Learning General Representations to Act and Plan: Models and Policies

Hector Geffner and Blai Bonet

RWTH & USB

https://ml.rwth-aachen.de/~hector.geffner/ijcai2026-tutorial.html

Agenda

- What is generalized planning about?
- Abstractions (states, transitions, ...)
- Learning policies with symbolic methods
- Learning policies with neural methods
- Wrap up

Generalized Planning

- Get policy/strategy that solves **multiple instances** of same domain
- Class $\mathcal{Q}$ of **grounded** problems $P = \langle D, I \rangle$ over **common** domain D
- $\mathcal{Q}$ often infinite, not "representable"

- **Challenges:**
  - ▷ No single shared set of actions
  - ▷ Unbounded "branching factor" (bounded at each state, though)
  - ▷ **How to define and specify policies/strategies?**
  - ▷ **Policies must solve all instances in class $\mathcal{Q}$**

- **Example:** get policy that solves every instance of Blocks

- Is a (standard) planner a solution?

Examples Generalized Policies (Informal)

- **Gripper:** move all balls from room A to room B

```
1: if robot in room B then move to room A
2: while there are balls in room A do
3:     while there are balls in room A and free capacity do
4:         Pick a ball in room A
5:     Move to room B
6:     while there are balls being held do
7:         Drop ball in room B
8:     Move to room A
```

- **Blocks3ops:** build towers given in goal description

```
1: % Move all blocks to table
2: while there is a block on another block do
3:     Move some block to table
4: % Build towers from the bottom up
5: while there is still an incomplete tower do
6:     Let on(x, y) be a goal atom such that y is well-placed and x is on the table
7:     Stack x on top of y
```

Learning Isn't Optional

- Class $\mathcal{Q}$ often infinite, not "representable" (what does this mean?)
- Not every state description corresponds to a "real state"
  - ▷ Logistics: $[$`(in p1 airplane2)`, `(at p1 loc5)`$]$, etc
  - ▷ Blocks: $[$`(on A B)`, `(on B C)`, `(on C A)`$]$, etc
- Domain doesn't come with specification of intended states; FOL is not sufficient
- Avoid specification $\rightarrow$ synthesis not available $\rightarrow$ learning needed

Implications

- **Inductive:** find policy for **training instances;** expect generalization on entire $\mathcal{Q}$
- When $\mathcal{Q}$ is infinite (and discrete): extremal "out-of-distribution" generalization
- Sometimes can show policy is correct (ie. solves all $\mathcal{Q}$), but this is not guaranteed

Abstractions

- Need abstractions because:
  - Number and name of objects differ across instances
  - No single shared set of (grounded) actions
  - Unbounded "branching factor" (bounded at each instance, though)
- Policy language yields the abstractions:
  - States abstracted with features
  - Transition abstracted with rules
  - Trajectories (aka temporal abstractions; options/RL) abstracted also with rules
- **Result:** Policies agnostic to number/names of objects and transitions
- **Example (Blocks3ops):** "Increase number of well-placed blocks"

What's a Policy? What's a Solution? High-Level View

- A policy π is a **classifier** that separates transitions (edges) into **good or bad:**

  $\rightarrow$ When at a state, take any **good transition**

- A π-trajectory is $(s_0, s_1, s_2, ...)$ such that each transition (s_i, s_{i+1}) is good
- π solves P iff **each** maximal π-trajectory seeded at initial state ends in goal state
- Equivalently, π solves P (with initial state s_0) iff
  - ▷ **Safe:** no π-trajectory seeded at s_0 ends in dead end
  - ▷ **Closed:** if $(s_0, ..., s_n)$ is π-traj. and s_n isn't goal, there is good (s_n, s')
  - ▷ **Acyclic:** there is no π-trajectory $(s_0, ..., s_\ell, ..., s_k)$ with $s_\ell = s_k$ and $\ell < k$
- π solves $\mathcal{Q}$ if it solves each P in $\mathcal{Q}$
- **Learning:** for finite (and small) $\mathcal{Q}' \subseteq \mathcal{Q}$, find π that solves $\mathcal{Q}'$; expect solves $\mathcal{Q}$

State Abstractions

Feature-based abstractions, defined by small set of Boolean or numerical features:

- Boolean feature f maps states s into Boolean values $f(s)$
- numerical feature n maps states s into real values $n(s)$ (eg. non-negative integers)
- **Example (Block3ops):** number of well-placed blocks

Features obtained from **pool of concepts and roles;** constructed from **primitive** concepts and roles in planning domain D using a **grammar**

- Concept C at state s gets denotation C_s which is **subset of objects**
- Role R at state s gets denotation R_s which is **subset of object pairs**
- Item ϕ defines feature $f_\phi(s) = |\phi_s|$ (which is Boolean if always $\in \{0, 1\}$)

Grammar and Interpretations

- **Atoms:**
  - Native concepts C for unary predicates; roles R for binary predicates in domain
  - Native concepts/roles at goal: C/g and R/g
  - Universe $\top(s) = U(s)$ consists of all objects
  - $\texttt{nominal(C)}(s) = \{\texttt{C}\}$ for **PDDL constant** `C`
- **Concepts from concepts:**
  - $\texttt{union}(C,C')(s) = C_s \cup C'_s$
  - $\texttt{inter}(C,C')(s) = C_s \cap C'_s$
  - $\texttt{diff}(C,C')(s) = C_s \setminus C'_s$
  - $\texttt{compl}(C)(s) = U(s) \setminus C'_s$
- **Roles from roles:**
  - $\texttt{inverse}(R)(s) = \{(o,o') \mid (o',o) \in R_s\}$
- **Concepts from roles:**
  - $\texttt{exists}(R,C)(s) = \{o \mid \exists o'[(o,o') \in R_s \wedge o' \in C_s]\}$
  - $\texttt{forall}(R,C)(s) = \{o \mid \forall o'[(o,o') \in R_s \implies o' \in C_s]\}$
  - $\texttt{proj}(R,1)(s) = \{o \mid \exists o'[(o,o') \in R_s]\}$ (similarly, $\texttt{proj}(R,2)$)
- **Roles from concepts:**
  - $\texttt{restr}(R,1,C)(s) = \{(o,o') \in R_s \mid o \in C_s\}$ (similarly, $\texttt{restr}(R,2,C)$)
  - $\texttt{equals}(R,R')(s) = \{o \mid \forall o'[(o,o') \in R_s \iff (o,o') \in R'_s]\}$
- Size- (resp. depth-) complexity of item ϕ is size (resp. depth) of its parse tree

Examples of State Abstractions

- **Gripper** (where goal is to move balls from one room to another):
  - ▷ Vocabulary: `at-robot/1, at/2, carry/2, free/1`

robot at target room	$\rightsquigarrow$	`exists(at/g, at-robot)`
held balls	$\rightsquigarrow$	`exists(carry,`$\top$`)`
balls not in target room	$\rightsquigarrow$	`compl(equals(at/g, at))`
free grippers (capacity)	$\rightsquigarrow$	`free`

- **Blocks3opsTable** (where table is also an object):
  - ▷ Vocabulary: clear/1, on/2

clear blocks	$\rightsquigarrow$	`clear`
blocks not in final position	$\rightsquigarrow$	`compl(equals(on/g, on))`
blocks st all objects below are well-placed	$\rightsquigarrow$	`forall(tc(on), equals(on/g, on))`

Construction of Feature Pool

- Two items ϕ and ϕ' are:
  - Equivalent for state s: $\phi \sim_s \phi'$ iff $\phi_s = \phi'_s$
  - Equivalent for set $\mathcal{S}$ of states: $\phi \sim_{\mathcal{S}} \phi'$ iff $\phi \sim_s \phi'$ for all $s \in \mathcal{S}$
- Naive algorithm for constructing pool of features up to given complexity K:

```
Let 𝓘 := ∅
while true do
    Let 𝓘′ := ∅
    for S ⊆ 𝓘 with |S| ≤ 2 of total complexity < K do
        Create item φ from S (e.g. forall concept or inverse role)
        if φ isn't equivalent to any other item φ′ in 𝓘 ∪ 𝓘′ then
            𝓘′ := 𝓘′ ∪ {φ}
    if 𝓘′ = ∅ then break
    𝓘 := 𝓘 ∪ 𝓘′
```

- Recall item ϕ defines feature $|\phi|$ which is Boolean if always in $\{0, 1\}$

Extended Expressivity

- All items in grammar can be described in FOL with 2 variables (i.e., FO^2)
- Beyond FO^2:
  - Transitive closure: $\texttt{tc}(R)(s) = \{(o, o') \mid (o, o') \in R_s^+\}$
  - Composition: $\texttt{comp}(R, R')(s) = \{(o, o') \mid \exists o''[(o, o'') \in R_s \wedge (o'', o') \in R'_s\}$
  - ...
- Numerical distance features:
  - $\texttt{dist}(C, R, C')(s) \leq n$ iff $\exists(o_1, \ldots, o_n)$ st $o_1 \in C_s$, $o_n \in C'_s$, and $(o_i, o_{i+1}) \in R_s$
  - $\texttt{dist}(C, R, C')(s) = n$ iff $\texttt{dist}(C, R, C')(s) \leq n$ and $\texttt{dist}(C, R, C')(s) \not\leq n$
  - I.e., $\texttt{dist}(C, R, C')(s)$ measures the "shortest-path distance from C to C' mod R"
  - Distances are important in problems that involve movements

