Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits

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

[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

It has been recently shown that general policies for many classical planning domains can be expressed and learned in terms of a pool of features defined from the domain predicates using a description logic grammar. At the same time, most description logics correspond to a fragment of k-variable counting logic (C_k) for $k = 2$, that has been shown to provide a tight characterization of the expressive power of graph neural networks. In this work, we make use of these results to understand the power and limits of using graph neural networks (GNNs) for learning optimal general policies over a number of tractable planning domains where such policies are known to exist. For this, we train a simple GNN in a supervised manner to approximate the optimal value function $V^*(s)$ of a number of sample states s. As predicted by the theory, it is observed that general optimal policies are obtained in domains where general optimal value functions can be defined with C_2 features but not in those requiring more expressive C_3 features. In addition, it is observed that the features learned are in close correspondence with the features needed to express V^* in closed form. The theory and the analysis of the domains let us understand the features that are actually learned as well as those that cannot be learned in this way, and let us move in a principled manner from a combinatorial optimization approach to learning general policies to a potentially, more robust and scalable approach based on deep learning.

Introduction

Deep learning (DL) and deep reinforcement learning (DRL) are behind most of key milestones in AI of recent years (Mnih et al. 2015; LeCun, Bengio, and Hinton 2015; Silver et al. 2017a,b). Yet, these methods struggle to produce solutions that are structurally general (Goyal and Bengio 2020). Even in simple tasks, such as retrieving a key to open a door in a simple environment, they may require a large number of simulations, and even then, they may fail to generalize to all possible situations (Chevalier-Boisvert et al. 2019). Interestingly, the computation of general policies has been addressed recently in a model-based setting that assumes that a general model of the actions is known in terms of action schemas and predicates (Bonet and Geffner 2018; Francès, Bonet, and Geffner 2021). This paper is a step aimed at bringing these threads together with two motivations: to replace the combinatorial methods that have been proposed to learn general policies by more robust and scalable deep learning methods, and to do so in a principled manner where the intermediate representations and experimental results, both positive and negative, can be understood.

For this, we exploit two existing results. On the one hand, the realization that general policies and value functions for many classical benchmark domains can be expressed in terms of features defined from the domain predicates using a description logic (DL) grammar (Martín and Geffner 2004; Fern, Yoon, and Givan 2006; Bonet, Francès, and Geffner 2019; Francès et al. 2019). On the other, the correspondence established between the expressive power of a decidable fragment of first-order logic, called C_2, which includes most common DLs (Baader, Horrocks, and Sattler 2008), and the expressive power of graph neural networks (GNNs) (Barceló et al. 2020; Grohe 2020). The two results together suggest that general policies could be learned from the domain predicates directly by means of GNNs, except for those which are not expressible in terms of C_2 features at all.

In this paper, we carry out this exploration in a context where the learned general policies are expected to be *optimal*, leading to optimal (shortest) plans in any instance of the target class of problems $\mathcal{Q}$. In addition, instead of seeking for representations of an optimal policy, we seek representations of the optimal value function. If this function V is optimal, the policy π_V greedy in V is optimal as well. The focus on optimal values allows us to learn the general function V using labeled data in the form of pairs $\langle s, V^*(s)\rangle$, and to evaluate the learned function V in a crisp manner where *the execution of a non-optimal transition at any state is an error.* The optimality requirement is thus a methodological choice which simplifies the training and evaluation procedures in order to determine whether the graph neural networks manage to learn the value functions that can be expressed in terms of C_2 features without having to make explicit the feature pool or the underlying grammar. Recent works have used deep learning methods for addressing similar problems in the broader setting of stochastic MDPs (Toyer et al. 2020; Garg, Bajpai, and Mausam 2020). Our approach is inspired by them and follows on their footsteps, but it is not so much focused on performance but on understanding the scope of the methods and the features learned.

The rest of the paper is organized as follows. We review classical planning, general policies and value functions, and present value functions for a number of tasks in terms of logical features, most in C_2. We review GNNs, their relation to finite-variable logics, and define the architecture used for learning value functions. We report the experiments and analyze results, discuss related work, and conclude.

Classical Planning

A classical planning instance is a pair $P = \langle D, I \rangle$ where D is a first-order planning *domain* and I is *instance* information (Geffner and Bonet 2013; Ghallab, Nau, and Traverso 2016). The planning 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 information is a tuple $I = \langle O, Init, Goal \rangle$ where O is a (finite) set of object names c_i, and $Init$ and $Goal$ are sets of *ground atoms* $p(c_1, \ldots, c_k)$. This is the structure of planning problems expressed in PDDL (Haslum et al. 2019) where the domain and instance information are provided in separate files.

A classical problem $P = \langle D, I \rangle$ encodes a state model $S(P) = \langle S, s_0, S_G, Act, A, f \rangle$ 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 $i = 0, \ldots, n$, 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.

The representation of planning problems P in two parts D and I, one that is general, and the other that is specific, is essential for defining and computing general policies, as the instances are assumed to come all from the same domain. Recent work has addressed the problem of learning such first-order representations from unstructured data (Asai 2019; Bonet and Geffner 2020; Rodriguez et al. 2021).

General Policies and Value Functions

Generalized planning studies the representation and computation of policies that solve many classical planning instances from the same domain at once (Srivastava, Immerman, and Zilberstein 2008; Bonet, Palacios, and Geffner 2009; Hu and De Giacomo 2011; Belle and Levesque 2016).

For example, $\mathcal{Q}_{clear}$ consists of all classical problems in Blocksworld where a block x must be cleared, regardless of the number or initial configuration of blocks, and a general policy for $\mathcal{Q}_{clear}$ 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 the target block x, by means of the rules $\{\neg H, n > 0\} \mapsto \{H, n\downarrow\}$ and $\{H\} \mapsto \{\neg H\}$ (Bonet and Geffner 2018). The first rule says that when the gripper is empty and there are blocks above x, any action that decreases n and makes H true should be selected. The second that when the gripper is not empty, any action that makes H false and does not affect n should be selected. It has been shown that general policies of this form can be learned without supervision by solving a Max-Weighted SAT theory $T(\mathcal{S}, \mathcal{F})$ where $\mathcal{S}$ is a set of sampled state transitions, and $\mathcal{F}$ is a large but finite pool of Boolean and numerical features obtained from the domain predicates (Francès, Bonet, and Geffner 2021).

