where each x_i is an argument of the schema. The instance is a tuple $I = \langle O, \mathit{Init}, \mathit{Goal} \rangle$ where O is a set of object names c_i, and *Init* and *Goal* are sets of *ground atoms* $p(c_1, \ldots, c_k)$.

A classical problem $P = \langle D, I \rangle$ encodes a state model $S(P) = \langle S, s_0, S_G, \mathit{Act}, A, f \rangle$ in compact form where the states $s \in S$ are sets of ground atoms from P, s_0 is the initial state I, S_G is the set of goal states s such that $S_G \subseteq s$, Act is the set of ground actions in P, $A(s)$ is the set of ground actions whose preconditions are (true) in s, and f is the transition function so that $f(a, s)$ for $a \in A(s)$ represents the state s' that follows action a in the state s. An action sequence $a_0, \ldots, a_n$ is applicable in P if $a_i \in A(s_i)$ and $s_{i+1} = f(a_i, s_i)$, for $i = 1, \ldots, n$, and it is a plan if $s_{n+1} \in S_G$. The *cost* of a plan is assumed to be given by its length and a plan is *optimal* if there is no shorter plan.

Note that the information in D is common to all instances, this is essential for defining and computing general policies. Recent work has addressed the problem of learning the domain D, i.e. the action schemas and predicates (Cresswell, McCluskey, and West 2013; Asai 2019; Bonet and Geffner 2020; Rodriguez et al. 2021).

General Policies and Value Functions

One approach for expressing general policies is as rules $C \mapsto E$ where the condition C and the effect E are defined in terms of state features (Bonet and Geffner 2018). State features or simply, features, refer to functions ϕ over the state, and Boolean and numerical features refer to state functions that return Boolean and numerical values. For example, a general policy for clearing a block x can be expressed in terms of the two features $\Phi = \{H, n\}$, where H is a true in a state if a block is being held, and n represents the number of blocks above x. The policy rules are

$$\neg H, n > 0 \mapsto H, n\downarrow \quad , \quad H \mapsto \neg H \tag{1}$$

that say that, when the gripper is empty and there are blocks above x, any action that decreases n and makes H true should be selected, and that when the gripper is not empty, any action that makes H false and does not affect n should be selected. General policies of this form can be learned without supervision by solving a combinatorial optimization problem $T(\mathcal{S}, \mathcal{F})$ where $\mathcal{S}$ is a set of sampled state transitions and $\mathcal{F}$ is a large but finite pool of description logic features obtained from the domain predicates (Bonet, Francès, and Geffner 2019; Francès, Bonet, and Geffner 2021).

Another way to represent (general) policies is by means of (general) value functions. In dynamic programming and RL (Bellman 1957; Sutton and Barto 2018; Bertsekas 1995), a value function V defines a (non-deterministic) *greedy policy* π_V that selects in a state s any possible successor state s' with minimum $V(s')$ value under the assumption that actions are deterministic and have the same cost. A policy π solves an instance P if the state transitions compatible with π, starting with the initial state, eventually end up in a goal state. If V is optimal, i.e., $V = V^*$, the greedy policy π_V is optimal too, selecting state transitions along optimal paths.

General value functions for a class of problems are defined in terms of features ϕ_i that have well-defined values over all states of such problems as:

$$V(s) \;=\; F(\phi_1(s), \ldots, \phi_k(s))\,. \tag{2}$$

Linear value functions have the form

$$V(s) \;=\; \sum\nolimits_{1 \leq i \leq k} w_i \phi_i(s) \tag{3}$$

where the coefficients w_i are constants that do not depend on the states. For example, a general, linear value function for clearing block x while having an empty gripper is $V = 2n + H$, where the states are left implicit, and the Boolean feature H has value 1 when true, 0 otherwise.

Linear value functions using description logic features (Bonet, Francès, and Geffner 2019), called generalized potential heuristics, can be learned from small instances via a mixed integer programming formulation, leading to an alternative representation of general policies that solve many of the standard planning domains (Francès et al. 2019).

Features

Logical features derived from the domain predicates using a description logic grammar have been used to define and learn *policies* of the form (1) and *value functions* of the form (3). The complexity of such features is defined in terms of the number of grammar rules required to derive them, and the pool of features used is obtained by placing a bound on the complexity of the features. An important limitation of these methods is that the pool of features grows exponentially with the complexity bound, and that some domains require complex features. For example, Francès et al. (2019) cannot learn general value functions for Logistics and Blocks because they appear to require features of complexity 22 and 49, respectively. Interestingly, the features required to express the policy rules for some of these domains is much smaller (Francès, Bonet, and Geffner 2021).

For learning general policies without using a precomputed pool of features, it turns out to be simpler and more direct to learn general value functions, and then define greedy policies from them. A first step in this direction was taken by Ståhlberg, Bonet, and Geffner (2022) where the value function V was learned in a supervised fashion using graph neural networks from optimal targets V^*. Graph neural networks have also been used in other approaches to generalized planning using deep nets (Toyer et al. 2020; Garg, Bajpai, and Mausam 2020; Rivlin, Hazan, and Karpas 2020), but in combination with other techniques and without drawing on the relation between the features that can be learned by GNNs and those that are actually needed.