Rules: Syntax and Semantics

Rules have form $C \mapsto E$ where C and E are the conditions and effects, resp:

- **Conditions:** $\{p, \neg p\}$ for Booleans p; $\{n > 0, n = 0\}$ for numericals n
- **Effects:** $\{p, \neg p, p?\}$ for Booleans p; $\{n\uparrow, n\downarrow, n?\}$ for numericals n
- **Example:** $\{H, n > 0\} \mapsto \{\neg H, n\downarrow\}$ is rule over Boolean H and numerical n

Transition (s, s') is **compatible** with rule $C \mapsto E$ iff:

- Conditions evaluated at source s:
  - p (resp. $\neg p$) in $C \implies p(s) = 1$ (resp. $p(s) = 0$)
  - $n = 0$ (resp. $n > 0$) in $C \implies n(s) = 0$ (resp. $n(s) > 0$)
- Effects evaluated at (s, s'):
  - p (resp. $\neg p$) in $E \implies p(s') = 1$ (resp. $p(s') = 0$)
  - $n\uparrow$ (resp. $n\downarrow$) in $E \implies n(s') > n(s)$ (resp. $n(s') < n(s)$)
  - ϕ **not mentioned in** $E \implies \phi(s') = \phi(s)$

Rule-Based Policies

- Policy π is set of rules over the features Φ mentioned in π
- Transition (s, s') is good iff compatible with some rule in π
- Trajectory $(s_0, s_1, \ldots)$ is π-trajectory iff each transition (s_i, s_{i+1}) is good

Example: Policy for navigation problem where rewards in grid must be picked

$$\{r > 0, d > 0\} \mapsto \{d\!\downarrow\} \qquad \text{(approach closest reward by dec. distance } d\text{)}$$
$$\{r > 0, d = 0\} \mapsto \{r\!\downarrow, d?\} \qquad \text{(pick reward at cell; } d \text{ may increase)}$$

where $r \equiv |\texttt{at}|$ is number of rewards, and $d = \texttt{dist(at-robot,adj,exists(at,}\top\texttt{))}$ is distance to closest reward

Policy agnostic to #rewards and grid size. It's safe, closed and acyclic!

Principles for Learning: Projection and Separation

- **Projection** of transition (s, s') on features in Φ is rule $C \mapsto E$ where
  - C is Boolean valuation of the features in Φ on state s
  - E is qualitative change of the features in Φ across (s, s')
  - Fact. Transition (s, s') is compatible with its projection $C \mapsto E$

- Feature ϕ **separates** (s, s') and (t, t') iff ϕ changes differently across them

Example: Consider features $\Phi = \{n, target\}$ in Gripper:

- $n = |\texttt{exists(carry,}\top\texttt{)}|$ (# balls carried)
- $at = |\texttt{inter(at-robot,proj(at/g,2))}|$ (robot at target room)
- $target = |\texttt{exists(at,proj(at/g,2))}|$ (# balls at target room)

– Projection of `(drop B1 rB left)` is $\{n > 0, target = 0\} \mapsto \{n{\downarrow}, target{\uparrow}\}$

– at separates `(move rB rA)` and `(move rA rB)`

– n separates `(drop B1 rB left)` and `(move rB rA)`

SAT-Based Learning [Francès et al., 2021]

- **Input:** transitions $\mathcal{T}$ over states $\mathcal{S}$, feature pool $\mathcal{F}$, distance budget D
- **Output:** Propositional theory $\mathrm{Th}(\mathcal{T}, \mathcal{F}, D)$ over $\{\mathrm{select}(f), \mathrm{good}(s, s'), \mathrm{dist}(s, k)\}$

A1.	$\mathrm{Oneof}\ \{\mathrm{dist}(s, k) : d = 0, 1, \ldots, D\}$	[for non-dead end $s \in \mathcal{S}$]
A2.	$\bigvee_{(s,s') \in \mathcal{T}} \mathrm{good}(s, s')$	[for non-dead end $s \in \mathcal{S}$]
A3.	$\neg\mathrm{good}(s, s')$	[for dead end $s' \in \mathcal{S}$, $(s, s') \in \mathcal{T}$]
A4.	$\mathrm{good}(s, s') \wedge \mathrm{dist}(s, d) \wedge \mathrm{dist}(s', d') \implies d' < d$	[for $(s, s') \in \mathcal{T}$]
A5.	$\mathrm{good}(s, s') \wedge \neg\mathrm{good}(t, t') \implies \bigvee_{f : \Delta_f(s,s') \neq \Delta_f(t,t')} \mathrm{select}(f)$	$[(s, s'), (t, t') \in \mathcal{T}]$

where $\Delta_f(s, s') \in \{=, +, -\}$ is the change of f across (s, s')

Let $\Pi(\mathcal{T}, \mathcal{F}, D)$ be class of policies over $\mathcal{F}$ that solve non-deadend states in $\mathcal{S}$ in at most D steps. $\mathcal{T}$ is **closed** iff for each non deadend $s \in \mathcal{S}$, there is $(s, s') \in \mathcal{T}$ where s' is not deadend

Theorem. Let $\mathcal{T}$ be closed. $\Pi(\mathcal{T}, \mathcal{F}, D)$ is non-empty iff $\mathrm{Th}(\mathcal{T}, \mathcal{F}, D)$ is SAT.

Furthermore, if ν satisfies $\mathrm{Th}(\mathcal{T}, \mathcal{F}, D)$, the **projection** of good_ν on $\mathcal{F}_\nu$ belongs to $\Pi(\mathcal{T}, \mathcal{F}, D)$ where $\mathrm{good}_\nu = \{(s, s') \in \mathcal{T} : \nu \models \mathrm{good}(s, s')\}$ and $\mathcal{F}_\nu = \{f \in \mathcal{F} : \nu \models \mathrm{select}(f)\}$

Learning Algorithm

- Given (small) set $\{P_1, \ldots, P_k\}$ of (small) training instances (e.g. $k=1$ or $k=2$)
- Get pool $\mathcal{F}$ of feature of complexity bounded by $K_{\mathcal{F}}$ (parameter)
- Obtain reachable transitions $\mathcal{T}$ and states $\mathcal{S}$; construct theory $\mathrm{Th}(\mathcal{T}, \mathcal{S}, D)$
- Call Max-SAT solver on $\mathrm{Th}(\mathcal{T}, \mathcal{S}, D)$ to prefer "simple over complex" policies
- If successful, project transitions in $\mathcal{T}$ over solution ν to get policy, and evaluate. Else, increase pool, D, or fail.
- **Optimizations:**
  - Constraint values of d for $\mathrm{dist}(s, d)$; eg. $d \in \{d \mid V^*(s) \leq d \leq \delta V^*(s)\}$ where $\delta = 2$
  - Pool $\mathcal{F}$ defines equivalence among states; formulate theory on representatives (preprocessing)
  - Lazy generation of separation clauses (A5) (since their number is quadratic in $|\mathcal{T}|$)
- **Bottlenecks:** size of pool, number of states and transitions

Results

- Open-WBO Weighted Max-SAT solver [Martins, Manquinho, and Lynce 2014]
- Pool complexity $K_{\mathcal{F}} = 8$

Domain	$\lvert\mathcal{Q}\rvert$	dim	$\mathcal{T}$	$\mathcal{T}/\sim$	$d_{\max}$	$\lvert\mathcal{F}\rvert$	vars	clauses	time	SAT	compl.	$\lvert\Phi\rvert$	K^*	$\lvert\pi\rvert$
`Blocks4ops-clear`	1	5	1,161	55	7	532	7.9K	247.7K	6	< 1	8	3	4	3
`Blocks4ops-on`	1	5	1,852	329	10	1,412	17.3K	376.6K	33	22	13	5	5	7
`Gripper`	1	4	1,140	61	12	835	6.5K	102.6K	2	< 1	9	3	4	4
`Reward`	1	5×5	432	361	15	514	5.5K	214.9K	2	< 1	7	2	6	2
`Delivery`	2	4×4	42,473	5,442	56	1,373	753.4K	38.2M	3,071	2,902	30	4	14	6
`Visitall`	1	3×3	2,396	310	8	188	13.9K	244.5K	3	< 1	7	2	5	1
`Spanner`	3	$(6, 10)$	10,777	96	19	764	85.0K	2.2M	32	< 1	9	3	6	2
`Miconic`	2	$(4, 7)$	4,706	4,636	14	1,073	23.8K	23.6M	41	61	11	4	5	5
`Blocks3ops`	2	5	4,275	4,275	8	1,896	22.1K	9.3M	80	40	11	3	6	1

