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

- General policies represent strategies for solving many planning instances
  - ▷ E.g., general policy for solving all Blocksworld problems
- Three main methods for learning such policies (no "synthesis" methods yet!)
  - ▷ **Combinatorial optimization using explicit pool of $\mathcal{C}_2$ features** obtained from domain predicates [B. *et al.*, 2019; Francès *et al.*, 2021]
    - □ Transparent, can be proved correct, trouble scaling up
  - ▷ **Deep learning (DL) using domain predicates but no explicit pool** [Toyer *et al.*, 2020; Garg *et al.*, 2020]
    - □ Opaque, complex, not 100% coverage, scalable
  - ▷ **DL exploiting relation between $\mathcal{C}_2$ logic and GNNs** [Barceló *et al.*, 2020; Grohe, 2020; Ståhlberg *et al.*, 2022]
    - □ More transparent, scalable, simple
    - □ GNN architecture adapted from Max-CSP$[\Gamma]$ [Toenshoff *et al.*, 2021]
    - □ Supervised and optimal: Learn value function $V(s)$ from optimal $V^*(s)$ targets
    - □ **Problem:** In many domains no optimal general policy exists

In this work . . .

- Learn general policies with **no supervision** and **not necessarily optimal**
- Same GNN architecture [Ståhlberg *et al.*, 2022] adapted from [Toenshoff *et al.*, 2021]
- Transparent, 100% coverage, scalable, simple
- $\mathcal{C}_2$ and GNN correspondence used to understand limitations and potential fixes, **at logical level**
- Insights on limitations of RL due to "conflict" between **optimality and generality**

Outline

- Generalized planning and representations in Lifted STRIPS
- Value functions and policies
- Optimality vs. generality
- Graph neural networks (GNNs)
- GNN for learning value functions
- Loss functions
- Experimental results
- Analysis
- Conclusions

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 **init and goal descr.**

- 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

Example: 2-Gripper Problem $P = \langle D, I \rangle$ in PDDL

```
(define (domain gripper)
   (:requirements :typing)
   (:types room ball gripper)
   (:constants left right - gripper)
   (:predicates (at-robot ?r - room) (at ?b - ball ?r - room)
                (free ?g - gripper)  (carry ?o - ball ?g - gripper))
   (:action move
       :parameters   (?from ?to - room)
       :precondition (at-robot ?from)
       :effect       (and (at-robot ?to) (not (at-robot ?from))))
   (:action pick
       :parameters   (?obj - ball ?room - room ?gripper - gripper)
       :precondition (and (at ?obj ?room) (at-robot ?room) (free ?gripper))
       :effect       (and (carry ?obj ?gripper) (not (at ?obj ?room)) (not (free ?gripper))))
   (:action drop
       :parameters   (?obj - ball ?room - room ?gripper - gripper)
       :precondition (and (carry ?obj ?gripper) (at-robot ?room))
       :effect       (and (at ?obj ?room) (free ?gripper) (not (carry ?obj ?gripper)))))

(define (problem gripper2)
    (:domain gripper)
    (:objects roomA roomB - room Ball1 Ball2 - ball)
    (:init (at-robot roomA) (free left) (free right) (at Ball1 roomA) (at Ball2 roomA))
    (:goal (and (at Ball1 roomB) (at Ball2 roomB))))
```

Value Functions and Greedy Policies

- **General value functions** for a class of problems defined over features ϕ_i that have well-defined values on **all reachable 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 $a = \mathrm{argmin}_{a \in A(s)} \, 1 + V(s_a)$:
  - If $V(s) = 0$ for goals, and $V(s) = 1 + \min_a V(s_a)$ for non-goals, π_V is **optimal**
  - If $V(s) = 0$ for goals, and $V(s) \geq 1 + \min_a V(s_a)$ for non-goals, π_V **solves any state** s

Optimal vs. Suboptimal Policies

- Some domains have **general optimal policies;** others like Logistics and Blocks don't
- Indeed,
  - ▷ For NP-hard tasks, no (general) optimal value function can be learned
  - ▷ Even if planning task is in P, no neural net (circuit) may exist that produces (general) optimal value functions
- Forcing learned value function V to be optimal [Ståhlberg *et al.*, 2022] not **good idea** for broad coverage