Graph Neural Networks

The GNN architecture for learning value functions follows the one used by Ståhlberg, Bonet, and Geffner (2022): it accepts states s over arbitrary instances of a given planning domain, and outputs the scalar value $V(s)$. For this, the form of the general value function $V(s)$ in (2) is reformulated as:

$$V(s) \;=\; F(\phi(o_1), \ldots, \phi(o_n)) \tag{4}$$

where $o_1, \ldots, o_n$ are objects in the instance where the state s is drawn from, $\phi(o)$ is a vector of feature values associated with object o in state s (dependence on s omitted), represented as a vector of real numbers, and F is a function that aggregates these feature vectors and produces the scalar output $V(s)$. The vectors $\phi(o)$ are usually called *object embeddings* and the function F, the readout. Before revising the details of the architecture, we discuss the meaning and implication of transitioning from fully general value functions expressed as (2) to value functions expressed as (4).

From State Features to Object Embeddings

We are moving from state features to object features $\phi(o)$ that depend not just on the state s but on the objects o. In addition, the same feature function ϕ is applied to all the objects, and the same aggregation function F is applied to the states s of any of the domain instances so that the number of feature vectors $\phi(o)$ expands or contracts according to the number of objects in the instance. This is key for having a well-defined value function over the whole collection of domain instances that involve a different numbers of objects.

We can see that function (4) can capture capture value functions needed for generalized planning by comparing (4) to function (3), used by Francès et al. (2019) with features expressed in description logic. These Boolean and numerical features $b_q(s)$ and $n_q(s)$ are defined in terms of derived unary predicates q, where $b_q(s) = 1$ (true) if there is an object o such that $q(o)$ is true in s, otherwise 0; and $n_q(s) = m$ is the number of objects o for which $q(o)$ is true in s. Clearly, if the feature vectors $\phi(o_i)$ in (4) contain a bit encoding whether $q(o)$ is true in s, then the readout function F just need to take the *max* and the *sum* of the bits $q(o)$ as

$$b_q(s) \;=\; \max_o\, q(o)\,, \quad n_q(s) \;=\; \sum_o\, q(o) \tag{5}$$

in order to capture such features, where the objects o range over all the objects o in the instance. In other words, the object-embedding form (4) is no less expressive than the linear form that uses description logic features, provided that the feature vectors $\phi(o)$ are expressive enough to represent

the bits $q_i(o)$ for unary predicates q_i derived from the domain predicates using the description logic grammar. This in turn is known to be within the capabilities of standard GNNs that can capture properties expressible in the guarded fragment of the variable logic with counting C_2, which includes the standard description logics (Barceló et al. 2020).

Below we follow the terminology of graph neural networks and refer to graphs and not states, and to vertex embeddings $f(v)$ and not object embeddings $\phi(o)$. After considering standard GNNs for undirected graphs, we introduce the generalization needed for dealing with the relational structures represented by planning states.

GNNs on Graphs

GNNs represent trainable, parametric, and generalizable functions over graphs (Scarselli et al. 2008; Hamilton 2020) specified by means of aggregate and combination functions agg_i and $comb_i$, and a readout function F. For each vertex v of the input graph G, the GNN maintains a state (vector) $f_i(v) \in \mathbb{R}^k$, the vertex embedding, $i = 0, \ldots, L$, where L is the number of iterations or layers. The vertex embeddings $f_0(v)$ are fixed and the embeddings f_{i+1} for all v are computed from the f_i embeddings as:

$$f_{i+1}(v) := comb_i\big(f_i(v), agg_i\big(\{\!\{f_i(w) | w \in N_G(v)\}\!\}\big)\big) \qquad (6)$$

where $N_G(v)$ is the set of neighbors for vertex v in G, and $\{\!\{\ldots\}\!\}$ denotes a multiset. In words, the embeddings $f_{i+1}(v)$ at iteration $i+1$ are obtained by combining the aggregation of neighbors' embeddings $f_i(w)$ at iteration i with v's own embeddings $f_i(v)$. This process is usually seen as an exchange of messages among neighbor nodes in the graph. The aggregation functions agg_i map arbitrary collections of real vectors of dimension k into a single $\mathbb{R}^k$ vector. Common aggregation functions are *sum*, *max*, and *smooth-max* (a smooth approximation of the max function). The combination functions $comb_i$ map pairs of $\mathbb{R}^k$ vectors into a single $\mathbb{R}^k$ vector. The embeddings $f_L(v)$ in the last layer are aggregated and mapped into the output of the GNN by means of a readout function F. In our setting, the output will be a scalar V, and the aggregation and combination functions agg_i and $comb_i$ will be homogeneous and not depend on the layer index i. All the functions are parametrized with weights that are adjusted by minimizing a suitable loss function. By design, the function computed by a GNN is *invariant* with respect to graph isomorphisms, and once a GNN is trained, its output is well defined for any graph G regardless size.