#training instances, dimension, #transitions, #equiv. classes modulo features, pool size, size SAT theory in vars and clasuses, total time, time in SAT, complexity and size of selected features, effective pool complexity, and number of rules

Takeaway: these domains solved with few features of low complexity and few rules

Example Learned Policy: Blocks3opsTable

- Vocabulary: `table/0, clear/1, on/2`
- Features:
  - $c = |$`clear`$|$ [#objs clear]
  - $t' = |$`compl(equals(on/g,on))`$|$ [#objs **not** in target]
  - $bwp = |$`forall(tc(on),equals(on/g,on))`$|$ [#objs st all below are well-placed]
- Learned policy:

$$\{c>0, t'>0, bwp>0\} \mapsto \{c\uparrow, bwp\uparrow, t'?\} \quad \text{(if possible, increase } bwp\text{)}$$
$$\{c>0, t'>0, bwp>0\} \mapsto \{c\uparrow\} \quad \text{(move block to table)}$$
$$\{c>0, t'>0, bwp>0\} \mapsto \{c\uparrow, t'\downarrow\} \quad \text{(move away bad-placed block)}$$
$$\{c>0, t'>0, bwp>0\} \mapsto \{c\downarrow, t'\downarrow\} \quad \text{(move away bad-placed block)}$$

- Policy is safe, closed, and acyclic [Francès et al., 2021]

Structural Termination

Consider the policy:

- $\{H\} \mapsto \{\neg H\}$ (H flips value from **true** to **false**)
- $\{\neg H, n > 0\} \mapsto \{H, n\downarrow\}$ (H flips value from **false** to **true**, and n **decrements**)

The policy is acyclic by design:

- ▷ each rule flips the value of the Boolean H
- ▷ each flip from $\neg H$ to H comes with a decrement of n which no rule increments

Hence, for non-negative integer-valued features, there is a **finite** number of flips

Policy π is **structurally terminating** if it cannot generate a cycle independently of the denotation of its features. Termination is implied by how features change

Monotone Features

Let R be a set of rules over the features in Φ:

- Numerical n is **monotone in** R if no rule increases it, or no rule decreases it
- Boolean p is **monotone in** R if no rule makes it true, or no rule makes it false

For numerical (or Boolean) g, consider the following subsets of rules:

- $\mathrm{Rules}(R, g, =) = \{r \in R : \{g\uparrow, g\downarrow\} \cap \mathrm{eff}(r) = \emptyset\}$
- $\mathrm{Rules}(R, g, 0) = \{r \in \mathrm{Rules}(R, g, =) \mid \text{`}g > 0\text{'} \notin \mathrm{cond}(r)\}$
- $\mathrm{Rules}(R, g, 1) = \{r \in \mathrm{Rules}(R, g, =) \mid \text{`}g = 0\text{'} \notin \mathrm{cond}(r)\}$

f is **monotone given** g iff it is monotone in $\mathrm{Rules}(R, g, 0)$ and $\mathrm{Rules}(R, g, 1)$

1-Stratified Policies

Policy π is **1-stratified** iff

- Every rule entails the change of some feature
- There is ranking κ of features such that
  - if $\kappa(f) = 0$, feature f is monotone in π
  - if $\kappa(f) > 0$, there is g such that $\kappa(g) < \kappa(f)$ and f is monotone given g

Theorem. If π is 1-stratified, it's acyclic. Testing 1-stratification takes polytime

Example: Policy for Blocks3opsTable is 1-stratified: $bwp \prec t' \prec c$

$$\{c > 0, t' > 0, bwp > 0\} \mapsto \{c\uparrow, bwp\uparrow, t'?\} \quad \text{(if possible, increase } bwp\text{)}$$
$$\{c > 0, t' > 0, bwp > 0\} \mapsto \{c\uparrow\} \quad \text{(move block to table)}$$
$$\{c > 0, t' > 0, bwp > 0\} \mapsto \{c\uparrow, t'\downarrow\} \quad \text{(move away bad-placed block)}$$
$$\{c > 0, t' > 0, bwp > 0\} \mapsto \{c\downarrow, t'\downarrow\} \quad \text{(move away bad-placed block)}$$

Learning Terminating Policies From Examples

- SAT-based learning solves two (coupled) combinatorial tasks:
  - ▷ Choosing good transitions for alive states
  - ▷ Choosing features that separate good from bad transitions

- **Waive first task;** an example from tutor provides **good transitions**
- Policy rules obtained by **projecting** good transitions into selected features
- **New task:** choose features $\{f_1, \ldots, f_k\}$ that make up a "terminating set"
- **No SAT/ASP;** want something fast and scalable; switch to **hitting sets!**

Hitting Set Problem

- Classical NP-Complete problem (one of Karp's problems):
  - **Input:** collection $S_1, S_2, \ldots, S_n$ of subsets over universe $U = \cup_{1 \leq i \leq n} S_i$
  - **Input:** non-negative integer $k \geq 0$
  - **Question:** is there $H \subseteq U$ such that $|H| \leq k$, and $S_i \cap H \neq \emptyset$ for $1 \leq i \leq n$?
- Dual of Set Cover, non-approximable, FPT for different parameters
- Weighted version: $w : U \to \mathbb{N}$, and ask for H of weight $\sum_{e \in H} w(e) \leq k$
- **Approximation greedy algorithm** chooses element that hits maximum number of not-yet-hit subsets normalized by weight:

1: $H := \emptyset$
2: **while** there are subsets not hit by H **do**
3: Choose e with max score$(e) \doteq |\{i \mid e \in S_i,\ S_i \cap H = \emptyset\}| \,/\, w(e)$
4: $H := H \cup \{e\}$

[Approximation ratio of $\ln n + o(1)$ for n subsets]

Base for Learning Terminating Policies using Hitting Sets

- **Inputs:** Examples as set $\mathcal{T}$ of transitions, and (possibly large) pool $\mathcal{F}$ of features
- **Output:** Subset $\Phi \subseteq \mathcal{F}$ of features on which to project the transitions in $\mathcal{T}$
- Features in Φ must satisfy set of **requirements** expressed **extensionally** as subsets:
  - For each edge $e \in \mathcal{T}$, some $f \in \Phi$ must change: $R_e = \{f \in \mathcal{F} \mid f \text{ changes across } e\}$
  - (whatever else suits you . . . more about this shortly)

Base for Learning Terminating Policies using Hitting Sets

- **Inputs:** Examples as set $\mathcal{T}$ of transitions, and (possibly large) pool $\mathcal{F}$ of features
- **Output:** Subset $\Phi \subseteq \mathcal{F}$ of features on which to project the transitions in $\mathcal{T}$
- Features in Φ must satisfy set of **requirements** expressed **extensionally** as subsets:
  - For each edge $e \in \mathcal{T}$, some $f \in \Phi$ must change: $R_e = \{f \in \mathcal{F} \mid f \text{ changes across } e\}$
  - (whatever else suits you . . . more about this shortly)
- Hitting set formulation:

Input: $\mathcal{T}$, pool $\mathcal{F}$ (each with comp. $w(f)$), and set $\mathcal{R}$ of requirements (each is $\mathcal{F}$-subset)

1: $\Phi := \emptyset$
2: **while** there is some requirement not hit by Φ **do**
3: Let $\mathcal{F}_{\text{enabled}} \subseteq \mathcal{F}$ be the set of "features enabled by Φ"
4: Choose f in $\mathcal{F}_{\text{enabled}}$ with max score$(f) \doteq |\{R \in \mathcal{R} \mid f \in R,\ R \cap \Phi = \emptyset\}| \,/\, w(f)$
5: $\Phi := \Phi \cup \{f\}$

where $\mathcal{F}_{\text{enabled}} = \{f \in \mathcal{F} \mid f \text{ is monotone, or monotone given some } g \text{ in } \Phi\}$

- Resulting policy π is **projection** of $\mathcal{T}$ over Φ
- **Benefits:** can handle large pools and examples

Evaluate the Policy

- What can go wrong?
- **Policy generates a cycle** **Impossible. By design, the policy is terminating!**
- Policy reaches state s where no transition (s, t) is compatible with a rule:
  - ▷ It can happen. **Policy isn't necessarily closed**
  - ▷ Ask the tutor (planner) to provide new transition $e = (s, t)$
  - ▷ Augment transitions and requirements: $\mathcal{T} := \mathcal{T} \cup \{e\}$ and $\mathcal{R} := \mathcal{R} \cup \{R_e\}$
  - ▷ Number of requirements grows by 1
- Policy generates trajectory $(s_0, s_1, \ldots, s_n)$ where s_n is dead end state:
  - ▷ It can also happen. **Policy isn't necessarily safe**
  - ▷ **Find guilty transition** $b = (s_i, s_{i+1})$ for the purpose of marking it as **bad transition**
  - ▷ For each good transition $e \in \mathcal{T}$, **add requirement** $R_{e,b} = \{f \in \mathcal{F} \mid \Delta_f(e) \neq \Delta_f(b)\}$
  - ▷ Number of requirements grows by $|\mathcal{T}|$
  - ▷ **Guilty transition:** (s_i, s_{i+1}) is "first transition" such that all previous are good