In this work, it is convenient to represent policies in terms of *value functions*. As it is usual in dynamic programming and RL (Sutton and Barto 1998; 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 of the same cost. A policy π solves an instance P if the state transitions compatible with π, starting with the initial state, always end up in a goal state, and π solves a class of problems $\mathcal{Q}$ if it solves each problem in the class. Similarly, π solves P and $\mathcal{Q}$ *optimally* when goals are always reached optimally. Clearly, if V is optimal, i.e., V is the optimal cost function V^*, the greedy policy π_V is optimal too. The general value functions are defined over general features ϕ_i, which are well-defined state functions over the states arising in instances of $\mathcal{Q}$ as:

$$V(s) = F(\phi_1(s), \ldots, \phi_k(s))$$

where $\phi_i(s)$ is the value of the feature ϕ_i in state s. Value functions that are *linear* have the form:

$$V(s) = \textstyle\sum_{1 \leq i \leq k} w_i \phi_i(s)$$

where the coefficients w_i are constants that do not depend on the states. For example, the general value function for the collection of problems $\mathcal{Q}_{clear}$, assuming different actions for picking and placing objects, is:

$$V = 2n + H$$

where the states are left implicit, and the Boolean feature H is assumed to have value 1 when true, and 0 otherwise (the opposite for $\neg H$). This value function is optimal for $\mathcal{Q}_{clear}$. In planning, two types of linear value functions that have been used are "potential heuristics" (Pommerening et al. 2015), that are instance-dependent and use features that stand for conjunction of atoms, and "generalized potential heuristics" (Francès et al. 2019) that use the Boolean and the numerical description logic features introduced by Bonet, Francès, and Geffner (2019).

Domain Predicates, Features, and Logics

A key problem in reinforcement learning (RL), and in particular in RL with linear function approximation, is the choice of the features (Wu and Givan 2010; Geramifard et al. 2013; Song et al. 2016; Bellemare et al. 2019). A relevant observation made early on is that the features can often be defined using a simple DL grammar from the domain predicates (Martín and Geffner 2004; Fern, Yoon, and Givan 2006). For example, if $q(z)$ and $r(x, y)$ are two domain predicates of arities 1 and 2 respectively, one can define new unary predicates $p_1(x)$ and $p_2(x)$ as $\exists y[q(y) \wedge r(x, y)]$ and $\forall y\, r(x, y)$, and use the new unary predicates to define new ones, etc.

Unary predicates p can be used to define numerical features n_p, whose value is the number of objects that satisfy p in a state s, and Boolean features b_p, whose value is true when n_p is greater than 0 (Bonet, Francès, and Geffner 2019).

Interestingly, most variants of DLs are parts of a fragment of first-order logic (FOL) known as FO_2, that involves just two variables, such as x and y above (Baader, Horrocks, and Sattler 2008). In other words, DLs can express some FO-formulas that make use of two variables but not three. The extension of FO_k, with k variables, with counting quantifiers $\exists^{\geq i}$ to express that there are at least i *different* objects that comply with a formula is the logic C_k.

The relation between the features required for expressing general policies and value functions, and the finite-variable logics required to express such features is relevant as it has been shown recently that *guarded* C_2 (GC_2), which corresponds to a standard description logic, provides a tight characterization of the expressive power of (message passing) graph neural nets (Barceló et al. 2020; Grohe 2020). This suggests that GNNs can be used to learn general policies using the domain predicates without having to generate a pool of C_2 features by assuming some fixed grammar and a bound on the complexity of the features. *This is the hypothesis that we explore in this work by focusing on the problem of learning general value functions that are optimal.*

Value Functions for Tractable Tasks

We consider *optimal value functions* for a number of tasks and domains, selected mostly from Lipovetzky and Geffner (2012), where they are shown to be solvable, optimally, in polynomial time, suggesting that a compact optimal value function may exist. All features are defined in terms of the domain predicates, and for simplicity, they are all Boolean. Predicates p_G refer to predicates p evaluated in the goal (goal predicates); i.e., while an atom like $\textsc{On}(a, b)$ is true or false in a state, the atom $\textsc{On}_G(a, b)$ is true in a state iff it is true in the goal of the instance (Martín and Geffner 2004).

In these domains, the optimal value functions are linear and expressed as sums $V^*(s) = \sum_{i=1}^{N} c_i [\![\varphi_i(s)]\!]$, where c_i is a constant and $[\![\varphi_i]\!]$ is the Iverson bracket that evaluates to 1 if feature φ_i holds in s, and else to 0. The formulas φ_i all belong to the logic C_k for $k = 2$, and in one case, for $k = 3$.

In many domains, we need features that reflect the existence of paths of length k from object x to some object y such that some condition $T(y)$ holds, where objects are connected by "edges" $E(x, y)$. This can be expressed in C_2 as:

$$\mathrm{P}_0(x) = \mathrm{T}(x)\,,$$
$$\mathrm{P}_k(x) = \exists y(\mathrm{E}(x, y) \wedge \mathrm{P}_{k-1}(y))\,.$$

The distance of a shortest path of length k is then captured by $\mathrm{SP}_k(x) = \mathrm{P}_k(x) \wedge \neg \mathrm{P}_{k-1}(x)$, while the existence of such path of length up to N is captured by $\mathrm{CONN}_N(x) = \mathrm{P}_0(x) \vee \cdots \vee \mathrm{P}_N(x)$. The constant N is related to a hyperparameter L in the architecture, to be described below. Notation $\mathrm{SP}_k[\mathrm{T}', \mathrm{E}']$ (resp. $\mathrm{CONN}_N[\mathrm{T}', \mathrm{E}']$) denotes SP_k (resp. CONN_N) where T/E are replaced by T'/E' resp. Additionally, $\mathrm{P}^{\mathbb{1}}(x) = \exists y\, \mathrm{P}(x, y)$ and $\mathrm{P}^{2}(x) = \exists y\, \mathrm{P}(y, x)$ denote that x appears as first and second argument of some atom. Lastly, $\mathrm{P}^{-1}(x, y)$ holds iff $\mathrm{P}(y, x)$ holds.

