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

ICAPS 2022

Introduction

- Deep learning (DL) and deep reinforcement learning (DRL) are behind many recent breakthroughs in AI and quickly replaced many previous approaches
- Interestingly, DL and DRL have yet to get the same foothold in planning
- We present a simple extension to graph neural networks (GNNs) so that planning domains and states can easily be accommodated
- We learn optimal policies that generalize properly to much larger instances

Introduction

- The expressive power of GNNs is well-understood: its expressive power is bound by k-variable logic, C_k, where $k = 2$
- C_k is a fragment of first-order logic where formulas are written using at most k different variables
- Since we extend GNNs, then its expressive power is likely the same, however, we do not formally prove this

Introduction

- Conveniently, we can express C_k formulas over the predicates of the domain
- We define policies with C_k and show that the experiments align with the stated expressive power:
  - If $k \leq 2$, then we can learn an optimal policy that generalize properly
  - If $k \geq 3$, then we cannot learn such policy
- This means that the capabilities of the presented architecture are well-understood

Classical Planning

- A *classical planning instance* consists of two parts: a *domain* and a *problem*
- The *domain* contains problem-independent information: *predicates* and *actions*
- The *problem* contains a set of *objects*, an *initial state*, and a *goal*
- We exploit this separation to define a network that can be used for all possible problems in the domain
- An instance implicitly defines a *transition system* where states are *ground atoms*
- We want to find an optimal (shortest) path in the transition system from the initial state to a state that contains all ground atoms in the goal

Policies and Value Functions

- We want to learn policies that can solve a classes of problems *optimally*
- We define policies based on *value functions*:
  - Value functions predict the unit distance to the closest goal state
  - Greedily follow successors with the smallest value
- We can only learn such a value function if its features can be expressed in C_2

Shortest Paths and C_2

Distances can be computed with 2 variables as follows:

$$
\begin{aligned}
Path_k(x) &= T(x),\ k \geq 0 \\
Path_k(x) &= \exists y(E(x,y) \wedge Path_{k-1}(y)) \\
Shortest\text{-}Path_k(x) &= Path_k(x) \wedge \neg Path_{k-1}(x)
\end{aligned}
$$

where $Path_k(x)$ defines the existence of a path from x to y such that $T(y)$ holds and can be reached in at most k steps; and $Shortest\text{-}Path_k(x)$ determines if the shortest path is k long.
We use the notation $Shortest\text{-}Path_k[T', E']$ to substitute T and E with T' and E', respectively.

Blocks-Clear

For Blocks-Clear, we define the following Boolean features:

$$\begin{aligned}
\alpha &= \exists x(\textsc{Clear}_G(x) \wedge \neg\textsc{Clear}(x)) \\
H &= \exists x\textsc{Holding}(x) \\
B_k &= \exists x(\textsc{Clear}_G(x) \wedge \textit{Shortest-Path}_k[\textsc{Clear}, \textsc{On}^{-1}](x))
\end{aligned}$$

We can define an optimal value function as:

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

For each block above the "goal block", we perform two actions: pick it up and put it down. We do not need to put the very last block down. The integer N is a hyperparameter; arbitrarily long shortest distances cannot be determined.

Rovers

In Rovers, we want to find the closest available and capable rover to the soil to be sampled, however, each Rover has its own traversal map so we cannot use the previous definition of *Shortest-Path*. In this domain, shortest path is defined as follows:

$$\begin{aligned} Path_k(r,x) &= \textsc{At-Soil-Sample}(x),\ k \geq 0 \\ Path_k(r,x) &= \exists y(\textsc{Can-Traverse}(r,x,y) \wedge Path_{k-1}(r,y)) \\ Shortest\text{-}Path_k(r,x) &= \textsc{At}(r,x) \wedge Path_k(r,x) \wedge \neg Path_{k-1}(r,x) \end{aligned}$$

Note that $\textsc{Can-Traverse}$ is a ternary predicate and requires 3 variables distinct variables; this means C_2 is insufficient to define an optimal value function for Rovers.

Neural Network Architecture

- Graph neural networks (GNNs) are neural models that capture dependencies by sending messages between nodes
- Let (V, E) be a graph. Then, for each edge $(v_1, v_2) \in E$, up two two messages are sent: one to v_1, and one to v_2
- We extend GNNs so that for each ground atom $p(v_1, \ldots, v_n)$, n messages are sent, one to each v_i
- Since the number of predicates is fixed, messages are sent by a specific MLP_p

Neural Network Architecture

Algorithm 1: General architecture that outputs a scalar value v for a given state s.

Input: A a set of ground atoms s (state and goal atoms) over a set of objects $\mathcal{O}$

Output: A scalar value v

```
1 $f_0(o) \sim 0^{k/2}\mathcal{N}(0,1)^{k/2}$ for each object $o \in \mathcal{O}$;
2 for $i \in \{0, \dots, L-1\}$ do
3     for $q := p(o_1, \dots, o_m) \in s$ do
          // Msgs $q \rightarrow o$ for each $o = o_j$ in $q$
4         $m_{q,o} := [\mathrm{MLP}_p(f_i(o_1), \dots, f_i(o_m))]_j$;
5     for $o \in \mathcal{O}$ do
          // Aggregate, update embeddings
6         $f_{i+1}(o) := \mathrm{MLP}_U(f_i(o), \mathrm{agg}(\{\!\{ m_{q,o} | q \in s \}\!\}))$;
7 $v = \mathrm{MLP}_2(\Sigma_{o \in O}\mathrm{MLP}_1(f_L(o)))$
```

Neural Network Architecture

Algorithm 2: General architecture that outputs a scalar value v for a given state s.

Input: A a set of ground atoms s (state and goal atoms) over a set of objects $\mathcal{O}$

Output: A scalar value v

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

Neural Network Architecture

- The networks are trained in a *supervised* fashion
- Let $V(s)$ be the value predicted by the network, and $V^*(s)$ be the target value
- The loss function is $L(s) = |V(s) - V^*(s)|$

Experiments

- In this work, we are interested in *optimal policies*
- Finding *optimal solutions* is *NP-hard* for many domains
- For such domains, unless P $=$ NP, our networks cannot learn an optimal policy
- We get around this by simplifying the goal of every instance: we only consider instances with a single atom in the goal
- The architecture works well with several atoms in the goal (KR'22)

Experiments

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: 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 $[29, 35]$.

Experiments

		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: 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. L is the sum of all optimal plan lengths.

Experiments

- Rovers is the only domain we did not get 100% coverage, even so, we got a surprisingly high coverage (20 opt. and 6 subopt. of 26)
- There was always an available and capable rover close to the dirt to be sampled
- The main issue with the domain Rovers is not due to the number of rovers, but that they have their own traversal map

Experiments

- We designed a new domain *Vacuum* to show, experimentally, that Rovers does not generalize properly since its rover has its own traversal map
- We consider three different versions of this domain:
  - *Vacuum*: Arbitrarily many robots with different traversal maps
  - *Vacuum-R*: There are at most 5 robots with different traversal maps
  - *Vacuum-M*: Arbitrarily many robots with the same traversal map
- The only variation where we could learn a proper policy was *Vacuum-M*

Conclusion

- We considered the problem of learning generalized policies for classical planning domains, learned from small instances in lifted STRIPS
- We presented an architecture, an extension of GNNs, such that domain descriptions and states can easily be accommodated
- Although not theoretically shown, the expressive power is understood
- We can determine whether a domain can possibly be learned by checking if C_2 is sufficient to express a value function
- Instances with more than one atom in the goal works as well (KR'22)
- Suboptimal policies can also be learned (KR'22)

Thanks for listening!