Results (Successes)

					Last outer iteration							Time in seconds			
Domain	$\|\mathcal{Q}\|$	$\|\mathcal{F}\|$	Outer	Inner	$\|\mathcal{Q}'\|$	Inner*	$\|\mathcal{T}\|$	$\|\mathcal{B}\|$	$\|\|H\|\|$	$\|H\|$	$\|\pi\|$	Prep.	Alg.	Verif.	Total
3puzzle	19	1,000	1	1	1	1	6	0	12	3	5	0.0	0.0	0.1	2.4
Blocks4ops-clear	53	13,579	1	1	1	1	7	0	14	2	2	0.6	0.0	0.1	10.4
Delivery-1pkg	169	5,765	1	1	1	1	9	0	18	4	4	0.2	0.0	3.2	24.6
Gripper	5	6,154	1	1	1	1	19	0	38	3	4	0.4	0.2	3.9	8.2
Reward	207	249,122	1	1	1	1	17	0	34	2	2	26.1	33.7	6.9	152.5
Visitall	460	146,085	1	1	1	1	20	0	40	2	3	15.6	1.4	991.5	1,110.1
Childsnack	10	2,900	1	2	1	2	15	0	30	6	8	0.5	0.0	32.1	35.6
Spanner-1nut	90	8,001	1	2	1	2	6	1	19	3	3	0.5	0.0	0.5	11.7
Logistics-1truck	67	262,509	1	17	1	17	32	0	64	5	8	905.5	803.5	2.7	1,926.6
Barman-1cocktail-1shot	90	69,040	1	21	1	21	32	2	133	11	22	243.1	539.9	216.1	1,049.9
Blocks4ops-on	94	183,322	2	2	1	1	10	0	20	4	7	24.2	1.9	1.1	108.2
Spanner	270	13,679	2	3	1	2	10	1	31	3	3	1.9	0.3	176.0	210.4
Delivery	397	13,904	4	21	1	3	23	0	46	4	5	42.1	6.6	23.0	135.3
Ferry	180	8,547	5	13	1	6	17	0	34	4	5	8.6	3.9	3.6	39.3
Miconic	360	107,785	9	30	1	6	20	0	40	4	5	253.8	73.7	50.8	502.8
8puzzle-1tile-fixed	18	11,230	7	38	4	3	40	0	80	8	14	44.7	25.5	1.8	81.5
8puzzle-1tile	16	12,417	8	52	3	6	40	0	80	8	18	69.1	35.6	2.2	117.8
Blocks4ops	5	100,897	10	66	3	5	81	0	162	7	24	5,349.4	7,159.4	19.6	12,863.9
Sokoban-1stone-7x7	8	115,214	12	309	5	15	94	4	671	15	50	14,248.4	12,712.3	896.1	28,196.5
Logistics-1pkg	24	225,518	15	138	4	8	56	0	112	7	16	6,936.5	5,913.8	416.7	16,136.6
Zenotravel-1plane	73	24,959	28	266	7	3	122	0	244	7	18	2,039.2	277.3	2,905.9	5,406.4

#instances, #features, #outer/inner iterations, #instances seen, #good/bad transitions, size of hs. problem, size of hs, policy size, and preprocessing, algorithm, and verification times

Takeaway: pools with 3+ orders-of-magnitude more features; much larger training instances

Results (Failures)

					Last outer iteration						Time in seconds			
Domain	$\|\mathcal{Q}\|$	$\|\mathcal{F}\|$	Outer	Inner	$\|\mathcal{Q}'\|$	Inner*	$\|\mathcal{T}\|$	$\|\mathcal{B}\|$	$\|\|H\|\|$	Reason	Prep.	Alg.	Verif.	Total
Rovers	608	190,064	1	1	1	1	25	0	50	Edge	17.2	0.5	0.0	225.7
Tidybot-opt11-strips	8	59,402	1	1	1	1	40	0	80	Edge	7.7	0.2	0.0	119.0
Tpp	11	14,128	1	1	1	1	201	0	402	Edge	9.3	0.2	0.0	237.1
8puzzle-2tiles	16	4,458	1	2	1	2	20	0	40	Edge	0.8	0.1	0.1	6.3
Hiking	180	16,723	1	2	1	2	8	0	16	Edge	1.4	0.0	4.0	190.0
Depot	18	255,079	1	17	1	17	119	0	238	Edge	4,771.5	379.5	415.2	15,511.7
Freecell	65	146,428	1	35	1	35	133	19	2,964	Timeout	—	—	—	—
Barman-1cocktail	270	69,040	1	94	1	94	105	6	868	Edge	3,056.5	5,474.4	36.4	8,769.3
Tetris-opt14-strips	16	37,496	1	142	1	142	180	1	550	Edge	5,109.8	3,458.8	7,157.7	16,500.1
Satellite	950	171,475	2	209	1	157	168	0	334	Timeout	—	—	—	—
Driverlog	381	172,818	6	152	1	16	31	0	62	Edge	5,768.1	1,822.2	52.3	8,088.9
Zenotravel-1person	80	42,271	12	385	1	77	80	3	507	Edge	2,262.9	350.4	1,359.2	4,701.4
Logistics	282	51,418	11	72	4	5	99	0	198	Edge	987.5	169.3	15.4	1,310.9
Blocks3ops	392	236,954	17	102	7	8	106	0	206	Timeout	—	—	—	—

#instances, #features, #outer/inner iterations, #instances seen, #good/bad transitions, size of hs. problem, **failure reason**, and preprocessing, algorithm, and verification times

Takeaway: understand failures, mainly due to lack of expressivity in feature pool

Beyond STRIPS Planning

- Policy language only knows about features (Boolean and numerical), and rules
- **It is agnostic to underlying state model**
- STRIPS is not bounding
- Same applies to previous learning algorithms based on projection and separation
- Feature pool does assumes underlying first-order state representation
- Non-deterministic settings also accomodated [Hofmann & Geffner, 2024]

Related Work

- Srivastava et al. [2009, 2011]: Origin of generalized planning; plans with loops; qualitative numeric planning; SIEVE algorithm
- Srivastava [2023]: Hierarchical decompositions and termination analysis for generalized plans
- De Giacomo et al. [various]: GP via FOND/LTL-style formulations
- Segovia-Aguas & Jonsson [2021, 2024]: GP as heuristic search; learning of programs formulated as STRIPS planning
- Illanes & McIlraith [2019]: Generalized planning via numeric abstractions
- Chen et al. [2026]: Generalized planning via regression

GenSieve: General Termination Test [Srivastava, 2023]

- Our policies borrowed from QNP model where increments/decrements are by uncertain non-fixed amounts. Termination of given policy π decided by SIEVE [Srivastava et al., 2011] in time exponential in $|\Phi|$
- SIEVE operates on the **policy graph** by removing edges until it becomes acyclic (iff π is terminating) or no further edge can be removed (iff π is non-terminating)
- Srivastava [2023] addresses more expressive model where increments/decrements are by fixed known amounts, and testing termination becomes **undecidable**
- Sound and incomplete test GENSIEVE subsumes SIEVE. Whereas SIEVE reasons about edges, GENSIEVE reasons about paths
- **Takeaway:** general-purpose verifier usable inside a synthesis/learning loop

Generalized Planning via Goal Regression [Chen et al., 2026]

- **Idea:** solve each goal atom separately (optimally, greedily, in some order) on handful of training problems, then **regress** each plan into a reusable rule
- Learning loop:
  - ▷ Order goal atoms; optimally solve each atom in turn, progressing the state
  - ▷ **Goal-regress** each optimal plan step-by-step; yields table with rows:

$$\big(\,\text{partial state}\,,\ \text{goal}\,,\ \text{action suffix}\,\big)$$

  - ▷ **Lift** each row into rule $C \mapsto$ Act. seq. with precedences
- Set of rules with precedences form a $\textsc{Moose}$ program. Completeness guaranteed on a class of STRIPS problems:
  - ▷ Problems where goal atoms are independent, solvable one at a time, in any order, without undoing progress on other goal atoms, and each individual goal is reachable within a bounded number of steps

Learning Policies with Neural Methods

Graph Neural Networks (GNNs)