In this work, we compute greedy suboptimal policies using GNNs

Graph Neural Networks (GNNs)

- GNN is message-passing computational model over undirected graphs:
  - ▷ Each vertex u **embbeded** into real vector $f(u)$ of **dimension** k
  - ▷ Computation performed for **number of rounds** L, where in a round:
    - □ Each vertex u **receives** embeddings $f(v)$ from its neighbours $v \in N(u)$
    - □ These are **aggregated** and then **combined** with $f(u)$ to produce **new** $f(u)$
  - ▷ **Final readout** for graph computed on aggregation of embeddings $f(u)$ for all vertices
- Typically, aggregation and combination functions are the same for all vertices
- Model specified by (embedding) dimension k, number of rounds L, aggregation and combination functions, and final readout

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

GNN Architecture for Computing and Learning $V(s)$

- 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
- 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)

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

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

Training and Loss Functions

- For given/learned θ, V_θ provides values for **any state in any instance** $P = \langle D, I \rangle$
- Training using SGD minimizes **loss over training set** by finding **best** θ
- Loss functions:

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

$$Loss' = \sum_{s \text{ in trainset}} \max\{0, \left[1 + \min_a V_\theta(s_a)\right] - V_\theta(s)\}$$

- If $Loss = 0$, $V_\theta = V^*$ yields **optimal policies** on training set [Ståhlberg *et al.*, 2022]
- If $Loss' = 0$, $V_\theta(s) \geq 1 + \min_a V_\theta(s_a)$ yields **policies that solve** training set
- We care about **generalization performance** over new, test instances

Experiments: Setup

- Experiments aimed at testing generalization (coverage and quality) of greedy policy π_V for $V = V_\theta$
- Standard instances from International Planning Competition (IPC)
- Hyperparameters k and L set to 64 and 30: L affects how far messages propagate, k affects number of features in V_θ
- **Optimizer:** Adam with learning rate 0.0002
- **Hardware:** NVIDIA A100 GPUs for up to 12 hours
- During training, loss measured on "validation set"; best θ selected
- Quality measured with respect to **optimal plans**

Experimental Results: Minimizing $Loss'$

- 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

- Coverage:
  - ▷ **5 out 8 fully solved:** Blocks, Delivery, Gripper, Miconic, Visitall
  - ▷ **Exceptions:** Logistics (60%), Reward (73%), Spanner (73%)
- Why not full coverage then?
  - ▷ Logistics needs role composition, not in $\mathcal{C}_2$/GNN
  - ▷ Reward/Spanner need to compute distances larger than # of GNN iterations
- Fixes:
  - ▷ **Logistics:** add new atoms in states representing role compositions
  - ▷ **Spanner:** add transitive closure of binary predicates
- Results:
  - ▷ **In Logistics and Spanner coverage jumps to 100%**

Summary

- Adapt GNN architecture for Max-CSP(Γ) [Toenshoff *et al.*, 2021] for learning general value functions V_θ that yield policies π_{V_θ}
- Approach like [Ståhlberg *et al.*, 2022] but V learned **without supervision**
- Limitations of approach understood and "fixed at logical level"
- **Conflict between generality and optimality** as often there are no optimal general policies over many domains (e.g., Blocks)
- Loss function prefers generality rather than optimality
- Standard Bellman/RL losses try to achieve both but don't get either
- General and crisp limitation of RL methods for computing general policies
- **Open:** formal characterization of expressivity à la [Barceló *et al.*, 2020; Grohe, 2020]

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*.

[Bonet et al., 2019] Bonet, B., Fuentetaja, R., E-Martín, Y., and Borrajo, D. (2019). Guarantees for sound abstractions for generalized planning. In *Proc. IJCAI*, pages 1566–1573.

[Francès et al., 2021] Francès, G., Bonet, B., and Geffner, H. (2021). Learning general planning policies from small examples without supervision. In *Proc. AAAI*, pages 11801–11808.

[Garg et al., 2020] Garg, S., Bajpai, A., and Mausam (2020). Symbolic network: generalized neural policies for relational mdps. In *International Conference on Machine Learning*, pages 3397–3407.

[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.

[Toyer et al., 2018] Toyer, S., Trevizan, F., Thiébaux, S., and Xie, L. (2018). Action schema networks: Generalised policies with deep learning. In *AAAI*.