Blocksworld: Clear and On. Gupta and Nau (1992) showed that finding an optimal solution for Blocksworld is **NP**-hard. We thus consider a tractable version where the goal $\textsc{Clear}(x)$ is to clear a specific block x. The optimal value function decomposes as

$$V^* = [\![\alpha \wedge H]\!] + \textstyle\sum_{k=1}^{N} (2k-1)[\![B_k]\!]$$

for features

$$\alpha = \exists x(\textsc{Clear}_G(x) \wedge \neg \textsc{Clear}(x))\,,$$
$$H = \exists x\, \textsc{Holding}(x)\,,$$
$$B_k = \exists x(\textsc{Clear}_G(x) \wedge \eta_k(x))\,,$$
$$\eta_k(z) = \mathrm{SP}_k[\textsc{Clear}, \textsc{On}^{-1}](z)$$

where α holds in a state, if it is not a goal state, H, if holding some block, and B_k (resp. $\eta_k(z)$), if there are k blocks above x (resp. z), $k \leq N$.

The other version of Blocksworld corresponds to instances with single-atom goals of the form $\textsc{On}(x, y)$ for some x and y. In this case, the optimal value function for problems with up to N blocks above x or y decomposes as

$$V^* = [\![\alpha \wedge H]\!] + 2[\![\alpha \wedge L]\!] + 2\textstyle\sum_{k=1}^{N} k([\![X_k]\!] + [\![Y_k]\!])$$

for features

$$\alpha = \exists xy(\textsc{On}_G(x, y) \wedge \neg \textsc{On}(x, y)),$$
$$L = \exists xy(\textsc{On}_G(x, y) \wedge (\neg \textsc{Clear}(y) \vee \neg \textsc{Holding}(x))),$$
$$X_k = \exists xy(\textsc{On}_G(x, y) \wedge \eta_k(x) \wedge \neg \mathrm{CONN}_N[\textsc{On}^{2}_G, \textsc{On}](x)),$$
$$Y_k = \exists xy(\textsc{On}_G(x, y) \wedge \eta_k(y) \wedge \neg \mathrm{CONN}_N[\textsc{On}^{\mathbb{1}}_G, \textsc{On}](y))$$

where L holds if x is not held or cannot be stacked on y, and X_k (resp. Y_k) holds if there are k blocks above x (resp. y) and x (resp. y) is not above y (resp. x).

Gripper. There is a robot with two grippers, and a set of rooms containing balls. While the goal is to move every ball to the correct room, we consider the subproblem of moving a single ball, whose goal is just $\textsc{At}(x, y)$ for some ball x and room y. The optimal value function is

$$V^* = [\![\alpha \wedge P]\!] + 3[\![\alpha \wedge B]\!] + [\![\alpha \wedge D]\!] + 2[\![\alpha \wedge G]\!] + [\![\alpha \wedge F]\!]$$

for features

$$\alpha = \exists xy(\textsc{At}_G(x, y) \wedge \neg \textsc{At}(x, y))\,,$$
$$P = \exists xy(\textsc{At}(x, y) \wedge \textsc{At}^{\mathbb{1}}_G(x) \wedge \textsc{At-robby}(y))\,,$$
$$B = \exists xy(\textsc{At}(x, y) \wedge \textsc{At}^{\mathbb{1}}_G(x) \wedge \neg \textsc{At-robby}(y))\,,$$
$$D = \exists xy(\textsc{At}_G(x, y) \wedge \textsc{At-robby}(y))\,,$$
$$G = \exists xy(\textsc{At}_G(x, y) \wedge \neg \textsc{At-robby}(y))\,,$$
$$F = \exists xy(\textsc{At}^{\mathbb{1}}_G(x) \wedge \textsc{At}(x, y) \wedge \neg \exists x(\textsc{Gripper}(x) \wedge \textsc{Free}(x)))\,.$$

where α holds when the goal is not achieved, P (resp. B) holds when Robby is (resp. is not) in the same room as the ball x and Robby should pick up x (resp. move to pick it up), D (resp. G) holds when Robby is (resp. is not) in the room y, and F holds when no gripper is free and Robby is not carrying the ball x. It is important to note that when Robby picks up a ball, the ball is no longer in any room.

Transport. The task is to deliver packages using trucks of bounded capacity. We consider a version where the goal is an atom $\textsc{At}(x, y)$ for package x and destination y. The optimal value function V^* decomposes as the sum for $1 \leq k \leq N$ of

$$(k+1)\big([\![\alpha \wedge T_k]\!] + [\![\alpha \wedge D_k]\!] + [\![\alpha \wedge D'_\ell]\!]\big) + [\![\alpha \wedge T_k \wedge F_k]\!]$$

for features

$$\begin{aligned}
\alpha &= \exists xy(\textsc{At}_G(x,y) \wedge \neg \textsc{At}(x,y))\,,\\
\beta(x,y) &= \textsc{At}(x,y) \wedge \textsc{At}^{\mathbb{1}}_G(x)\,,\\
\gamma(x,y) &= \textsc{In}(x,y) \wedge \textsc{At}^{\mathbb{1}}_G(x)\,,\\
L(y) &= \exists x(\textsc{Vehicle}(x) \wedge \textsc{At}(x,y))\,,\\
C(y) &= \exists x(\textsc{Vehicle}(x) \wedge \textsc{At}(x,y) \wedge\\
&\qquad \exists y(\textsc{Capacity}(x,y) \wedge\\
&\qquad\quad \exists x\, \textsc{Capacity-Predecessor}(x,y)))\,,\\
T_k &= \exists xy(\beta(x,y) \wedge \textsc{SP}_k[L, \textsc{Road}^{-1}](y))\,,\\
F_k &= \exists xy(\beta(x,y) \wedge \neg\textsc{SP}_k[C, \textsc{Road}^{-1}](y))\,,\\
D_k &= \exists xy(\beta(x,y) \wedge \textsc{SP}_k[\textsc{At}^{\mathbb{2}}_G, \textsc{Road}](y))\,,\\
D'_k &= \exists xy(\gamma(x,y) \wedge\\
&\qquad \exists x(\textsc{At}(y,x) \wedge \textsc{SP}_k[\textsc{At}^{\mathbb{2}}_G, \textsc{Road}](x)))
\end{aligned}$$

where α holds iff the goal is not achieved, T_k determines the distance to the closest truck, $T_k \wedge F_k$ determines if all closest trucks are full and need to drop a package, and D_k (resp. D'_ℓ) determines the distance from the package (resp. truck the package is in) to the destination.