- Neural architecture capable of processing graphs $G = (V, E)$ of **any order (size)**
- Primary function is to compute **node (vertex) embeddings,** each a vector in $\mathbb{R}^d$
- Architecture characterized by width (embedding dimension), depth (num. layers), (perm. invariant) aggregation and combination functions:

$$\begin{aligned} \boldsymbol{f}_0(u) &= \boldsymbol{e}_\alpha \\ \boldsymbol{f}_{k+1}(u) &= \mathrm{comb}_k\big(\boldsymbol{f}_k(u), \mathrm{aggr}_k\big(\{\!\!\{\boldsymbol{f}_k(v) \mid (u,v) \in E\}\!\!\}\big)\big) \end{aligned}$$

  where $\boldsymbol{e}_\alpha$ is initial embedding for vertices of "atomic type" α

- At the end, final embeddings aggregated and passed through "readout function":

$$\boldsymbol{f}(G) = \mathrm{readout}\big(\{\!\!\{\boldsymbol{f}_L(u) \mid u \in V\}\!\!\}\big)$$

- Functions have **learnable parameters** (often initial embeddings also learnable)

Concrete GNN Architectures

- **Important:** aggregation must be **permutation invariant;** eg. sum, max, etc
- Frequent choice:
  - ▷ Sum/max aggregation of embeddings
  - ▷ Linear combination; non-linear activation (eg ReLU); residual or non-residual
  - ▷ It can be homogeneous (all layers equal) or not; readout is MLP
  - ▷ Related to GIN model [Xu et al., 2019]
- For layers $k = 0, 1, \ldots, L-1$:

$$\begin{aligned} \boldsymbol{f}_0(u) &= \boldsymbol{e}_t \\ \boldsymbol{f}_{k+1}(u) &= \mathrm{ReLU}\left(\boldsymbol{A_k}\, \boldsymbol{f}_k(u) + \boldsymbol{B_k} \sum_{(u,v)\in E} \boldsymbol{f}_k(v) + \boldsymbol{C}_k \right) \\ \mathrm{readout}(S) &= \mathrm{MLP}\left(\textstyle\sum_{\boldsymbol{e}\in S} \boldsymbol{e}\right) \end{aligned}$$

where $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ are $\mathbb{R}^{d\times d}$ matrices, and $\boldsymbol{e}_t$ and $\boldsymbol{C}_k$ are $\mathbb{R}^d$ vectors

Implementation and Training of GNNs

- Easily implemented in PyTorch and trained with stoch. gradient descend (SGD)
- Data is split into training, validation, and test set:
  - **Training:** data directly used to do actual training (loss and gradient calculation)
  - **Validation:** after X epochs (epoch is when all data have been seen), net is evaluated on validation set (performance and parameters stored)
  - **Select** model (parameters) with best performance on validation set (across epochs)
  - **Evaluate** selected model on test set (withheld from entire training pipeline)
- Training can be supervised, self-supervised, or RL:
  - **Supervised:** training data consists of pairs (G, L) for graph G and label L
  - **Self-supervised:** focus on extracting patterns or (latent) representations without the need of labels (dimensionality reduction VAEs, MAEs, constrastive learning, etc)
  - **Reinforcement Learning:** different learning paradigm; GNN defines "policy" that interacts with environment which yields **reward signals**
- **Hyperparameter tuning crucial and difficult:** batch size, learning rate, ...

1-WL, Counting Logics, and GNNs

- Clean connection between graph coloring, logic and GNNs
- Weisfeiler-Lehman coloring algorithm (1-WL):

$$
\begin{aligned}
f_0(u) &= \text{“initial color depending on atomic type of vertex”} \\
f_{k+1}(u) &= \textsc{Relabel}\big(f_k(u), \{\!\{ f_k(v) \mid (u,v) \in E \}\!\}\big)
\end{aligned}
$$

  Iterate until no further refinement possible (max $|V|$ iterations)

- Final coloring f_∞: $u \sim v$ iff $f_\infty(u) = f_\infty(v)$. Vertex partition is $\{[u] \mid u \in V\}$
- If $G \simeq G'$, they generate equivalent (vertex) partitions; converse is false

Counting Logics and 1-WL

- FO^2 is FOL with 2 variables (restrictive fragment)
- Increase expressivity by allowing **counting quantifiers** $\exists^{\geq n} y\, \phi(x, y)$ meaning there are at least n **distinct objects** y such that $\phi(x, y)$ holds
- Resulting logic is called C^2 (in some variants, the integer n can be quantified and appear in "arithmetic" operations)
- C^2 in graphs:
  - $\phi(x) = \exists^{\geq 5} y\big(E(x, y) \wedge \mathrm{red}(y)\big)$ (x has at least 5 red neighbors)
  - $\psi_0(x) = \mathrm{red}(x)$
  - $\psi_{d+1}(x) = \psi_d(x) \vee \exists y\big(E(x, y) \wedge \psi_d(y)\big)$ (x is at dist. $\leq d + 1$ from red)
  - $\varphi(x) = \exists n, m\, \big((n < m) \wedge \exists^{=n} y\big(E(x, y) \wedge \mathrm{red}(y)\big) \wedge \exists^{=m} y\big(E(x, y) \wedge \mathrm{blue}(y)\big)\big)$
- **Theorem.** For any two graphs G and G', and vertices u and v in G:
  - $G \simeq_{\text{1-WL}} G'$ iff there is no C^2 formula that distinguish G and G'
  - $f_k(u) = f_k(v)$ iff for every C^2 formula $\phi(x)$ of **rank** k, $G \models \phi(u)$ iff $G \models \phi(v)$

GNNs and 1-WL

- GNNs cannot distinguish graphs/vertices beyond 1-WL:
  - ▷ If $G \simeq_{\text{1-WL}} G'$, then $\boldsymbol{f}(G) = \boldsymbol{f}(G')$ for every GNN
  - ▷ If $f_k(u) = f_k(v)$, then $\boldsymbol{f}_k(u) = \boldsymbol{f}_k(v)$ for every GNN
- **Guarded fragment** GC^2 made of guarded formulas:

$$\phi \leftarrow U(x) \mid E(x,y) \mid \neg\phi \mid \phi \vee \phi \mid \exists^{\geq n} y\big(E(x,y) \wedge \phi(y)\big)$$

- If vertices u and v cannot be distinguished by formulas in GC^2 of **rank** k, then $\boldsymbol{f}_k(u) = \boldsymbol{f}_k(v)$ for every GNN (without readouts)

Relational GNN for Planning

- GNNs are designed for graphs; planning states are not graphs
- STRIPS domain D defines vocabulary σ with constants and predicates in D
- Planning state s is **relational structure:** denotations given by grounded atoms
- R-GNN$[D]$ for planning [Stahlberg et al., 2022–today; Toenshoff et al., 2021]:

Input: set S of ground atoms, objects O
Output: embeddings $\boldsymbol{f}_L(o)$ for each object $o \in O$

Initialize $\boldsymbol{f}_0(o) = 0^d$ for each object $o \in O$
for layer index $i = 1, 2, \ldots, L-1$ **do**
  for each atom $q = R(o_1, o_2, \ldots, o_k)$ in S **do**
    $\boldsymbol{m}_q = \mathrm{MLP}_R\big(\boldsymbol{f}_i(o_1), \boldsymbol{f}_i(o_2), \ldots, \boldsymbol{f}_i(o_k)\big)$
    Send message $\boldsymbol{m}_{q,o_j} := [\boldsymbol{m}_q]_j$ to object o_j
  for each object o in O **do**
    $\boldsymbol{f}_{i+1}(o) := \boldsymbol{f}_i(o) + \mathrm{MLP}_U\big(\boldsymbol{f}_i(o), \mathrm{aggr}(\{\!\{\boldsymbol{m}_{q,o} \mid q \in S, o \in q\}\!\})\big)$

- MLP_R per predicate R; MLP_U for combining; residual connections
- Aggregation function either sum, smooth-max, or (component-wise) combination

Training and Loss Functions [Stahlberg et al. 2022]

- Readout generates scalar value (MLP over aggregation of final embeddings)
- Training minimizes **loss over training set** by finding **best parameter θ**
- Loss functions:

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

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

- If $Loss = 0$, $V_{\boldsymbol{\theta}} = V^*$ yields **optimal policies** on training set [Ståhlberg *et al.*, 2022]
- If $Loss' = 0$, $V_{\boldsymbol{\theta}}(s) \geq 1 + \min_a V_{\boldsymbol{\theta}}(s_a)$ yields **policies that solve** training set

Experiments: Setup

- Experiments aimed at testing generalization (coverage and quality)
- Standard instances from International Planning Competition (IPC)
- Width and depth set at $d = 64$ and $L = 30$: L affects how far messages propagate, d affects number of features in $V_{\boldsymbol{\theta}}$
- Adam optimizer with learning rate 0.0002
- **Hardware:** NVIDIA A100 GPUs for up to 12 hours
- During training, loss measured on "validation set"; best $\boldsymbol{\theta}$ 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 C_2/GNN
  - Reward/Spanner need to compute distances larger than # of layers
- 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%**

Deep Reinforcement Learning (DRL)

- Different ways to do it ...
  - ▷ Approximate V- or Q-functions with neural net, do VI or PI
  - ▷ Parameteric policy combined with **policy gradient methods**
- "Deep" means functions/policy represented with deep neural net

Policy Gradient Methods

- Learn parameterized policy to select actions
- Parameter learned with gradients of some performance measure $J(\boldsymbol{\theta})$. Methods seek to maximize performance, so they do gradient ascent
- To ensure exploration, policy should never become deterministic. If action space is "small", it is common **softmax actor** from preferences or logits $h(s, a, \boldsymbol{\theta})$:

$$\pi(a|s, \boldsymbol{\theta}) \;=\; \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

- Methods that learn policy **and** value functions called actor–critic: "actor" refers to policy, and "critic" to value function. Critic discarded after learning
- **Examples:** REINFORCE, Actor–Critic, PPO, ...