GNNs for Planning States

States s in planning do not represent graphs but more general relational structures that are defined by the set objects, the set of domain predicates, and the atoms $p(o_1, \ldots, o_m)$ that are true in the state: the objects define the universe, the domain predicates, the relations, and the atoms, their denotations. The set of predicate symbols p and their arities are fixed by the domain, but the sets of objects o_i may change from instance to instance. The adaptation of the basic GNN architecture for dealing with planning states s follows (Ståhlberg, Bonet, and Geffner 2022), which is an elaboration of the architecture for learning to solve Max-CSP problems over a fixed class of binary relations introduced by Toenshoff et al. (2021). The new GNN still maintains just the object embeddings $f_i(o)$ for each of the objects o in the input state s, $i = 0, \ldots, L$, but now rather than messages flowing from "neighbor" objects to objects as in (6), the messages flow from objects o_i to the true atoms q in s that include o_i, $q = p(o_1, \ldots, o_m)$, $1 \leq i \leq m$, and from such atoms q to all the objects o_j involved in q as:

$$f_{i+1}(o) := comb_U\big(f_i(o), agg\big(\{\!\{m_{q,o} | o \in q, q \in s\}\!\}\big)\big) \qquad (7)$$

where $m_{q,o}$ for $q = p(o_1, \ldots, o_m)$ and $o = o_j$ is:

$$m_{q,o} := [comb_p(f_i(o_1), \ldots, f_i(o_m))]_j \,. \qquad (8)$$

In these updates, the combination function $comb_U$ takes the concatenation of two real vectors of size k and outputs a vector of size k, while the combination function $comb_p$, that depends on the predicate symbol p, takes the concatenation of m vectors of size k, where m is the arity of p, and outputs m vectors of size k as well, one for each object involved in the p-atom. The expression $[\ldots]_j$ in (8) selects the j-th such vector in the output.

The resulting trainable function that maps states s into their values $V(s)$ is shown in Algorithm 1 with all the combination functions replaced by the multilayer perceptrons (MLPs) that implement them. During the iterations $i = 0, \ldots, L$, a single $\mathbf{MLP}_U$ is used for updating the object embeddings following (6), and a single $\mathbf{MLP}_p$ per predicate is used to collect the messages from atoms to objects as in (8). The readout function, the last line in Algorithm 1, uses two MLPs and a sum aggregator. Finally, for the aggregator in line 6, we use the differentiable smooth max function $smax(x_1, \ldots, x_n)$ defined as

$$x^* + \alpha^{-1} \log\Big(\sum\nolimits_{1 \leq j \leq n} \exp(\alpha(x_j - x^*))\Big) \qquad (9)$$

where $x^* = \max\{x_1, \ldots, x_n\}$ and $\alpha = 8$.

All MLPs consists of two dense layers with the activation functions ReLU and linear, resp. The hyperparameters are the object embedding size k and the number of layers L. The initial embeddings $f_0(o)$ are obtained by concatenating a zero vector with a random vector, each of size $k/2$, to break symmetries. The GNN can be applied to any state over the domain since there is a single MLP_p for every predicate p.

Algorithm 1: GNN maps state s into scalar $V(s)$

Input: State s: set of atoms true in s, set of objects
Output: V(s)

1 $f_0(o) \sim \mathbf{0}^{k/2}\mathcal{N}(0,1)^{k/2}$ for each object $o \in s$;
2 **for** $i \in \{0, \ldots, L-1\}$ **do**
3 **for** *each atom* $q := p(o_1, \ldots, o_m)$ *true in* s **do**
   `// Msgs` $q \to o$ `for each` $o = o_j$ `in` q
4 $m_{q,o} := [\mathbf{MLP}_p(f_i(o_1), \ldots, f_i(o_m))]_j$;
5 **for** *each* o *in* s **do**
   `// Aggregate, update embeddings`
6 $f_{i+1}(o) := \mathbf{MLP}_U\big(f_i(o), agg(\{\!\{m_{q,o} | o \in q\}\!\})\big)$;
   `// Final Readout`
7 $V := \mathbf{MLP}_2\big(\sum_{o \in s} \mathbf{MLP}_1(f_L(o))\big)$

Learning the GNN Parameters

The parameters of the network displayed in Algorithm 1 are learned by stochastic gradient descent by minimizing a

loss function. In the work of Ståhlberg, Bonet, and Geffner (2022), the training data $\mathcal{D}$ is a collection of pairs $\langle s, V^*(s)\rangle$ for sampled states s from selected instances, and $V^*(s)$ is the optimal cost for reaching the goal from s (min. number of steps). The loss is the average sum of the differences

$$L(s) \;=\; |V(s) - V^*(s)| \tag{10}$$

over the states s in the training set. The computation of the optimal targets $V^*(s)$ is not a problem because we are computing them over small instances. The real problem is that by forcing the value function to be optimal over the training instances in domains such as Blocks or Miconic, where optimal planning is NP-hard (Gupta and Nau 1992; Helmert 2001). Interestingly, as discussed in the next section, this limitation also appears in unsupervised RL methods where the optimal target values $V^*(s)$ are not given but are sought by minimizing the Bellman error:

$$L'_0(s) \;=\; |V(s) - (1 + \min_{s' \in N(s)} V(s'))| \tag{11}$$

for non-goal states s, where $N(s)$ are the states reachable from s in one step (possible successor states). For goal states, $L'_0(s)$ is $|V(s)|$. The optimal function V^* is the unique value function that minimizes the resulting loss, provided that actions costs are all 1 and the goal is reachable from all states. In this work, rather than penalizing departures from the Bellman optimality equation