Rovers. Multiple rovers equipped with different capabilities (soil analysis, etc) must perform experiments and send results back to lander. A simple version where just the soil at some specific location must be sampled is considered.

A feature for identifying the closest available and capable rover to sample soil is needed. Since each rover has its own map, shortest-path on different graphs must be considered:

$$\begin{aligned}
\textsc{P}_0(r,x) &= \textsc{At-soil-sample}(x)\,,\\
\textsc{P}_k(r,x) &= \exists y(\textsc{Can-traverse}(r,x,y) \wedge \textsc{P}_{k-1}(r,y))\,,\\
\textsc{SP}_k(r,x) &= \textsc{At}(r,x) \wedge \textsc{P}_k(r,x) \wedge \neg\textsc{P}_{k-1}(r,x)\,.
\end{aligned}$$

In addition to find the distance to the closest capable rover to sample the soil R_k, features are need to decide if such rover is full F_k, if the soil has not been sampled S, and if the goal has not been achieved α. Also needed are Boolean features L_k to express the distance from the soil or rover with the sample to some location where data can be sent to lander. The optimal value function decomposes as:

$$V^* = [\![\alpha]\!] + [\![S]\!] + \textstyle\sum_{k=1}^{N}(k[\![R_k]\!] + [\![F_k]\!] + k[\![L_k]\!])\,.$$

The formulas $\textsc{SP}_k(r,x)$ enter into the definition of the features R_k. Since these formulas involve 3 variables, the features used to decompose V^* do not belong to C_2 or GC_2.

Visitall. The task is to find a path that starts at an initial vertex and visits all vertices in a given graph. The simplified version involves a single target vertex to be visited. For graphs with up to N vertices, the optimal value function is

$$V^* = \textstyle\sum_{k=1}^{N} k[\![\alpha \wedge D_k]\!]$$

for features

$$\begin{aligned}
\alpha &= \exists x(\textsc{Visited}_G(x) \wedge \neg\textsc{Visited}(x))\,,\\
D_k &= \exists x(\textsc{At-robot}(x) \wedge \textsc{SP}_k[\textsc{Visited}_G, \textsc{Connected}](x))\,.
\end{aligned}$$

Other Domains. Logistics, Miconic, Parking-behind and Parking-curb, and Satellite are also considered in the experiments. We do not have space to discuss them in detail; however, the goals of all these problems are single atoms and optimal plan lengths are bounded by (small) constants (also in Gripper). On the other hand, for the two versions of Blocksworld, Transport, Rovers and Visitall the length of optimal plans is not bounded.

Learning The Value Functions

We turn to the problem of learning these value functions using GNNs directly from the domain predicates. For this, we review GNNs, their logic, and the GNN architecture used.

Graph Neural Networks

GNNs represent trainable, parametric functions over graphs (Scarselli et al. 2008; Hamilton 2020). We focus on aggregate-combine GNNs (AC-GNNs) (Barceló et al. 2020; Grohe 2020) with L layers that are specified with aggregate functions $\mathbf{agg}_i$, combination functions $\mathbf{comb}_i$, and a classification function CLS. On input graph G, a GNN maintains a state (vector) $\boldsymbol{x}_v \in \mathbb{R}^k$ for each vertex $v \in V(G)$, and computation consists of updating these states throughout L stages, with $\boldsymbol{x}_v^{(i)}$ denoting the states after stage i. The parameter k is the dimension of the node state or embedding. The computation model for AC-GNNs corresponds to updates

$$\boldsymbol{x}_v^{(i)} \;:=\; \mathbf{comb}_i\big(\boldsymbol{x}_v^{(i-1)}, \mathbf{agg}_i\big(\{\!\!\{\boldsymbol{x}_w^{(i-1)} \mid w \in N_G(v)\}\!\!\}\big)\big)$$

where $N_G(v)$ is the set of neighbors for vertex v in G, and $\{\!\!\{\ldots\}\!\!\}$ denotes a multiset (i.e., unordered set whose elements are associated with multiplicities). That is, at stage i, each vertex v receives the state of its neighbors which are then aggregated, and the result combined with the current state $\boldsymbol{x}_v^{(i-1)}$ to produce the next state $\boldsymbol{x}_v^{(i)}$. The fact that $\mathbf{agg}_i$ maps multisets of states into real vectors means that it does not depend on the source of the received messages. GNNs are used for node or graph classification. In the first case, after the final stage, the node v is classified into the class $\text{CLS}(\boldsymbol{x}_v^{(L)})$ determined by a function $\text{CLS} : \mathbb{R}^k \to \{0,1\}$. In the second case, the CLS function maps the multiset $\{\!\!\{\boldsymbol{x}_v^{(L)} \mid v \in V(G)\}\!\!\}$ into a single, scalar output; an operation referred to as a *readout*.

In our case, GNNs map planning states s into real values $V(s)$. However, the atoms in a planning state do not induce a graph (or hypergraph), but the more subtle relational structure. Therefore, we adapt below the GNN architecture to deal with relational structures. In any case, the functions involved in the mapping from inputs to outputs can be linear or non-linear, and they are all trainable; in the supervised case, by minimizing a loss function defined over a training set given by pairs $\langle s, V^*(s)\rangle$, where all the states s (sets of atoms) come from instances of different size but over a common planning domain and common set of goal predicates.

The Logic of GNNs

The expressive power of AC-GNNs has been recently studied in relation to decidable fragments of first-order logic (Barceló et al. 2020; Grohe 2020). For this, it is convenient to consider the more general vertex-colored graphs G and to assume that the AC-GNNs for such graphs initialize the embeddings of the vertices $\boldsymbol{x}_v^{(0)}$ to a one-hot encoding of the vertex colors. One of the first crisp results for node classification is that if the Weisfeiler-Lehman (WL) procedure, a well-known coloring algorithm that provides a sound but incomplete test for graph isomorphism (Lehman and Weisfeiler 1968), assigns the same color to two nodes in a graph, then every AC-GNN classifier will map the two nodes into the same class (Xu et al. 2018; Morris et al. 2019).