REINFORCE

- Classic (perhaps first) well-understood policy gradient method [Williams, 1992]
- Correctness established with **policy gradient theorem** [Sutton & Barto, 2000]
- Updates occur at end of episodes (which may be too long)

Input: differentiable (parametric) policy $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$

1: Initialize policy parameters $\boldsymbol{\theta}$ (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
2: **loop forever** (for each episode):
3: Generate episode $S_0, A_0, R_0, \ldots, S_{T-1}, A_{T-1}, R_T$ following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
4: **for** each step $t = 0, 1, 2, \ldots, T-1$ **do**
5: $G := \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
6: $\boldsymbol{\theta} := \boldsymbol{\theta} + \alpha \, \gamma^k \, G \, \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Actor–Critic

- Critic $\hat{v}(s, \boldsymbol{w})$ estimates V-value of state s
- Unlike REINFORCE, parameters updated at each step

Input: diff. policy (actor) $\pi(a|s, \boldsymbol{\theta})$ and state-value function (critic) $\hat{v}(s, \boldsymbol{w})$
Algorithm parameters: step sizes $\alpha > 0$ and $\beta > 0$

Initialize policy parameters $\boldsymbol{\theta}$ and state-value parameters $\boldsymbol{w}$
loop forever (for each episode):
    Initialize S (first state of episode)
    $I := 1$
    while S is not terminal **do**
        $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
        Take action A, observe resulting state S' and reward R
        $\delta := R + \gamma\, \hat{v}(S', \boldsymbol{w}) - \hat{v}(S, \boldsymbol{w})$ (if S' is terminal, $\hat{v}(S', \boldsymbol{w}) \doteq 0$)
        $\boldsymbol{w} := \boldsymbol{w} + \beta\, \delta\, \nabla \hat{v}(S, \boldsymbol{w})$
        $\boldsymbol{\theta} := \boldsymbol{\theta} + \alpha\, I\, \delta\, \nabla \ln \pi(A|S, \boldsymbol{\theta})$
        $I := \gamma I$
        $S := S'$

Actor–Critic Algorithms for Generalized Planning

- Policies map states s into probabilities $\pi(s'|s)$ for successors s'
- Training set is $\{M_i\}$ of deterministic MDPs. Sample M_i with some prior p_i, then sample state s in M_i (uniform)
- Action costs set to 1
- When sampled successor s' is goal, parameter update ensure value of s' is zero
- No trajectories sampled, just state transitions

Actor–Critic Algorithms for Generalized Planning

Algorithm 1 Standard Actor-Critic for generalized planning: successor states s' sampled with probability $\pi(s'|s)$.

Input: Training MDPs $\{M_i\}_i$, each with state priors p_i
Input: Differentiable policy $\pi(s|s')$ with parameter θ
Input: Diff. value function $V(s)$ with parameter ω
Parameters: Step sizes $\alpha, \beta > 0$, discount factor γ
Initialize parameters θ and ω
Loop forever:
  Sample MDP index $i \in \{1, \ldots, N\}$
    Sample non-goal state S in M_i with probability p_i
    Sample successor state S' with probability $\pi(S'|S)$
    Let $\delta = 1 + \gamma V(S') - V(S)$
    $\omega \leftarrow \omega + \beta\delta\nabla V(S)$ Eq. (14)
    $\theta \leftarrow \theta - \alpha\delta\nabla \log \pi(S'|S)$ Eq. (8)
    If S' is a goal state, $\omega \leftarrow \omega - \beta V(S')\nabla V(S')$

Algorithm 2 All-Actions Actor-Critic: all successor states considered for updating policy and value functions.

Input: Training MDPs $\{M_i\}_i$, each with state priors p_i
Input: Differentiable policy $\pi(s|s')$ with parameter θ
Input: Diff. value function $V(s)$ with parameter ω
Parameters: Step sizes $\alpha, \beta > 0$, discount factor γ
Initialize parameters θ and ω
Loop forever:
  Sample MDP index $i \in \{1, \ldots, N\}$
    Sample non-goal state S in M_i with probability p_i
    Let $V' = 1 + \gamma\Sigma_{s' \in N(S)} [\pi(s'|S)V(s')]$
    Let $b(S) = V' - 1$ be the baseline
    $\omega \leftarrow \omega + \beta(V' - V(S))\nabla V(S)$
    $\theta \leftarrow \theta - \alpha\Sigma_{s' \in N(S)} [(V(s') - b(S))\nabla\pi(s'|S)]$
    If $s' \in N(S)$ is goal state, $\omega \leftarrow \omega - \beta V(s')\nabla V(s')$

GNN-Based General Policies

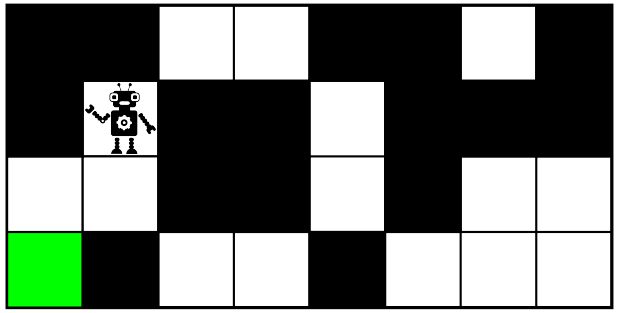
- **Parametric policy:** when at state s, for each successor s':
  - ▷ Run net to obtain object embeddings for s and s'
  - ▷ "Stack" emb., MLP, sum-aggregate, and another MLP to get (s, s') **logit**
  - ▷ **Softmax policy:** choose edge (s, s') that maximizes

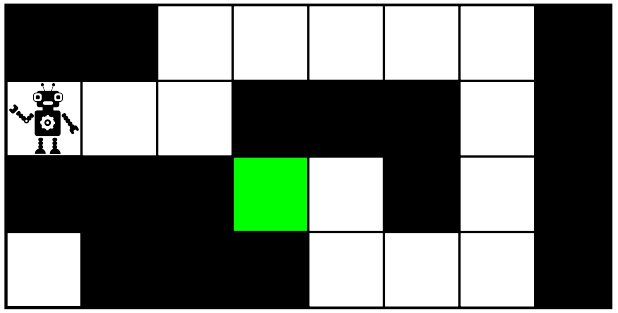
$$\frac{\exp\big(f(s,s')\big)}{\sum_{s''\in \mathrm{Succ}} \exp\big(f(s,s'')\big)}$$