$$V(s) \;=\; 1 + \min_{s' \in N(s)} V(s')\,, \tag{12}$$

departures from the inequality $V(s) \geq 1 + \min_{s' \in N(s)} V(s')$ are penalized with a loss for non-goal states s defined as

$$L'_1(s) \;=\; \max\{0, (1 + \min_{s' \in N(s)} V(s')) - V(s)\}\,. \tag{13}$$

Furthermore, this loss is extended with two regularization terms that penalize large departures from V^*; namely, as done by Francès, Bonet, and Geffner (2021), we want a value function V that also satisfies $V^* \leq V \leq \delta V^*$, and thus settle for the minimization of the loss:

$$\begin{aligned} L_1(s) \;=\; L'_1(s) + \; & \max\{0, V^*(s) - V(s)\} + \\ & \max\{0, V(s) - \delta V^*(s)\}\,, \end{aligned} \tag{14}$$

where $\delta = 2$. The loss over a set $\mathcal{S}$ of states is the sum of the average of $L_1(s)$ for non-goal states $s \in \mathcal{S}$ and the average of $|V(s)|$ for goal states $s \in \mathcal{S}$. For comparison purposes, the L'_0 loss is extend into the regularized L_0 loss as well as:

$$\begin{aligned} L_0(s) \;=\; L'_0(s) + \; & \max\{0, V^*(s) - V(s)\} + \\ & \max\{0, V(s) - \delta V^*(s)\}\,. \end{aligned} \tag{15}$$

If all the states in a small instance are in $\mathcal{S}$ and the overall loss is close to zero, the loss function L_1 results in value functions that lead greedily to the goal (by picking the min-V successors), while the loss L_0 results in value functions that lead greedily and *optimally* to the goal. For simplicity, it is assumed that the domains considered do not have *dead-ends*, i.e. states from which the goal is not reachable and where $V^*(s)$ is not well-defined. Learning to plan in such domains requires an slight extension, with extra inputs, for labeling states as dead-ends in the training data, and extra outputs, for predicting if a state is a dead-end (Ståhlberg, Francès, and Seipp 2021). This extension is implemented and tested, but it will be skipped over in the presentation.

Domain	Train	Validation	Test
Blocks	[4, 7]	[8, 8]	[9, 17]
Delivery	[12, 20]	[28, 28]	[29, 85]
Gripper	[8, 12]	[14, 14]	[16, 46]
Logistics	[5, 18]	[13, 16]	[15, 37]
Miconic	[3, 18]	[18, 18]	[21, 90]
Reward	[9, 100]	[100, 100]	[225, 625]
Spanner*	[6, 33]	[27, 30]	[22, 320]
Visitall	[4, 16]	[16, 16]	[25, 121]

Table 1: Instance sizes used training, validation, and testing datasets, as measured by the number of objects involved. E.g., the training set for Blocks consists of IPC instances with a number of blocks between 4 and 7. There is no instance that is in more than 1 set.

Experiments

The experiments are aimed to test the generalization, coverage, and quality of the plans obtained by the policy π_V greedy in value functions learned using L_0 and L_1 losses. We describe the datasets used, and the results. A key difference with prior work (Ståhlberg, Bonet, and Geffner 2022) is that the test instances are standard IPC planning problems from standard planning domains, several of which are intractable for optimal planning. We seek crisp experimental results, which means close to 100% generalization, or alternatively, crisp explanations of why this is not possible, with logical fixes that restore generalization in certain cases.

Data. The states in the training and validation sets are obtained by fully expanding instances from the initial state through a breadth-first search. For each reachable state, the length of the shortest path to a goal state is computed. For instances with large (reachable) state spaces we keep up to $40,000$ randomly sampled states to avoid them from dominating the training set. The actual size of the instances used in training, validation, and testing are shown in Table 1, measured by the number of objects involved. In almost all cases, the testing instances are IPC (International Planning Competition) instances. The exception is the domain Spanner*, which is a slight variant of the Spanner domain that does not give rise to dead-end states by allowing the agent to move not just forward but also backward.

Domains. The domains are those used by Francès, Bonet, and Geffner (2021) with the addition of Logistics, and the above modification of Spanner. Briefly, Blocks is the standard blocks world. Delivery is the problem of picking up objects in an empty grid and delivering them one by one to a target cell. Gripper is about moving balls from one room to another with a robot that has two grippers. Logistics involves moving packages between locations, possibly in different cities, with trucks and airplanes. Miconic is about controlling an elevator to pick up and drop off passengers between different floors. Rewards is about reaching certain cells in a grid while avoiding others. Spanner is about collecting single-use spanners spread in a one dimensional grid, each one to be used to tighten a nut at the other end. Visitall is about visiting all or some cells in an empty grid.