This result has been extended in two ways: one, where the WL procedure is replaced by the logic C_2, making use of a seminal result relating the two (Cai, Fürer, and Immerman 1992), and the second, where the characterization of the expressive power of AC-GNNs is made tight, describing not just what they can compute, but also what they cannot (Barceló et al. 2020). For this, the logical formulas considered are those that involve equality and two types of predicates: a binary edge $E(x, y)$ predicate representing the edges in the graph, and unary predicates $c_i(x)$ representing the color of vertices. A (Boolean) node classifier can be expressed then as a logical formula $\varphi(x)$ over these predicates with a single free variable x. The question is what is the relation between the node classifiers that can be captured in an AC-GNNs and those that can be described logically.

The logical classifiers that can be captured by AC-GNNs are fully characterized in terms of *graded modal logic* GC_2, which is equivalent in expressive power to the standard description logic $\mathcal{ALCQ}$ (Barceló et al. 2020). GC_2 is the class of all formulas in C_2 in which each quantified variable is *guarded* by the edge relation; e.g., $\psi(x) = \exists y\,[E(x, y) \wedge \mathsf{blue}(y)]$ that holds when x has a blue neighbor. The main result is:

Theorem 1 (Barceló et al., 2020)**.** *A logical classifier is captured by AC-GNNs if and only if it can be expressed in graded modal logic (GC_2), or equivalently, in the description logic $\mathcal{ALCQ}$.*

Moreover, each GC_2 classifier can be captured by a simple and homogeneous AC-GNN; i.e., with linear combinators, and combinators and aggregators that are identical across all layers. There is a similar result for C_2 classifiers, but this requires an slightly modified version of AC-GNNs, called ACR-GNNs, where the combination function for each vertex v is extended to take an extra argument given by an aggregation of the states for all vertices in the graph:

Theorem 2 (Barceló et al., 2020)**.** *Logical node classifiers in C_2 are captured by simple and homogeneous ACR-GNNs.*

GNNs for Relational Structures

In as much as truth valuations give meaning to propositional formulas, relational structures give meaning to first-order formulas. In the case of planning states, the induced relational structures only have relations, and do not involve constants nor functions. Thus, a relational structure $\mathcal{R} = (\mathfrak{D}, R_1^{\mathfrak{D}}, \ldots, R_m^{\mathfrak{D}})$ consists of a domain of interpretation $\mathfrak{D}$ and relations $R_i^{\mathfrak{D}}$ of arity k_i that stand for sets of k_i-tuples from $\mathfrak{D}$. In the relational structure defined by a graph, $\mathfrak{D}$ is given by the vertices and there is a single relation $R^{\mathfrak{D}}$ given by the edges. In the structure defined by a planning state s, $\mathfrak{D}$ is given by the set of objects in the instance, and $R_i^{\mathfrak{D}}$ is the set of object tuples that satisfy the predicate R_i in s.

Our modification of GNNs to handle relational structures is inspired by the one introduced by Toenshoff et al. (2021) for solving Max-CSP problems where all relations are assumed to be binary and thus any such Max-CSP instance maps straightforwardly to a directed graph. A major difference though is that our architecture does not make use of LSTMs (Hochreiter and Schmidhuber 1997).

For dealing with relations of any arity, the computation maintains states $\boldsymbol{s}_o^{(i)}$ for each object $o \in \mathfrak{D}$ and proceeds in stages $i = 1, \ldots, L$, where each atom $p = R(o_1, \ldots, o_n)$ computes messages $\boldsymbol{m}_{p,o_i}$ that are sent to each object o_i. Each object o then aggregates the incoming messages $\boldsymbol{m}_{p,o}$ from the atoms p that mention o, and combines this aggregation with the current state $\boldsymbol{s}_o^{(i-1)}$ to produce the next state $\boldsymbol{s}_o^{(i)}$. The final state (object) vectors are passed through a neural net, aggregated, and the result passed again to a final network to produce a single output vector $\boldsymbol{v}$ of dimension q. For relational structures that capture a state s, the output $\boldsymbol{v}$ is aimed to approximate the scalar function $V^*(s)$, and hence the output dimension is $q = 1$.

The architecture shown in Algorithm 1 uses one feed-forward neural net $\mathbf{MLP}_R$ for each relational symbol R (domain and goal predicate), one such net $\mathbf{MLP}_U$ as a combination function, and two nets $\mathbf{MLP}_1$ and $\mathbf{MLP}_2$ for constructing the final output $\boldsymbol{v}$.[1] All MLPs consists of a dense layer with a ReLU activation function, followed by a dense layer with a linear activation function. For the aggregation function $\mathbf{agg}$, we use either sum or smooth maximum (implemented as LogSumExp). The trainable parameters are thus the trainable parameters in the MLPs, while the hyperparameters are the embedding dimension k, the output dimension q, and the number of stages L. The initial embeddings $\boldsymbol{s}_o^{(0)}$ are obtained by concatenating the zero vector $\mathbf{0}$ and a random vector $\mathcal{N}(0, 1)$, each of dimension $k/2$ (Aboud et al. 2021; Sato, Yamada, and Kashima 2021).

The parameters of the network are learned by stochastic gradient descent by minimizing the loss $\mathcal{L}(\mathcal{R}, \boldsymbol{\ell}) = \|\boldsymbol{v} - \boldsymbol{\ell}\|_1$ from training data $\{(\mathcal{R}_i, \boldsymbol{\ell}_i)\}_i$. In our setting, the relational structures $\mathcal{R}_i$ encode (the atoms that are true in) the states s, and the target value $\boldsymbol{\ell}_i$ for s is $V^*(s)$.

Experiments

We now evaluate if models (neural nets) can be trained and used as policies in the domains and tasks considered above.

[1] Another major difference is that the messages $\boldsymbol{m}_{p,o}$ sent to objects, line 4 in Alg. 1, are computed with MLPs whereas in the architecture of Toenshoff et al. (2021) the messages are computed with linear transforms.

