Learning Generalized Policies Without Supervision Using GNNs

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

[1]Linköping University, Sweden

[2]Universitat Pompeu Fabra, Spain

[3]ICREA, Barcelona, Spain

Introduction

- This is continuation of previous work appeared at ICAPS-22:
  - ▷ Presented neural network architecture for classical planning based on GNNs
  - ▷ GNN architecture can handle inputs of different size
  - ▷ Learn **optimal policies with supervision** that generalize over **much larger instances**

- In this work:
  - ▷ Learn **suboptimal policies without supervision**
  - ▷ Show how some expressive power limitations of architecture can be overcomed

Generalized Planning and First-Order STRIPS

- Generalized planning is about finding **general plans or strategies** that solve classes of planning problems

- **Generalized task** is collection of ground instances $P_i = \langle D, I_i \rangle$ that share a common **first-order STRIPS** domain D together with a **goal description**

- Instances $P = \langle D, I \rangle$ for general planning domain:
  - ▷ **Domain** D specified in terms of **action schemas** and **predicates**
  - ▷ **Instance** is $P = \langle D, I \rangle$ where I details **objects**, **init**, **goal**

Distinction between **general** domain D and **specific** instance $P = \langle D, I \rangle$ important for **reusing** action models, and also for **learning** them

Value Functions and Greedy Policies

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

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

- E.g., linear value functions have the form

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

- **Greedy policy** $\pi_V(s)$ chooses action a such that $V(s) = 1 + V(s_a)$:
  - ▷ If $V(s)=0$ for goals, and $V(s) = 1 + \min_a V(s_a)$ for non-goals, π_V is **optimal**
  - ▷ If second replaced by $V(s) \geq 1 + \min_a V(s_a)$, π_V "solves" **any state** s

Optimal vs. Suboptimal Policies

- In work at ICAPS-22, we trained neural nets to learn **optimal value functions** for generalized planning in supervised manner

- However, this isn't feasible in general:
  - ▷ In NP-hard tasks, no (general) optimal value function can be learned (unless P equals NP)
  - ▷ Even if planning task is in P, no neural net (circuit) may exist that produces (general) optimal value function

- Alternatively, some provable NP-hard tasks admit **greedy suboptimal policies** defined in terms of value functions over "simple" state features

In this work, we compute greedy suboptimal policies using GNNs

Graph Neural Networks (GNNs)

- GNN is computational model over undirected graphs:
  - ▷ Each vertex u **embbeded** into real vector $f(u) \in \mathbb{R}^k$
  - ▷ Computation performed for a **number of rounds,** where in a round:
    - ☐ Each vertex u **receives** embeddings $f(v)$ from its neighbours v
    - ☐ Then, u **aggregates** incomming messages and **combines** with $f(u)$ to produce new $f(u)$
  - ▷ **Final readout** for graph computed by aggregating embeddings $f(u)$ of all vertices
- Typically, aggregation and combination functions are the same for all vertices
- Model specified by **embedding dimension** k, the aggregation and combination functions, and final readout

GNN not tied to fixed-sized graphs; it can be applied to graphs of any size!

GNN-Based Architecture for Relational Structures

- Planning states s over STRIPS domain D correspond to **relational structures:**
  - ▷ Relational symbols given by D and hence shared by all states s
  - ▷ Denotations of predicates p given by ground atoms $p(\bar{o})$ true at s
- We adapt architecture of [Toenshoff *et al.*, 2021] for handling relational structures

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)

```
f_0(o) ∼ 0^{k/2} N(0,1)^{k/2} for each object o ∈ s;
for i ∈ {0, ..., L−1} do
    for each atom q := p(o_1, ..., o_m) true in s do
        // Msgs q → o for each o = o_j in q
        m_{q,o} := [MLP_p(f_i(o_1), ..., f_i(o_m))]_j;
    for each o in s do
        // Aggregate, update embeddings
        f_{i+1}(o) := MLP_U(f_i(o), agg({{m_{q,o} | o ∈ q}}));
// Final Readout
V := MLP_2(∑_{o∈s} MLP_1(f_L(o)))
```