Setup. The hyperparameters k and L in Algorithm 1 are set to 64 and 30, respectively: k is the number of "features" per object; i.e., the size of the real object embedding vectors; and L the number of layers in the GNN (fixed for training and testing). Both hyperparameters affect training speed, memory, and generalization. Hyperparameter L affects how far messages can propagate in the graph, and indeed, the GNN cannot capture shortest paths between two objects if longer than L, even if the existence of paths up to length $2L$ can be determined. The architecture is implemented in PyTorch (Paszke and et. al. 2019) and the optimizer Adam (Kingma and Ba 2015) was used with a learning rate of 0.0002.[1] The networks are trained with NVIDIA A100 GPUs for up to 12 hours. Five models for each domain are trained to ensure that the optimizer did not get stuck in "bad" local minima, and the final model used is the one with the best validation loss. The quality of the plans obtained by following the greedy policy π_V for the learn value function V are evaluated in comparison with optimal plans that are computed with the Fast Downward (FD) planner (Helmert 2006) using the *seq-opt-merge-and-shrink* configuration with time and memory outs set to 10 minutes and 64 GB, respectively, on a Ryzen 9 5900X CPU.

Policies. The greedy policy π_V selects the action applicable in a non-goal state s that leads to the child state s' with minimum $V(s')$ value (action costs are all assumed to be 1). It is common to add "noise" in this selection process by either breaking ties randomly or by choosing the action leading to the best child probabilistically, by soft-mapping the children values $V(s')$ into probabilities that add up to 1. This has the benefit that cycles can be avoided in the execution, but at the same time, the results are blurred. Instead, Table 2 shows (on the right) the results of the executions that follow the deterministic greedy policy π_V, which always chooses the action leading to the child s' with lowest $V(s')$ value, breaking ties for the first such action encountered. Since the learned value function is not perfect, we show on the left the execution of the greedy policy but with *cycle avoidance*; namely, executions keep track of the visited states and deterministically select the first action leading to the best *unvisited* child (min-V value). When there are no such children, the execution fails. Executions are also terminated when the goal is not reached within $1,000$ steps.

Results: L_1 Loss

Table 2 shows the results for various experiments: learning using the L_1 loss (top), learning using the L_0 loss (middle), and learning using states augmented with derived atoms in domains that benefit from C_3 features (explained below). Furthermore, the three subtables are divided horizontally in two, according to the way in which the greedy policy π_V for the learned value function V is used: with cycle avoidance, on the left, and without cycle avoidance, on the right. We focus on the top part of the table first.

Coverage. The first thing to notice is that in 4 out of the 8 domains considered, Blocks, Delivery, Gripper, and Miconic, the deterministic greedy policy π_V for the learned value function V solves all the test instances. This is remarkable as the resulting plans are often long. In Blocks, the average plan length is $790/20 = 39.5$ steps, while in Miconic, it is $7,331/120 = 61.09$. As we will see, the quality of these plans is very good, and moreover, in none of these cases, the deterministic greedy policy generates an execution where a state is revisited. Indeed, if revisits are explicitly excluded by executing the greedy policy while avoiding cycles (left), a fifth domain is solved in full: Visitall. The other three domains are not solved in full in either mode: Logistics, Reward, and Spanner. The network lacks expressive power to represent some features for Logistics, and the distances involved in Reward and Spanner exceed the number of layers L; we discuss these limitations in detail later. We note that other domains solved in full also need distances to be computed, for these domains the magnitude of the distances needed in the test instances is within this bound (L).

Quality. Somewhat surprisingly, the quality of the executions delivered by the models trained with the L_1 loss is very close to optimal, as measured with respect to the optimal plans computed by FD. The only exception is the Logistics domain where plans are up to 10 times longer than optimal, on average. These results are surprising not just because the L_1 loss does not force the value function V to be optimal, but because optimal planning in several of these domains, Blocks, Miconic, and Logistics, and possibly in Reward and Visitall as well, is NP-hard (Gupta and Nau 1992; Helmert 2001). For example, FD with the given time and memory bounds computes optimal solutions for 35 instances in Miconic comprising a total of 1,164 actions, while the sum of execution lengths for the learned, greedy policy π_V with or without cycle avoidance on the same 35 instances is 1,170. Indeed, the execution lengths that follow from the learned value function do not exceed the optimal plan lengths in more than 12% with the exception of Logistics.

L_0 Loss: General Policies and RL

The differences between the L_1 loss (14) and the L_0 loss (15) are small but significant. Zero loss for L_0 arises just when the learned V function has zero Bellman error over the training set; i.e. when $V(s) = 1 + \min_{s' \in N(s)} V(s')$ for the possible children s' of s, and thus when V is the optimal cost function V^*. Zero loss for L_1, on the other hand, arises just when the learned V function is such that $V(s) \geq 1 + \min_{s' \in N(s)} V(s')$. Thus, zero L_0 loss implies zero L_1 loss, but not the other way around, as the L_1 loss captures just one half of Bellman's optimality equation. Provided that only the goal states have zero value and that non-goal states have positive values, one can use a value function V with zero L_1 loss to solve problems *greedily* by always moving to the best child (min V). On the other hand, a value function V with zero L_0 loss can be used in the same manner to solve problems greedily and *optimally*. The difference between solving a class of problems optimally or suboptimally is crucial in domains where optimal planning is NP-hard: there are often suboptimal but no optimal general policies.