Results of Actor–Critic Algorithms

	Stochastic Policy			Deterministic Policy			Validation	
Domain (#)	Coverage (%)	L	PQ = PL / OL (#)	Coverage (%)	L	PQ = PL / OL (#)	V_{π^*}	V_{π}
Standard Actor-Critic								
Blocks (23)	23 (100 %)	810	1.00 = 478 / 476 (16)	23 (100 %)	810	1.00 = 478 / 476 (16)	18.60	18.60
Blocks-multiple (26)	26 (100 %)	898	1.00 = 450 / 448 (16)	26 (100 %)	898	1.00 = 450 / 448 (16)	17.57	17.57
Delivery (24)	21 (88 %)	591	1.01 = 544 / 540 (20)	23 (96 %)	701	1.02 = 596 / 586 (21)	16.23	34.46
Grid (14)	5 (36 %)	43	1.00 = 43 / 43 (5)	10 (71 %)	516	2.68 = 356 / 133 (9)	18.13	668.28
Gripper (20)	17 (85 %)	5 031	1.00 = 176 / 176 (4)	16 (80 %)	1 312	1.00 = 176 / 176 (4)	14.89	14.89
Logistics (22)	3 (14 %)	33	1.10 = 33 / 30 (3)	17 (77 %)	37 667	77.4 = 23 839 / 308 (15)	9.04	636.43
Miconic (30)	29 (97 %)	1 438	1.00 = 185 / 185 (10)	29 (97 %)	1 438	1.00 = 185 / 185 (10)	6.42	6.42
Reward (15)	13 (87 %)	2 953	2.38 = 1 328 / 558 (7)	6 (40 %)	655	1.24 = 362 / 292 (4)	22.15	120.34
Spanner (22)	18 (82 %)	682	1.11 = 218 / 197 (7)	18 (82 %)	678	1.11 = 218 / 197 (7)	130.74	130.82
Visitall (24)	24 (100 %)	1 083	1.23 = 762 / 621 (20)	24 (100 %)	1 032	1.12 = 696 / 621 (20)	4.36	4.39
Total (220)	179 (81 %)	13 562	-	192 (87 %)	45 707	-	-	-
All-Actions Actor-Critic								
Blocks (23)	23 (100 %)	806	1.00 = 476 / 476 (16)	23 (100 %)	806	1.00 = 476 / 476 (16)	18.60	18.60
Blocks-multiple (26)	26 (100 %)	902	1.00 = 450 / 448 (16)	26 (100 %)	902	1.00 = 450 / 448 (16)	17.57	17.57
Delivery (24)	23 (96 %)	691	1.00 = 586 / 586 (21)	24 (100 %)	757	1.03 = 652 / 632 (22)	16.23	16.23
Grid (14)	6 (43 %)	71	1.00 = 71 / 71 (6)	11 (79 %)	292	1.50 = 248 / 165 (10)	18.13	84.21
Gripper (20)	20 (100 %)	1 840	1.00 = 176 / 176 (4)	20 (100 %)	1 840	1.00 = 176 / 176 (4)	14.89	14.89
Logistics (22)	0 (0 %)	-	-	8 (36 %)	7 981	63.8 = 7 981 / 125 (8)	9.04	511.76
Miconic (30)	30 (100 %)	1 527	1.00 = 185 / 185 (10)	30 (100 %)	1 527	1.00 = 185 / 185 (10)	6.42	6.42
Reward (15)	7 (47 %)	2 493	1.29 = 442 / 342 (4)	12 (80 %)	1 464	1.21 = 582 / 481 (6)	22.15	167.25
Spanner (22)	15 (68 %)	552	1.10 = 149 / 135 (5)	15 (68 %)	552	1.10 = 149 / 135 (5)	130.74	130.81
Visitall (24)	23 (96 %)	5 670	7.05 = 3 934 / 558 (19)	24 (100 %)	971	1.08 = 671 / 621 (20)	4.36	4.40
Total (220)	173 (79 %)	14 552	-	193 (88 %)	17 092	-	-	-

Grid Navigation with Decoupled Coordinates





- Navigation in grid with obstacles. Decoupled coordinates: $\textsc{cell}(x, y)$ and $\textsc{blocked}(x, y)$, position with $\textsc{at}(x, y)$, and $\textsc{adj}(i, i + 1)$ along dimensions

- For computing goal distances (eg V^*):
  - ▷ Need embedding for each cell (x, y) (quadractic in number of objects)
  - ▷ Cells (x, y) must "communicate" with neighbors (x, y') and (x', y)
  - ▷ In plain R-GNN, there must be atoms involving $\{x, y, y'\}$ (similarly, $\{x, x', y\}$). **No such atoms exist**

k-Dimensional WL

- Coloring for k-tuples of vertices rather than single vertices
- For graph $G = (V, E)$ and tuple $\langle v \rangle \in V^k$, coloring $C_{i+1}(\langle v \rangle)$:

$$C_{i+1}(\langle v \rangle) \;=\; \textsc{Relabel}\big(C_i(\langle v \rangle), M_i(\langle v \rangle)\big)$$

  where $M_i(\langle v \rangle)$ is the multiset of colored tuples

$$M_i(\langle v \rangle) \;=\; \{\!\{C_i(\phi_1(\langle v \rangle, w)), \ldots, C_i(\phi_k(\langle v \rangle, w)) \mid w \in V\}\!\}$$

  function $\phi_j(\langle v \rangle, w)$ replaces j-th component of $\langle v \rangle$ with w
- Characterization with logic C^{k+1} (at most $k+1$ vars, and counting quant.)
- Coloring computed in $\mathcal{O}(|V|^k)$ time
- Expressivity increases with k: $k = 2$ is significant jump over $k = 1$; e.g.

$$(R \bowtie S)(x, y) \;\equiv\; \exists z \big[R(x, z) \wedge S(z, y)\big]$$

Parametric Extended R-GNN$[t]$ [Stahlberg et al., 2025]

- Built on top of R-GNN architecture: R-GNN$[t](S, O) =$ R-GNN$(A_t(S), O^2)$
  - t is non-negative integer parameter: as $t \to \infty$ expressivity approaches 2-WL
  - $A_t(S) = A_0(S) \cup \Delta_t(S)$ transforms and extend atoms in S

$$
\begin{aligned}
A_0(S) &= \{p(\langle w \rangle^2) \mid p(w) \in S\} \\
\Delta_t(S) &= \{\Delta(\langle o, o' \rangle, \langle o', o'' \rangle, \langle o, o'' \rangle) \mid \langle o, o' \rangle, \langle o', o'' \rangle \in R_t\} \\
\langle w \rangle^2 &= \langle (o_1, o_1), (o_1, o_2), \ldots, (o_n, o_n) \rangle \text{ for } \langle w \rangle = \langle o_1, \ldots, o_n \rangle
\end{aligned}
$$

  where relation R_t is given by

$$
\langle o, o' \rangle \in R_t \text{ iff } \begin{cases} o \text{ and } o' \text{ are both in some atom in } S & t = 1 \\ \exists o'' \left[\{\langle o, o'' \rangle, \langle o'', o' \rangle\} \subseteq R_{t-1} \right] & t > 1 \end{cases}
$$

  - Final readout $V(S) = \sum_o \boldsymbol{f}(\langle o, o \rangle)$

- Controlled and relational version of 2-WL:
  - Embeddings for object pairs by replacing atoms in S with $A_0(S)$
  - Messages among pairs mediated by new Δ atoms (and MLP$_\Delta$)
  - t controls number of Δ-atoms via depth of compositions

Results

Domain	Model	Coverage (%)	Plan Length		
			Total	Median	Mean
Blocks-s	R-GNN	**17 / 17 (100 %)**	674	38	39
	R-GNN[0]	**17 / 17 (100 %)**	670	36	39
	R-GNN[1]	**17 / 17 (100 %)**	684	36	40
	R-GNN$_2$	14 / 17 (82 %)	922	35	65
Blocks-m	R-GNN	**22 / 22 (100 %)**	868	40	39
	R-GNN[0]	**22 / 22 (100 %)**	830	39	37
	R-GNN[1]	**22 / 22 (100 %)**	834	39	37
	R-GNN$_2$	**22 / 22 (100 %)**	936	39	42
Gripper	R-GNN	**18 / 18 (100 %)**	4,800	231	266
	R-GNN[0]	**18 / 18 (100 %)**	1,764	98	98
	R-GNN[1]	11 / 18 (61 %)	847	77	77
	R-GNN$_2$	**18 / 18 (100 %)**	1,764	98	98
Miconic	R-GNN	**20 / 20 (100 %)**	1,342	67	67
	R-GNN[0]	**20 / 20 (100 %)**	1,566	71	78
	R-GNN[1]	**20 / 20 (100 %)**	2,576	71	128
	R-GNN$_2$	**20 / 20 (100 %)**	1,342	67	67
Visitall	R-GNN	18 / 22 (82 %)	636	29	35
	R-GNN[0]	21 / 22 (95 %)	1,128	35	53
	R-GNN[1]	**22 / 22 (100 %)**	886	35	40
	R-GNN$_2$	20 / 22 (91 %)	739	33	36

Domain	Model	Coverage (%)	Plan Length		
			Total	Median	Mean
Grid	R-GNN	9 / 20 (45 %)	109	11	12
	R-GNN[0]	12 / 20 (60 %)	177	11	14
	R-GNN[1]	**15 / 20 (75 %)**	209	13	13
	R-GNN$_2$	10 / 20 (50 %)	124	11.5	12
Logistics	R-GNN	10 / 20 (50 %)	510	51	51
	R-GNN[0]	9 / 20 (45 %)	439	48	48
	R-GNN[1]	**20 / 20 (100 %)**	1,057	52	52
	R-GNN$_2$	15 / 20 (75 %)	799	52	53
Rovers	R-GNN	9 / 20 (45 %)	2,599	280	288
	R-GNN[0]	**14 / 20 (70 %)**	2,418	153	172
	R-GNN[1]	**14 / 20 (70 %)**	1,654	55	118
	R-GNN$_2$	11 / 20 (55 %)	2,225	239	202
Vacuum	R-GNN	**20 / 20 (100 %)**	4,317	141	215
	R-GNN[0]	**20 / 20 (100 %)**	183	9	9
	R-GNN[1]	**20 / 20 (100 %)**	192	9	9
	R-GNN$_2$	**20 / 20 (100 %)**	226	9	11
Visitall-xy	R-GNN	5 / 20 (25 %)	893	166	178
	R-GNN[0]	15 / 20 (75 %)	1,461	84	97
	R-GNN[1]	**20 / 20 (100 %)**	1,829	83	91
	R-GNN$_2$	19 / 20 (95 %)	2,428	116	127

