Representation Learning for Acting and Planning: A Top Down Approach

Tutorial ICAPS 2022

Blai Bonet
Universitat Pompeu Fabra, Barcelona

Hector Geffner
ICREA & Universitat Pompeu Fabra
Wallenberg Guest Professor, Linköping University

With inputs from Dominik, Simon, Andrés; RLeap Team (UPF, LiU)
Deep learning (DL) and Deep Reinforcement Learning (DRL) have revolutionised the landscape of AI, exploiting power of stochastic gradient descent.

Yet DL and DRL struggle with OOD/structural generalization:
- Inductive biases in neural architectures assumed to help but vague, informal.

Alternative: Language-based representation learning:
- Don’t choose low-level arch and expect “right representation” to emerge.
- Choose high-level language instead, and learn representations over language.

Separation between what is to be learned and how.

“... Systematic generalization hypothesized to arise from efficient factorization of knowledge into recomposable pieces corresponding to reusable factors ...”

Language-based representation learning:
- learn the “recomposable pieces” in a language
- recombinations and generalization will follow semantics

Very much in line with traditional AI: just learn from data the representations that have traditionally been crafted by hand

Potential benefits: meaningful learning bias, semantics, transparency, reasoning
Example: Minigrid/BabyAI [Chevalier-Boisvert et al., 2019]

▶ **Task:** *Pick up grey box behind you, then go to grey key and open door*

▶ Red triangle is agent at bottom right. Light-grey is field of view

▶ Learn **controller** that accepts **goals** and **obs**, and outputs **action** to do

▶ Like a “classical planning problem” **but** state representation **not known**, and goals to be achieved **reactively** (not by planning) with policies that **generalize**

B. Bonet, H. Geffner. Language-based Representation Learning for Acting and Planning. ICAPS/IJCAI 2022 Tutorial
DRL vs. Language-based Representation Learning

• Surprise is not that DL and DRL methods struggle in Minigrid, but that they manage to generate meaningful behavior at all, given so little prior knowledge.

• Yet methodology largely ad hoc: from intuitions to architectures and experiments using baselines . . .

• From perspective of language-based representation learning, key questions are:
  ▶ What are the domain-independent languages for representing dynamics?
  ▶ What the languages for representing general reactive policies, subgoals?
  ▶ How can representations over such languages be learned?
Outline of the Tutorial

• **Background 1:** Classical planning, planning **width**

• **Languages** for
  ▶ representing general **dynamics**
  ▶ representing general **policies**
  ▶ representing general **subgoal structures** (sketches; ‘intrinsic rewards’)

• **Background 2:** Qualitative numerical planning problems (**QNPs**)  

• **Learning** representations over these languages:
  ▶ learning general **dynamics**
  ▶ learning general **policies**
  ▶ learning general **subgoal structures**

• **Wrap up; Challenges**

Copy of these slides at https://www.dtic.upf.edu/~hgeffner/tutorial-2022.pdf
Outline of the Tutorial (2)

- Tutorial is **not a survey** on learning to act and plan; too much for us; too much for 1:30h

- Focus is on a **coherent** research thread that covers a lot of ground:
  - **Crisp** and **simple** ideas and formulations for **stating**, **understanding**, and **addressing** key problems

- Many **open problems**; many opportunities for research
Background 1:
Classical Planning and Planning Width
Background: Model for Classical AI Planning

A (classical) **state model** is a tuple \( S = (S, s_0, S_G, Act, A, f, c) \):

- finite and discrete **state space** \( S \)
- a known **initial state** \( s_0 \in S \)
- a set \( S_G \subseteq S \) of **goal states**
- **actions** \( A(s) \subseteq Act \) **applicable** in each \( s \in S \)
- a **deterministic state-transition function** \( s' = f(a, s) \) for \( a \in A(s) \)
- positive **action costs** \( c(a, s) \), assumed 1 by default

A **solution** to the model or **plan** is a sequence of applicable actions \( a_0, \ldots, a_n \) that maps \( s_0 \) into \( S_G \)

i.e. there must be state sequence \( s_0, \ldots, s_{n+1} \) such that \( a_i \in A(s_i), s_{i+1} = f(a_i, s_i) \), and \( s_{n+1} \in S_G \)