The question is whether minimizing L_0 loss leads to

[1] Code and data: https://doi.org/10.5281/zenodo.6511809

	Deterministic policy π_V with cycle avoidance			Deterministic policy π_V alone		
Domain (#)	Coverage (%)	L	PQ = PL / OL (#)	Coverage (%)	L	PQ = PL / OL (#)
			L_1 Loss			
Blocks (20)	20 (100%)	790	1.0427 = 440 / 422 (13)	20 (100%)	790	1.0427 = 440 / 422 (13)
Delivery (15)	15 (100%)	400	1.0000 = 400 / 400 (15)	15 (100%)	404	1.0100 = 404 / 400 (15)
Gripper (16)	16 (100%)	1,286	1.0000 = 176 / 176 (4)	16 (100%)	1,286	1.0000 = 176 / 176 (4)
Logistics (28)	17 (60%)	4,635	9.7215 = 3,665 / 377 (15)	0 (0%)	0	—
Miconic (120)	120 (100%)	7,331	1.0052 = 1,170 / 1,164 (35)	120 (100%)	7,331	1.0052 = 1,170 / 1,164 (35)
Reward (15)	11 (73%)	1,243	1.2306 = 1,062 / 863 (10)	3 (20%)	237	1.1232 = 237 / 211 (3)
Spanner*-30 (41)	30 (73%)	1,545	1.0000 = 1,545 / 1,545 (30)	24 (58%)	940	1.0000 = 940 / 940 (24)
Visitall (14)	14 (100%)	904	1.0183 = 556 / 546 (10)	11 (78%)	631	1.0107 = 471 / 466 (9)
Total (269)	243 (90%)	18,134	1.6410 = 9,014 / 5,493 (132)	209 (77%)	11,619	1.0156 = 3,838 / 3,779 (103)
			L_0 Loss			
Blocks (20)	0 (0%)	0	—	0 (0%)	0	—
Delivery (15)	12 (80%)	278	1.0000 = 278 / 278 (12)	12 (80%)	278	1.0000 = 278 / 278 (12)
Gripper (16)	16 (100%)	1,288	1.0000 = 176 / 176 (4)	12 (75%)	816	1.0000 = 176 / 176 (4)
Logistics (28)	1 (3%)	134	16.7500 = 134 / 8 (1)	0 (0%)	0	—
Miconic (120)	120 (100%)	7,758	1.0241 = 1,192 / 1,164 (35)	108 (90%)	6,438	1.0000 = 1,084 / 1,084 (33)
Reward (15)	12 (80%)	1,362	1.1226 = 861 / 767 (9)	7 (46%)	661	1.0285 = 505 / 491 (6)
Spanner*-30 (41)	24 (58%)	1,221	1.0374 = 1,221 / 1,177 (24)	14 (34%)	475	1.0000 = 475 / 475 (14)
Visitall (14)	14 (100%)	838	1.0073 = 550 / 546 (10)	12 (85%)	664	1.0073 = 550 / 546 (10)
Total (269)	199 (73%)	12,879	1.0719 = 4,412 / 4,116 (95)	165 (61%)	9,332	1.0059 = 3,068 / 3,050 (79)
			Derived Atoms (L_1 Loss)			
Logistics-atoms (28)	28 (100%)	8,147	5.5711 = 2,546 / 457 (17)	4 (14%)	88	1.0353 = 88 / 85 (4)
Spanner*-10 (36)	12 (33%)	557	1.0000 = 557 / 557 (12)	8 (22%)	373	1.0000 = 373 / 373 (8)
Spanner*-atoms-5 (36)	31 (86%)	1,370	1.0000 = 1,112 / 1,112 (27)	28 (77%)	1,190	1.0000 = 996 / 996 (25)
Spanner*-atoms-2 (36)	36 (100%)	1,606	1.0000 = 1,348 / 1,348 (32)	36 (100%)	1,606	1.0000 = 1,348 / 1,348 (32)
Total (136)	107 (78%)	11,680	1.6013 = 5,563 / 3,474 (88)	76 (55%)	3,257	1.0011 = 2,805 / 2,802 (69)

Table 2: Performance of the deterministic greedy policy π_V for the learned value function V when executed with cycle avoidance (left) and without (right). The top subtable contains results when using the L_1 loss, middle when using the L_0 loss, and bottom when using L_1 loss when states are extended with derived atoms (encoding role compositions and transitive closures). The domains are shown on the left with the number of instances tested in each. Coverage is the number of solved problems. L is the sum of the solution lengths over the test instances solved by the learned policy. PQ is a measure of overall plan quality given by the ratio of the sum of the plan lengths found by the policy (PL) and the optimal plan lengths (OL) found by FD, over the instances solved by both within the time and memory limits (number of such problems shown after OL in parenthesis).

greedy policies π_V that are as good as or better than those obtained by minimizing L_1 loss. The question is particularly relevant because the standard methods for learning policies without supervision are usually based on reinforcement learning, which in their value-based variant (as opposed to the policy gradient version) are based on the minimization of Bellman error (Sutton and Barto 2018). The expectation is that the minimization of L_0 loss will not be as good. Indeed, the value functions V that yield greedy policies π_V that generalize correctly over domains that are intractable for optimal planning are unlikely to yield zero L_0 loss.