Algorithm 1: General architecture (trainable, parametric function) that maps relational structures $\mathcal{R} = (\mathfrak{D}, R_1^{\mathfrak{D}}, \ldots, R_m^{\mathfrak{D}})$ into vector $\boldsymbol{v}$. In our setting, $\mathcal{R}$ encodes the states s, and $\boldsymbol{v}$ approximates $V^*(s)$. Atoms $p(o_1, \ldots, o_n)$ true in the input send messages to the objects o_i in p, and objects o aggregate all messages received and update their state $\boldsymbol{s}_o^{(i)}$.

Input: Relational struct. $\mathcal{R} = (\mathfrak{D}, R_1, \ldots, R_m)$ [states s]
Output: $\boldsymbol{v} \in \mathbb{R}^q$ of dimension q [value $V(s)$]

// Partial random initialization
$\boldsymbol{s}_o^{(0)} \sim \mathbf{0}^{k/2}\mathcal{N}(0,1)^{k/2}$ for each object $o \in \mathfrak{D}$;
for $i \in \{1, \ldots, L\}$ **do**
  for *atom* $p := R(o_1, \ldots, o_n)$ *with* $\bar{o} \in R$ **do**
    // Generate messages $p \to o_j$
    $(\boldsymbol{m}_{p,o_j})_j := \mathbf{MLP}_R(\boldsymbol{s}_{o_1}^{(i-1)}, \ldots, \boldsymbol{s}_{o_n}^{(i-1)})$;
  for $o \in O$ **do**
    // Aggregate messages and update
    $\boldsymbol{s}_o^{(i)} := \mathbf{MLP}_U\big(\boldsymbol{s}_o^{(i-1)}, \mathbf{agg}(\{\!\!\{\boldsymbol{m}_{p,o} \mid o \in p\}\!\!\})\big)$;
// Final Readout
$\boldsymbol{v} := \mathbf{MLP}_2\big(\sum_{o \in \mathfrak{D}} \mathbf{MLP}_1(\boldsymbol{s}_o^L)\big)$

Domain	Train	Validation	Test
Blocks-clear	[2, 9]	[10, 11]	[12, 17]
Blocks-on	[2, 9]	[10, 11]	[12, 17]
Gripper	[10, 18]	[20, 22]	[24, 48]
Logistics	[17, 24]	[31, 31]	[31, 39]
Miconic	[5, 26]	[29, 35]	[38, 92]
Parking-behind	[21, 27]	[30, 30]	[30, 36]
Parking-curb	[21, 27]	[30, 30]	[30, 36]
Rovers	[15, 52]	[53, 62]	[67, 116]
Satellite	[14, 41]	[47, 59]	[50, 103]
Transport	[14, 39]	[38, 43]	[41, 77]
Visitall	[27, 102]	[102, 146]	[171, 326]

Table 1: Number of objects in the problems in the training, validation and test datasets; e.g., each problem for Miconic in the validation set has a number of objects in $[29, 35]$.

We first describe how states are sampled and labeled, then the experimental setup, and finally, the results.[2]

Data. For a set of instances, we sample and label states for each as follows. First, we perform a single random walk $s_1, \ldots, s_n$ from the initial state. Then, for each $1 \leq i \leq n$, we construct a planning problem with initial state s_i, and find an optimal plan $s'_1, \ldots, s'_m$ with A* using the admissible h_{max} heuristic (Bonet and Geffner 2001). For each $1 \leq j \leq m$, we add the pair $\langle s'_j, m - j \rangle$ to the dataset, up to $40,000$ such pairs, balancing the number of states per label (distance). The value of n is set to produce that many pairs if possible.

Setup. The hyperparameters k and L are set to 32 and 30, respectively; k affects the number of features per object, but also training speed and memory usage. The domain with the most predicates is Rovers with 32 predicates, so the value for k ensure that at least one feature (scalar) per predicate is possible. Our architecture can find shortest paths of length up to $2L$. In the experiments, we evaluate nets with sum- and (smooth) max-aggregation denoted by GNN-SUM and GNN-MAX, respectively. The architecture is implemented in PyTorch (Paszke and et. al. 2019) and each net is trained with NVIDIA A100 GPUs for up to 12 hours. GNN-SUM is trained with L1 regularization set to 0.0001, and no regularization for GNN-MAX (resulted in the lowest loss on the validation set). Training is done with Adam (Kingma and Ba 2015) with a learning rate of 0.0002.

Table 1 shows the number of objects for the problems in the training, validation and test datasets. We trained 5 networks for each domain, and for each training session, the net with the best validation loss at the end of each epoch is selected. Among the 5 trained nets, the final net is the one with the best validation loss. For the learned V function, we run the policy π_V, selecting from each non-goal state s, the successor s' with least V-value, breaking ties by selecting the first such successor. This is repeated for at most 100 steps, or until a goal state is reached. In the latter case, the problem is solved, and if the number of steps is minimal (verified with A* and h_{max}), *the problem is counted as solved optimally.*

		GNN-SUM		GNN-MAX	
Domain (#)	L	Opt.	Sub.	Opt.	Sub.
Blocks-clear (11)	82	**11**	0	**11**	0
Blocks-on (11)	150	**11**	0	**11**	0
Gripper (39)	117	31	8	**39**	0
Logistics (8)	48	5	3	**8**	0
Miconic (95)	378	**95**	0	**95**	0
Parking-behind (32)	77	**32**	0	**32**	0
Parking-curb (32)	101	7	12	**32**	0
Rovers (26)	111	0	4	**20**	6
Satellite (20)	97	**20**	0	**20**	0
Transport (20)	208	18	1	**20**	0
Visitall (12)	93	**12**	0	**12**	0
Total (306)	1,462	242 (79%)	28 (9%)	300 (98%)	6 (2%)

Table 2: Number of problems in test set solved optimally, suboptimally, or not solved at all with policy π_V for learned V, when aggregation is done by SUM or MAX. Total number of problems (#) shown in parenthesis. Tasks and domains from Section 4. L is the sum of all optimal plan lengths.

Results. As it is shown in Table 2, the value functions learned with GNN-MAX yield policies that *solve all of the 306 test instances, 98% of them optimally*. The 6 instances not solved optimally are all in Rovers, that as shown above, requires C_3 features. This is a pretty impressive result that shows that deep nets can produce very crisp results. In our case, it means that the GNN-MAX nets *deliver policies that do not make a single mistake in the plans of 300 test problems, and this means, practically no errors in the 1,462 intermediate decisions made in the construction of these plans*.