B. Bonet, H. Geffner. Language-based Representation Learning for Acting and Planning. ICAPS/IJCAI 2022 Tutorial
A Language for Classical Planning: STRIPS

• A (grounded) problem in STRIPS is a tuple \( P = \langle F, O, I, G \rangle \):
  - \( F \) is set of (ground) atoms
  - \( O \) is set of (ground) actions
  - \( I \subseteq F \) stands for initial situation
  - \( G \subseteq F \) stands for goal situation

• Actions \( o \in O \) represented by
  - **Add** list \( \text{Add}(o) \subseteq F \)
  - **Delete** list \( \text{Del}(o) \subseteq F \)
  - **Precondition** list \( \text{Pre}(o) \subseteq F \)

A problem \( P \) in STRIPS defines state model \( S(P) \) in compact form . . .
STRIPS problem $P = \langle F, O, I, G \rangle$ determines state model $S(P)$ where

- the states $s \in S$ are collections of atoms from $F$
- the initial state $s_0$ is $I$
- the goal states $s_G$ are such that $G \subseteq s_G$
- the actions $a$ in $A(s)$ are ops in $O$ s.t. $Prec(a) \subseteq s$
- the next state is $s' = [s \setminus Del(a)] \cup Add(a)$
- action costs $c(a, s)$ are all 1

Common approach for solving $P$ is using path-finding/heuristic search algorithms over graph defined by $S(P)$ where nodes are the states $s$, and edges $(s, s')$ are state transitions caused by an action $a$; i.e., $s' = f(a, s)$ and $a \in A(s)$

The source node is the initial state $s_0$, and the targets are the goal states $s_G$
Background: Width and Width-based Algorithms

- IW(1) is a **breadth-first search** that **prunes** states \( s \) that don’t make a **feature** true for first time in the search, given set of Boolean features \( F \)
  - In **classical planning**, \( F \) is the set of (ground) atoms in problem

- IW(\( k \)) is IW(1) but over set \( F^k \) made up of conjunctions of \( k \) features from \( F \)

- **Alternatively**, IW(\( k \)) is a breadth-first search that prunes \( s \) if \text{novelty}(s) > k

- IW runs IW(1), IW(2), ..., IW(\( k \)) sequentially until problem solved or \( k = N \)

- IW is blind like DFS and BFS but diff **enumeration**; uses **state structure**

- IW(\( k \)) expands up to \( N^k \) nodes and runs in **polytime** \( \exp(2k - 1) \)
# Planning for *Atomic Goals* with IW(1) and IW(2)

<table>
<thead>
<tr>
<th>#</th>
<th>Domain</th>
<th>I</th>
<th>IW(1)</th>
<th>IW(2)</th>
<th>Neither</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>8puzzle</td>
<td>400</td>
<td>55%</td>
<td>45%</td>
<td>0%</td>
</tr>
<tr>
<td>2.</td>
<td>Barman</td>
<td>232</td>
<td>9%</td>
<td>0%</td>
<td>91%</td>
</tr>
<tr>
<td>3.</td>
<td>Blocks World</td>
<td>598</td>
<td>26%</td>
<td>74%</td>
<td>0%</td>
</tr>
<tr>
<td>4.</td>
<td>Cybersecure</td>
<td>86</td>
<td>65%</td>
<td>0%</td>
<td>35%</td>
</tr>
<tr>
<td>5.</td>
<td>Pegsol</td>
<td>964</td>
<td>92%</td>
<td>0%</td>
<td>8%</td>
</tr>
<tr>
<td>6.</td>
<td>Pipes-NonTan</td>
<td>259</td>
<td>44%</td>
<td>56%</td>
<td>0%</td>
</tr>
<tr>
<td>7.</td>
<td>Pipes-Tan</td>
<td>369</td>
<td>59%</td>
<td>37%</td>
<td>3%</td>
</tr>
<tr>
<td>8.</td>
<td>PSRsmall</td>
<td>316</td>
<td>92%</td>
<td>0%</td>
<td>8%</td>
</tr>
<tr>
<td>9.</td>
<td>Rovers</td>
<td>488</td>
<td>47%</td>
<td>53%</td>
<td>0%</td>
</tr>
<tr>
<td>10.</td>
<td>Satellite</td>
<td>308</td>
<td>11%</td>
<td>89%</td>
<td>0%</td>
</tr>
<tr>
<td>11.</td>
<td>Scanalyzer</td>
<td>624</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>12.</td>
<td>Transport</td>
<td>330</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>13.</td>
<td>Trucks</td>
<td>345</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>14.</td>
<td>Visitall</td>
<td>21,859</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>15.</td>
<td>Woodworking</td>
<td>1659</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>16.</td>
<td>Zeno</td>
<td>219</td>
<td>21%</td>
<td>79%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td><strong>Total/Avg</strong></td>
<td><strong>37,921</strong></td>
<td><strong>37.0%</strong></td>
<td><strong>51.3%</strong></td>
<td><strong>11.7%</strong></td>
</tr>
</tbody>
</table>