The middle part of Table 2 shows the results of the greedy policies π_V for value functions V learned by minimizing L_0 loss instead of L_1. The L_0-based policies are observed to perform worse than the L_1-based policies. The extreme case is precisely in Blocks where coverage drops from 100% to 0% when using the greedy policy with cycle avoidance and also without. A big difference also surfaces in Logistics where coverage drops from 60% to 3% with cycle avoidance (otherwise no instances are solved). For the other domains, the drops are not as drastic, yet the greedy policy with no cycle avoidance based on L_1 solves four domains fully (100% coverage) while the same policy based on L_0 does not solve fully any single domain. The L_0-policies, however, do slightly better in two of the domains where the L_1-policy is not good: Reward and Visitall where coverage increases from 20% and 78% to 46% and 86%. As expected, the lower coverage of L_0-policies goes along with executions whose lengths are better overall. With cycle avoidance, the performance resulting from the two loss functions is closer, with the exception mentioned for Blocks and Logistics. In general, the ability of the learned value functions V to yield greedy policies that generalize can be predicted from the corresponding loss on the validation set. In both Blocks and Logistics, the validation loss after L_1 training is close to zero, but significantly higher than zero after L_0 training.

Derived Atoms: Beyond C_2

The learned policies failed to generalize in the domains Logistics, Reward, and Spanner*, this can be tracted to two limitations. Logistics requires features that cannot be expressed in C_2 and which therefore are not captured by GNNs (Barceló et al. 2020; Grohe 2020). Spanner*, like Reward and other domains, involves the computation of distances in the test instances that exceed the number of layers used in the GNN. The bottom part of Table 2 shows the results that are obtained in Logistics and Spanner* when these limitations are addressed logically by extending the states (in training, validation, and testing) with suitable derived atoms and predicates, a facility provided by PDDL (Thiébaux, Hoffmann, and Nebel 2005; Haslum et al. 2019b). For example, one can extend the states in Blocks with the derived predicate $above$ that corresponds to the transitive closure of the domain predicate on, so that every state s contains additional atoms $above(x, y)$ when block x is above block y in s.

In Logistics, four derived predicates are added, following the four role compositions used by Francès et al. (2019) to obtain a general value function. In Logistics, there are binary predicates to express that a package or truck is at some location ('at'), to express that a package is inside a truck or airplane ('in'), and to express that a location is in a city ('in-city'). Additionally, as done in previous works, "goal versions" of these predicates denoted by 'at@', 'in@' and 'in-city@' whose denotation is provided by the goal descriptions are added to the domain. The Logistics domain is extended with the following role compositions from Francès et al. (2019): 'at $\circ$ in-city' and 'at@ $\circ$ in-city' that tells the city where a package is located, either in the current or goal state; 'in $\circ$ at' that tells the location of a package that is inside a truck; and 'in $\circ$ at $\circ$ in-city' that tells the city where a package that is inside a truck is located.

In Spanner*, a single derived predicate is added which is the transitive closure of the 'link' predicate. Provided with the new link$^+$ predicate, the required distances in Spanner* are not restricted by the number of layers L in the GNN and can be computed in a single layer, as the distance to the gate location equals the number of locations to the right of the current location c; i.e., dist2gate $= |\{x \mid \text{link}^+(c, x)\}|$.

The results obtained by learning from states with these derived predicates in Logistics and Spanner* are shown at the bottom of Table 2. The simple addition of the atoms makes the coverage jump from from 0% to 14% for the greedy policy alone, and from 60% to 100% for the greedy policy with cycle avoidance. For Spanner*, three rows are shown: the first is for the domain without derived atoms but with two modifications that preclude comparison with the Spanner* results reported previously in the same table. The first is that the test instances involving more than 100 locations have been replaced by smaller instances with up to 45 and 50 locations. The second is that the number of layers L in the GNN are reduced from 30 to 10. These modifications provide a more convenient baseline for evaluating the impact of derived atoms: with 100 locations, there are $10,000 = 100^2$ extra derived atoms in the states, that make training and testing much slower. It is because of these modifications, and in particular from the reduction in the value of L from 30 to 10, that the coverage of the learned policies in the modified Spanner* setting is reduced to 33% and 22% percent (first of the last three rows in the table). This number however increases to 86% and 77% when the derived atoms are included, even if the number of GNN layers is reduced from 10 to 5 (second of the last three rows in table). Moreover, coverage increases further to 100% when the derived atoms are included and the number of GNN layers is reduced further to just 2 (last row in the table). This additional increase in coverage is probably due by reduced overfitting as the number of layers L is reduced from 5 to 2.

Conclusions

We have considered the problem of learning generalized policies for classical planning domains from small instances represented in lifted STRIPS. Unlike previous work that makes uses of a predefined pool of features based on description logic and combinatorial solvers, we have followed the GNN approach for learning general policies advanced by Ståhlberg, Bonet, and Geffner (2022) that exploits the relation between C_2 features and those that can be computed by GNNs. However, instead of learning optimal value functions in a supervised manner, we learn non-optimal value functions without supervision. For this, the change is technically small, as it affects the loss function and not the GNN architecture, but the consequences are interesting as the new method can be applied to domains that have general policies but no general policies that are optimal. We have shown that 100% generalization is achieved in many such domains, and have discussed and addressed two important additional issues: the limitations of value-based RL methods for computing general policies over domains where optimal planning is intractable, and the limitations of GNNs for capturing general value functions that require non-C_2 features. We have addressed the first limitation by using a novel loss function (L_1), and the second limitation by extending planning states with derived atoms. In the future, we would like to make the point about the limitations of RL methods for learning generalized plans more sharp, and to consider the use of recent GNN architectures that compute features beyond C_2 (Bevilacqua et al. 2021). We are interested in bridging the gap between deep learning and representation learning for planning, so that DL can be used as a more scalable alternative to ASP and Weighted Max-SAT solvers. This requires understanding what can be computed with them in a clean way and how, and this work is also a step in that direction.