[2]Code and data: https://doi.org/10.5281/zenodo.6353140

Domain	#	Train $\mathcal{L}'$	Test $\mathcal{L}'$
Blocks-clear	2	0.12	0.16
Blocks-on	6	0.88	0.96
Blocks-on-Σ	5	0.17	0.23
Gripper	5	0.04	0.11
Transport	4	0.71	1.13
Transport-Σ	3	0.30	0.48
Visitall	1	0.06	1.91

Table 3: Total loss $\mathcal{L}'$ of hand-crafted features over the train and test set, and the number of such features. Features are taken from the analysis of each domain (Section 4). Domains with Σ replaces two numerical features by their sum.

Notice that this is different than simply measuring "coverage" (number of problems solved) where policies are allowed to make mistakes, if they are not fatal, and typically "noise" is introduced to prevent being trapped in cycles. In terms of the aggregation functions, the performance of GNN-SUM is not as good as GNN-MAX. The theory does not help us to understand this difference, but it has been noted that max-aggregation is better suited for discrete decisions and tasks that involve shortest paths (Veličković et al. 2020).

Understanding the Learned Features

We also tested if the learned features in the trained models can be understood in terms of the hand-crafted features used in our analysis of the domains. For this, let y be the vector of n features based on our formal analysis, where distance features SP are treated as numerical features. The readout function consists of a sequence of layers: (1) ReLU; (2) linear; (3) summation; (4) ReLU; and (5) linear. Let x be the concatenation of all intermediate feature vectors after aggregation in the readout function, i.e., the results of layer (3), (4) and (5). Finally, let $y' = xA + b$ be a linear function of x optimized such that the linear coefficients A and b minimize the loss $\mathcal{L}'(y', y) = \sum_{i=1}^{n} |y'_i - y_i|$. If this is loss is zero or very small, it means that the learned features encode a linear transformation of the hand-crafted features.

Table 3 shows the loss on the test set, after A and b are optimized on the training set. In Visitall, this loss is largest among the domains considered (those for which V^* was given in compact form), 1.91, and yet the optimal coverage is 100%, meaning that the distances are ordered well but not linearly. The loss for Blocks-on and Transport over the training set is roughly 0.8 and this suggests that the networks do not learn one or more of the hand-crafted features well, although it turns out that they learn a suitable aggregation of them. The features for Blocks-on-Σ in the Table 3, replace the two numerical features induced by X and Y in Blocks-on by their sum, and the same is done for Transport-Σ for D and D'. The training and test losses then drop to roughly 20% of the previous loss in Blocks-on, and to 40% in Transport, implying that these features are learned instead.

Understanding the Limitations

The neural network does not approximate well the optimal value function in Rovers, which is the only domain where the optimal policy does not generalize 100% with max aggregation. The problem is that optimal policies for Rovers require C_3 features that cannot be computed with standard GNNs. Interestingly, the analysis reveals that this limitation is not due to the presence of multiple rovers, but to multiple rovers with their *own maps*. For illustrating this, we designed a simplified Rovers domain called Vacuum: an assortment of robot vacuums that have to clean a specific spot. The predicates of this domain are AT/2, DIRTY/1, and ADJACENT/3, and each robot r can clean a location x and move to an adjacent location y if $\textsc{Adjacent}(r, x, y)$. We consider three different versions: Vacuum-R with at most 5 robots, Vacuum-M where all robots share the same traversal map, and Vacuum with no restrictions. We generated 20 problems of each version of Vacuum and ensured that optimal plan lengths vary from approximately 3-8 for the training set, 6-9 for the validation set, and 6-12 for the test set. The number of problems solved optimally by GNN-MAX is 1 for Vaccum, 4 for Vaccum-R, and 20 for Vaccum-M. The only version with 100% generalization (or close) is Vacuum-M, which is precisely the version of the domain where there are C_2 features for deciding the length of shortest paths. The r argument in $\textsc{Adjacent}(r, x, y)$ is indeed redundant, and if $\textsc{Adjacent}'$ denotes the resulting binary predicate, the optimal value function for Vacuum-M decomposes as

$$V^* = \textstyle\sum_{k=1}^{N} (k+1)[\![D_k]\!]$$

for $D_k = \exists x(\textsc{Dirty}_G(x) \wedge \textsc{SP}_k[\textsc{At}^2, \textsc{Adjacent}'^{-1}](x))$, which is a C_2 feature.

Related Work

Neuro Symbolic AI. Many proposals have been advanced for integrating symbolic and DL approaches due to limitations and opacity of pure data-based approaches (Lake et al. 2017; Manhaeve et al. 2021; Lamb et al. 2020). Our integration combines domain predicates, that can potentially be learned (Asai 2019; Bonet and Geffner 2020; Rodriguez et al. 2021), builds on the correspondences between finite variable logics and GNNs (Barceló et al. 2020; Grohe 2020), and modifies the architecture for Max-CSPs. Interestingly, recent GNN methods can compute more general functions that are not limited to those defined on the C_2 features associated with DLs logics only (Abboud et al. 2021).

General Policies. The problem of learning general policies has been addressed using combinatorial approaches where the symbolic domains are given (Khardon 1999; Martín and Geffner 2004; Bonet, Francès, and Geffner 2019; Francès, Bonet, and Geffner 2021), DL approaches where the domains are given too (Toyer et al. 2020; Garg, Bajpai, and Mausam 2020), and DRL approaches that do not make use of prior knowledge about the structure of either domains or states (Groshev et al. 2018; Chevalier-Boisvert et al. 2019; Campero et al. 2021). This work is a step to bring the first two approaches together along with their potential benefits.