- R-GNN$_2$: all atoms $\Delta(\langle o, o' \rangle, \langle o', o'' \rangle, \langle o, o'' \rangle)$
- **Takeaway:** increase expressivity without paying full cost of 2-WL

Related Work

- ASNets [Toyer et al., 2018,2020; Wang & Thiebaux, 2024]:
  - ▷ Alternate proposition-action layers of generic "modules'; specialized GNN
  - ▷ Embeddings for grounded atoms and actions, not objects
  - ▷ Integrates information from heuristics; also applied to numeric planning (2024)
- Horčík & Šír [2024] study expressiveness of GNNs in STRIPS planning
- Rosetti et al. [2024]: Learning policies using GPT models Plan generation treated as sequence-generation task
- Pereira et al. [2026]: LLM used to synthesize heuristic functions (Python) for classical planning. Counter-example guided synthesis of heuristic functions

Action Schema Networks [Toyer et al., 2018–2020]

- **Core idea:** share weights across groundings of the same action schema / predicate
- Architecture is alternating layers of action/proposition modules:
  - ▷ **Proposition modules:** one per ground atom, embeds "is this true / relevant"
  - ▷ **Action modules:** one per ground action, aggregates over propositions in precondition/effects
- Key structural difference: embeddings live on ground actions/propositions, not on objects
  - ▷ **Output is direct:** softmax/score per action, no separate readout over (single/pair) objects
  - ▷ **Cost:** receptive field grows with depth similar to R-GNN/1-WL
  - ▷ **Practical fix:** inject heuristic features (e.g. LM-cut landmarks) into the first layer
- **Training:** originally policy-gradient / exploration-and-supervision on small problems (predates the actor-critic formulation); later work [Toyer et al., 2020] broadens to probabilistic planning
- R-GNN$[t]$ enriches object-pair messages via transitive Δ-atoms, while ASNets restructures the unit of computation around actions/propositions rather than objects

Generalized Planning via GPT [Rossetti et al., 2024]

- **Core idea:** train a GPT-2 model from scratch (not fine-tuned, $\sim$83M params), per domain, to autoregressively generate a plan given a tokenized (initial state, goal) pair
- **Representation is sequence, not graph:** initial state + goal + plan flattened into a token stream:

$$\langle \text{start} \rangle \text{ fluents } \langle \text{goal} \rangle \text{ fluents } \langle \text{actions} \rangle \text{ action-tokens } \langle \text{end} \rangle$$

- Standard masked self-attention over this sequence
- **Real limitations:** **fixed vocabulary** (max #objects baked in at training time), fixed context window (2048 tokens excludes some domains entirely), and no correctness guarantee whatsoever

Wrap Up

- Generalized planning is concrete task where different threads meet:
  - ▷ Abstract representations for states, transitions, and paths
  - ▷ Logic, coloring algorithms, etc
  - ▷ Symbolic and neural methods for learning general policies
  - ▷ Symbolic and neural methods for learning STRIPS models
- Temporal abstractions (sketches) **not covered** here:
  - ▷ Rule $r = C \mapsto E$ compatible with $s_0, \ldots, s_k$ iff (s_0, s_k) is compatible with r
  - ▷ If $r = C \mapsto E$ has **width** k, then from any state s finding s' such that (s, s') is compatible with r done in time $\mathcal{O}(N^k)$ where N is number of atoms
  - ▷ Sketch of width k is set of rules, each of width k
- Briefly covered related work, references provide further links

References

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D., Patel-Schneider, P., and Nardi, D. (2003). *The description logic handbook: Theory, implementation and applications*. Cambridge U.P.

[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., 2017] Bonet, B., De Giacomo, G., Geffner, H., and Rubin, S. (2017). Generalized planning: Non-deterministic abstractions and trajectory constraints. In *Proc. IJCAI*, pages 873–879.

[Bonet et al., 2019] Bonet, B., Francès, G., and Geffner, H. (2019). Learning features and abstract actions for computing generalized plans. In *Proc. AAAI*, pages 2703–2710.

[Bonet and Geffner, 2018] Bonet, B. and Geffner, H. (2018). Features, projections, and representation change for generalized planning. In *Proc. IJCAI*, pages 4667–4673.

[Bonet and Geffner, 2020a] Bonet, B. and Geffner, H. (2020a). Learning first-order symbolic representations for planning from the structure of the state space. In *Proc. ECAI*, pages 2322–2329.

[Bonet and Geffner, 2020b] Bonet, B. and Geffner, H. (2020b). Qualitative numeric planning: Reductions and complexity. *Journal of AI Research*, 69:923–961.

[Bonet and Geffner, 2024] Bonet, B. and Geffner, H. (2024). General policies, subgoal structure, and planning width. *JAIR*, 80:475–516.

[Bonet and Geffner, 2025] Bonet, B. and Geffner, H. (2025). Learning general policies from examples. In *Proc. KR*.

[Cai et al., 1992] 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.

[Chen et al., 2026] Chen, D. Z., Hofmann, T., Klassen, T. Q., and McIlraith, S. A. (2026). Satisficing and Optimal Generalised Planning via Goal Regression. In *Proc. AAAI*.

[De Giacomo et al., 2016] De Giacomo, G., Murano, A., Rubin, S., and Di Stasio, A. (2016). Imperfect-information games and generalized planning. In *IJCAI*, pages 1037–1043.

[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*.

[Groshev et al., 2018] Groshev, E., Goldstein, M., Tamar, A., Srivastava, S., and Abbeel, P. (2018). Learning generalized reactive policies using deep neural networks. In *Proc. ICAPS*.

[Groshev et al., 2017] Groshev, E., Tamar, A., Srivastava, S., and Abbeel, P. (2017). Learning generalized reactive policies using deep neural networks. *arXiv preprint arXiv:1708.07280*.

[Hofmann and Geffner, 2024] Hofmann, T. and Geffner, H. (2024). Learning generalized policies for fully observable non-deterministic planning domains. In *Proc. IJCAI*, pages 6733–6742.

[Horčík and Šír, 2024] Horčík, R. and Šír, G. (2024). Expressiveness of graph neural networks in planning domains. In *Proc. ICAPS*, pages 281–289.

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

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

[Martins et al., 2014] Martins, R., Manquinho, V., and Lynce, I. (2014). Open-WBO: A modular MaxSAT solver. In *Proc. SAT*, pages 438–445.

[Morris et al., 2019] 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*, pages 4602–4609.

[Rodriguez et al., 2021] 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 et al., 2008] 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.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

[Segovia-Aguas et al., 2021] Segovia-Aguas, J., Jiménez, S., and Jonsson, A. (2021). Generalized planning as heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 31, pages 313–321.

[Segovia-Aguas et al., 2022] Segovia-Aguas, J., Jiménez, S., and Jonsson, A. (2022). Computing programs for generalized planning as heuristic search. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4790–4796.

[Segovia-Aguas et al., 2024] Segovia-Aguas, J., Jiménez, S., and Jonsson, A. (2024). Generalized planning as heuristic search: A new planning search-space that leverages pointers over objects. *Artificial Intelligence*, 328:120593.

[Srivastava, 2023] Srivastava, S. (2023). Hierarchical decompositions and termination analysis for generalized planning. *Journal of Artificial Intelligence Research*, 77:1223–1256.

[Srivastava et al., 2008] Srivastava, S., Immerman, N., and Zilberstein, S. (2008). Learning generalized plans using abstract counting. In *Proc. AAAI*, pages 991–997.

[Srivastava et al., 2011a] Srivastava, S., Immerman, N., and Zilberstein, S. (2011a). A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2):615–647.

[Srivastava et al., 2011b] Srivastava, S., Zilberstein, S., Immerman, N., and Geffner, H. (2011b). Qualitative numeric planning. In *AAAI*.

[Ståhlberg et al., 2021] Ståhlberg, S., Bonet, B., and Geffner, H. (2021). Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. *arXiv preprint arXiv:2109.10129*.

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

[Ståhlberg et al., 2022b] Ståhlberg, S., Bonet, B., and Geffner, H. (2022b). Learning generalized policies without supervision using gnns. In *Proc. KR*.

[Ståhlberg et al., 2023] Ståhlberg, S., Bonet, B., and Geffner, H. (2023). Learning general policies with policy gradient methods. In *Proc. KR*.

[Ståhlberg et al., 2025] Ståhlberg, S., Bonet, B., and Geffner, H. (2025). Learning more expressive general policies for classical planning. In *Proc. AAAI*.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

[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., 2020] Toyer, S., Thiébaux, S., Trevizan, F., and Xie, L. (2020). Asnets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68:1–68.

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

[Wang and Thiébaux, 2024] Wang, R. X. and Thiébaux, S. (2024). Learning generalised policies for numeric planning. In *Proc. ICAPS*, pages 633–642.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256.

[Xu et al., 2019] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *ICLR*.