Acknowledgments

This research was partially supported by the European Research Council (ERC), Grant No. 885107, and by project TAILOR, Grant No. 952215, both funded by the EU Horizon research and innovation programme. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. 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

Asai, M. 2019. Unsupervised Grounding of Plannable First-Order Logic Representation from Images. In *Proc. ICAPS*.

Barceló, P.; Kostylev, E. V.; Monet, M.; Pérez, J.; Reutter, J.; and Silva, J. P. 2020. The logical expressiveness of graph neural networks. In *ICLR*.

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

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control, Vols 1 and 2*. Athena Scientific.

Bevilacqua, B.; Frasca, F.; Lim, D.; Srinivasan, B.; Cai, C.; Balamurugan, G.; Bronstein, M. M.; and Maron, H. 2021. Equivariant subgraph aggregation networks. *arXiv preprint arXiv:2110.02910*.

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.

Bonet, B.; and Geffner, H. 2020. Learning first-order symbolic representations for planning from the structure of the state space. In *Proc. ECAI*.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic Derivation of Memoryless Policies and Finite-State Controllers Using Classical Planners. In *Proc. ICAPS-09*, 34–41.

Campero, A.; Raileanu, R.; Kuttler, H.; Tenenbaum, J. B.; Rocktäschel, T.; and Grefenstette, E. 2021. Learning with AMIGo: Adversarially Motivated Intrinsic Goals. In *ICLR*.

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 *ICLR*.

Cobbe, K.; Hesse, C.; Hilton, J.; and Schulman, J. 2020. Leveraging procedural generation to benchmark reinforcement learning. In *Proc. ICML*, 2048–2056.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2): 195–213.

Fern, A.; Yoon, S.; and Givan, R. 2006. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *JAIR*, 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.

Francès, G.; Corrêa, A. B.; Geissmann, C.; and Pommerening, F. 2019. Generalized potential heuristics for classical planning. In *Proc. IJCAI*.

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

Garg, S.; Bajpai, A.; and Mausam. 2020. Symbolic network: generalized neural policies for relational MDPs. In *Proc. ICML*.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated planning and acting*. Cambridge U.P.

Grohe, M. 2020. The Logic of Graph Neural Networks. In *Proc. of the 35th ACM-IEEE Symp. on Logic in Computer Science*.

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

Gupta, N.; and Nau, D. S. 1992. On the complexity of blocks-world planning. *AIJ*, 56(2-3): 223–254.

Hamilton, W. L. 2020. Graph representation learning. *Synth. Lect. on Artif. Intell. Mach. Learn.*, 14(3): 1–159.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019a. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019b. An Introduction to the Planning Domain Definition Language. *Synth. Lect. on Artif. Intell. Mach. Learn.*, 13(2): 1–187.

Helmert, M. 2001. On the complexity of planning in transportation domains. In *Proc. ECP*.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR*, 26: 191–246.

Hu, Y.; and De Giacomo, G. 2011. Generalized planning: Synthesizing plans that work for multiple environments. In *Proc. IJCAI*, 918–923.

Karia, R.; and Srivastava, S. 2021. Learning generalized relational heuristic networks for model-agnostic planning. In *AAAI*.

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

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

Martín, M.; and Geffner, H. 2004. Learning generalized policies from planning examples using concept languages. *Appl. Intell.*, 20(1): 9–19.

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.

Paszke, A.; and et. al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché Buc, F.; Fox, E.; and Garnett, R., eds., *Adv. Neural Inf. Process. Syst. 32*, 8024–8035. Curran Associates, Inc.

Rivlin, O.; Hazan, T.; and Karpas, E. 2020. Generalized planning with deep reinforcement learning. *arXiv preprint arXiv:2005.02305*.

Rodriguez, I. D.; Bonet, B.; Romero, J.; and Geffner, H. 2021. Learning First-Order Representations for Planning from Black-Box States: New Results. In *KR*. ArXiv preprint arXiv:2105.10830.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80.

Segovia, J.; Jiménez, S.; and Jonsson, A. 2016. Generalized planning with procedural domain control knowledge. In *Proc. ICAPS*, 285–293.

Shen, W.; Trevizan, F.; and Thiébaux, S. 2020. Learning domain-independent planning heuristics with hypergraph networks. In *Proc. ICAPS*, volume 30, 574–584.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2008. Learning generalized plans using abstract counting. In *Proc. AAAI*, 991–997.

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

Ståhlberg, S.; Francès, G.; and Seipp, J. 2021. Learning Generalized Unsolvability Heuristics for Classical Planning. In *Proc. IJCAI*, volume 4, 4175–4181.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT Press.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *AIJ*, 168(1-2): 38–69.

Toenshoff, J.; Ritzert, M.; Wolf, H.; and Grohe, M. 2021. Graph neural networks for maximum constraint satisfaction. *Front. Artif. Intell. Appl.*, 3: 98.

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *JAIR*, 68: 1–68.

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