General Value Policies. It is known since the 1950s that a value function V defines a policy π_V which is optimal if V is optimal (Bellman 1957; Bertsekas 1995; Sutton and Barto 1998). Linear value functions have been particularly important in RL until the advent of deep RL methods that dispense with the need for hand-crafted features (Mnih et al. 2015; François-Lavet et al. 2018). In classical planning, linear value functions have been used under the name of "potential heuristics" (Pommerening et al. 2015), where the features are conjunctions of atoms, and "generalized potential heuristics" (Francès et al. 2019), where the (C_2) features are the Boolean and numerical features based on DLs (Bonet, Francès, and Geffner 2019). A "descending and dead-end avoiding potential function" V represents indeed a value function V that defines a greedy policy π_V that solves a problem. The proposed learning method provides crisp experimental evidence that generalized value functions with C_2 features can be computed without having to *explicate the pool of features and without having to assume a linear combination.* Our focus on optimal value functions is methodological: it allows for supervised learning with V^* targets, and a crisp evaluation (no single mistake allowed in the execution of plans). The same learning approach can be used in stochastic MDPs where the targets V^* represent optimal expected costs to the goal. Also, due to the correspondence between C_2 features and GNNs, the same architecture can be used for learning value functions without supervision (Francès et al. 2019), possibly using RL methods.

Summary

Previous works have shown that general policies and value functions for many classical planning domains can be expressed in terms of a pool of features that is obtained from the domain predicates using a DL grammar, and learned without supervision using combinatorial solvers. In this work, we have exploited the relations between DLs and the decidable fragment C_2 of FOL, and between GNNs and C_2, to approach a similar problem (optimal policies and value functions) but avoiding the grammar, the complexity bounds, and the combinatorial solvers that have been replaced by more robust and scalable deep learning engines.

Other authors have addressed the problem of learning general policies using GNNs and GNN-like architectures given the domain descriptions. What distinguishes our approach is that our deep learning architecture is simple and general; a modification of a GNN architecture introduced for solving a completely different task: Max-CSPs over binary constraints (Toenshoff et al. 2021). We also have a logical characterization of what are we trying to learn and we have used it to understand the scope of the computational model (power and limits), and what is actually learned. Recent extensions of GNN learning, however, suggest that (value) functions of features that are more complex than those associated with DLs could be learned effectively as well.

Acknowledgments

This work was partially supported by ERC Advanced Grant no. 885107, EU ICT-48 2020 project TAILOR (no. 952215) and by the Knut and Alice Wallenberg (KAW) Foundation under the WASP program. The computations were enabled by the supercomputing resource Berzelius provided by National Supercomputer Centre at Linköping University and the Knut and Alice Wallenberg foundation.

References

Abboud, R.; Ceylan, I. I.; Grohe, M.; and Lukasiewicz, T. 2021. The Surprising Power of Graph Neural Networks with Random Node Initialization. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*.

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

Baader, F.; Horrocks, I.; and Sattler, U. 2008. *Handbook of Knowledge Representation*, chapter Description Logics. Elsevier.

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.

Bellemare, M.; Dabney, W.; Dadashi, R.; Ali Taiga, A.; Castro, P. S.; Le Roux, N.; Schuurmans, D.; Lattimore, T.; and Lyle, C. 2019. A geometric perspective on optimal representations for reinforcement learning. *Advances in neural information processing systems*, 32: 4358–4369.

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

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control, Vols 1 and 2*. 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. 2001. Planning as Heuristic Search. *Artificial Intelligence*, 129(1–2): 5–33.

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.

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.

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*.

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. *Foundations and Trends in Machine Learning*, 11(3-4): 219–354.

Garg, S.; Bajpai, A.; and Mausam. 2020. Symbolic network: generalized neural policies for relational MDPs. In *International Conference on Machine Learning*, 3397–3407.

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

Geramifard, A.; Walsh, T. J.; Tellex, S.; Chowdhary, G.; Roy, N.; and How, J. P. 2013. *A tutorial on linear function approximators for dynamic programming and reinforcement learning*. Now Publishers.

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

Goyal, A.; and Bengio, Y. 2020. Inductive biases for deep learning of higher-level cognition. *arXiv preprint arXiv:2011.15091*.

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. *Artificial Intelligence*, 56: 223254.

Hamilton, W. L. 2020. *Graph representation learning*. Morgan & Claypool Publishers.

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

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780.

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

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 Bengio, Y.; and LeCun, Y., eds., *Proceedings of the 3rd International Conference on Learning Representations*.

Lake, B.; Ullman, T.; Tenenbaum, J.; and Gershman, S. 2017. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40.

Lamb, L. C.; Garcez, A.; Gori, M.; Prates, M.; Avelar, P.; and Vardi, M. 2020. Graph neural networks meet neural-symbolic computing: A survey and perspective. *arXiv preprint arXiv:2003.00330*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature*, 521(7553): 436.

Lehman, A.; and Weisfeiler, B. Y. 1968. Reduction of a graph to a canonical form and an algebra which appears in the process. *NTI Ser*, 2(9): 12–16.

Lipovetzky, N.; and Geffner, H. 2012. Width and serialization of classical planning problems. In *Proc. ECAI*, 540–545.

Manhaeve, R.; Dumančić, S.; Kimmig, A.; Demeester, T.; and De Raedt, L. 2021. Neural probabilistic logic programming in DeepProbLog. *Artificial Intelligence*, 298: 103504.

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

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529.

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 *AAAI*, 4602–4609.

Paszke, A.; and et. al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, 8024–8035.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From non-negative to general operator cost partitioning. In *Proc. AAAI*.

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.

Sato, R.; Yamada, M.; and Kashima, H. 2021. Random Features Strengthen Graph Neural Networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining*.

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.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2017a. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv preprint arXiv:1712.01815*.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017b. Mastering the game of go without human knowledge. *Nature*, 550(7676): 354.

Song, Z.; Parr, R. E.; Liao, X.; and Carin, L. 2016. Linear feature encoding for reinforcement learning. In *Advances in Neural Information Processing Systems*, 4224–4232.

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

Sutton, R.; and Barto, A. 1998. *Introduction to Reinforcement Learning*. MIT Press.

Toenshoff, J.; Ritzert, M.; Wolf, H.; and Grohe, M. 2021. Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3: 98.

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

Veličković, P.; Ying, R.; Padovano, M.; Hadsell, R.; and Blundell, C. 2020. Neural Execution of Graph Algorithms. In *Proceedings of the 8th International Conference on Learning Representations*.

Wu, J.-H.; and Givan, R. 2010. Automatic induction of Bellman-error features for probabilistic planning. *Journal of Artificial Intelligence Research*, 38: 687–755.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How Powerful are Graph Neural Networks? In *ICLR*.