Parameters θ: dimension k, rounds L, $\{\mathbf{MLP}_p : p \in D\}$, $\mathbf{MLP}_U$, $\mathbf{MLP}_1, \mathbf{MLP}_2$

Stochastic Gradient Descend and Loss Function

- Training aims at minimizing **loss over training set** by finding **best** θ with SGD
- Resulting function V_θ provides values for **any** state s in **any** instance $P = \langle D, I \rangle$
- In work at ICAPS-22, loss function is

$$Loss \;=\; \sum_{s \text{ in trainset}} Loss(s); \qquad Loss(s) = |V^*(s) - V_\theta(s)|$$

  This is **supervised learning** because targets $V^*(s)$

- If zero loss: $V_\theta(s) = V^*(s) = 1 + \min_a V^*(s_a)$ (Bellman equation)
- In this work, loss is essentially

$$Loss'(s) \;=\; \max\left\{\, 0,\ \big(1 + \min_a V_\theta(s_a)\big) - V_\theta(s) \,\right\}$$

- If zero loss: $V_\theta(s) \geq 1 + \min_a V_\theta(s_a)$ enough for greedy policy to be **solution**

Experimental Results 1/2

- Instance sizes in training, validation and testing by number of objects

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]

- Performance of two deterministic greedy policies: π_{V_θ} with and without **cycle avoidance**

	Deterministic policy π_V with cycle avoidance			**Deterministic policy π_V alone**		
Domain (#)	Coverage (%)	L	PQ = PL / OL (#)	Coverage (%)	L	PQ = PL / OL (#)
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)

Understanding and Overcoming Limitations

- Two sources for limitations of architecture:
  - ▷ **Number L of layers:** GNN cannot compute distances beyond $2L$
  - ▷ **Expressivity:** GNNs known to have **expressive power bounded by $\mathcal{C}_2$** [Barcelo *et al.*, 2020; Grohe, 2020]
  - ▷ Our model isn't equal to GNN model, yet we believe a similar bound applies
- To test our understanding, we perform the following:
  - ▷ **Spanner*:** add tr-closure of link/2 thus allowing computation of distances
  - ▷ **Logistics:** added some comp. of "roles" which are not expressible in $\mathcal{C}_2$
- Other domains not fully solved:
  - ▷ **Reward:** number of layers not enough
  - ▷ **Visitall:** implementing "cycle avoidance" achieves full coverage

Experimental Results 2/2

- After adding derived predicates and/or (even) reducing number L of layers:

	Deterministic policy π_V with cycle avoidance			**Deterministic policy π_V alone**		
Domain (#)	Coverage (%)	L	PQ = PL / OL (#)	Coverage (%)	L	PQ = PL / OL (#)
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)
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)

Conclusions and Discussion

- Use architecture of [Ståhlberg *et al.*, 2022] to learn general suboptimal policies for planning problems in a unsupervised fashion
- Understanding limitations of approach at "logical level"
- Aiming for suboptimal rather than optimal policies extends scope of approach as some tasks do not admit such general policies
- Notice that RL always aim at learning optimal policies
- Future work includes exploring the optimality vs. suboptimality tradeoff and relations with RL

References

[Barceló et al., 2020] 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*.

[Grohe, 2020] Grohe, M. (2020). The logic of graph neural networks. In *Proc. of the 35th ACM-IEEE Symp. on Logic in Computer Science*.

[Ståhlberg et al., 2022] Ståhlberg, S., Bonet, B., and Geffner, H. (2022). Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *Proc. ICAPS*.

[Toenshoff et al., 2021] Toenshoff, J., Ritzert, M., Wolf, H., and Grohe, M. (2021). Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3:98.