88.3% of the 37,921 instances solved by IW(1) or IW(2) [Lipovetzky and G., 2012]
Performance of IW is No Accident: Theory

- **Width** of $P$, $w(P)$, is $\text{min } k$ for which there is a sequence of **subgoals** (atom tuples) $t_0, t_1, \ldots, t_n$, $|t_i| \leq k$ such that:
  - $t_0$ is true in the initial situation
  - the optimal plans for $t_n$ are optimal plans for $P$
  - all **optimal plans for $t_i$ can be extended into optimal plans for $t_{i+1}$** by adding a **single action**

- Also $w(P) = 0$ if goal reachable in 0 or 1 step; $w(P) = N + 1$ if no solution, where $N$ is number of atoms in $P$.

- **Theorem:** If $w(P) = k$, then IW($k$) solves $P$ optimally in $\exp(2k - 1)$ time

- **Theorem:** Domains like Blocks, Logistics, Gripper, . . . have all width 2 independent of problem **size** provided that goals are **single atoms**

B. Bonet, H. Geffner. Language-based Representation Learning for Acting and Planning. ICAPS/IJCAI 2022 Tutorial
Practical Variations of IW

**SIW**: Serialized iterated width [Lipovetzky and G., 2012]

- Use IW greedily to decrease **number of unachieved goals** \(#g\); assumes conjunctive top goal (simple goal serialization)

**BFWS**: Best-first guided by **novelty measure** \(w(\#g, \#c)\) and \(#g\)

- BFWS\((f_5)\): back-end of state-of-the-art Dual-BFWS, \(#c\) from relaxed plans
- \(k\)-BFWS\((f_5)\): **poltytime** variant of BFWS\((f_5)\) used as front-end of Dual-BFWS
- BFWS\((R)\): version that does not use **action structure**; just **PDDL simulator**

[Lipovetzky and G., 2017; Francès et al., 2017]
How to prove in standard encodings that:

- Blocks world instances with goal $\text{clear}(x)$ or $\text{hold}(x)$ have width $1$
- Delivery instances with goal $\text{hold}(x)$ or $\text{AgentAt}(y)$ have width $1$
- Blocks world instances with goal $\text{on}(x,y)$ have width $2$
- Delivery instances with goal $\text{PkgAt}(x,y)$ have width $2$
- Blocks and Delivery with arbitrary conjunctive goals have no bounded width

Delivery is simplified Logistics: agent in grid, picking up and dropping pkgs

For proving $w(G) \leq k$:

- **Necessary 1:** If $a_1, \ldots, a_n$ is optimal plan for goal $G$, each prefix $a_1, \ldots, a_i$ must be optimal plan for some $t_i$, $|t_i| \leq k$

- **Necessary 2:** For these $t_i$’s, all optimal plans for $t_i$ extend into optimal plans for $t_{i+1}$.
Part II: Languages

- Language for expressing **dynamics**
- Language for expressing **general policies**
- Language for expressing **general subgoal structures**
Language for Expressing Dynamics: First-Order STRIPS

Problems specified as instances $P = \langle D, I \rangle$ of 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:

- Learning $P_i = \langle D, I_i \rangle$ implies learning $D$ that generalizes to other instances

In RL and DRL, there is no notion of domain: generalization to other “instances” analyzed experimentally; closest things are “procedurally generated instances,” and “probability distribution over tasks”
Example: 2-Gripper Problem \( P = \langle D, I \rangle \) in PDDL

(define (domain gripper)
  (:requirements :typing)
  (:types room ball gripper)
  (:constants left right - 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)
    :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))

(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))))
Learning Dynamics in Lifted STRIPS

- Planning problem $P_i = \langle D, I_i \rangle$ defines unique state graph $G(P_i)$
- Learning as inverse problem: from graphs $G_1, \ldots, G_k$, learn $D, I_i$:

  Given graphs $G_1, \ldots, G_k$, find simplest instances $P_i = \langle D, I_i \rangle$ such that graphs $G_i$ and $G(P_i)$ are isomorphic, $i = 1, \ldots, k$.

- Problem cast and solved as combinatorial optimization task [B. and G., 2020]
- Complexity of $D$ determined by $\#$ and arities of action schemas and predicates
- Variations: missing edges, noisy observations [Rodriguez et al., 2021a]
- Related
  - Learning schemas from ground traces [Cresswell et al., 2013]
  - Deep learning of action schemas from images via autoencoders [Asai, 2019]
  - Learning prop. action models from options [Konidaris et al., 2018]
  - Most work on learning action models assumes domain predicates known
Second Task: General Policies

- **General policy** represents strategy for solving **multiple** domain instances **reactively**; i.e., without having to search or plan
  - E.g., policy for achieving $\text{on}(x, y)$; **any #** of blocks, **any** configuration

- What are good **languages** for expressing such policies?

- Number of works have addressed the problem [Khardon 1999; Martin and G., 2004; Fern *et al.*, 2006; Srivastava *et al.*, 2011; Hu and De Giacomo, 2011]

- **Subtlety:** set of (ground) actions change from instance to instance with objects

Learning general policies also a key goal in (Deep) RL
General Policies: A Language [B. and G., 2018]

- **General policies** are given by rules $C \mapsto E$ over set $\Phi$ of **features**

- **Features** $f$ are state functions that have a well-defined value $f(s)$ on every reachable state of any instance of the domain
  
  ▶ **Boolean** features $p$: $p(s)$ is true or false
  
  ▶ **Numerical** features $n$: $n(s)$ is non-negative integer

Computation of feature values assumed to be “cheap”: features assumed to have **linear** number of values at most, computable in **linear** time (in $|P|$).
Example: General Policy for $clear(x)$

- **Features** $\Phi = \{H, n\}$: 'holding' and 'number of blocks above $x$'

- **Policy** $\pi$ for class $Q$ of Block problems with goal $clear(x)$ given by two rules:

  \[
  \{\neg H, n > 0\} \mapsto \{H, n\downarrow\} \quad ; \quad \{H, n > 0\} \mapsto \{\neg H\}
  \]

**Meaning:**

- if $\neg H$ & $n > 0$, move to successor state where $H$ holds and $n$ decreases
- if $H$ & $n > 0$, move to successor state where $\neg H$ holds, $n$ doesn’t change
Language and Semantics of General Policies: Definitions

- **Policy rules** $C \rightarrow E$ over set $\Phi$ of Boolean and numerical **features** $p, n$:
  - *Boolean conditions* in $C$: $p, \neg p, n = 0, n > 0$
  - *qualitative effects* in $E$: $p, \neg p, p?, n\downarrow, n\uparrow, n?$

- **State transition** $(s, s')$ satisfies rule $C \rightarrow E$ if
  - $f(s)$ makes body $C$ true
  - change from $f(s)$ to $f(s')$ satisfies $E$

- A **policy** $\pi$ for class $Q$ of problems $P$ is given by policy rules $C \rightarrow E$
  - *Transition* $(s, s')$ in $P$ compatible with $\pi$ if $(s, s')$ satisfies a policy rule
  - *Trajectory* $s_0, s_1, \ldots$ compatible if $s_0$ of $P$ and transitions compatible with $\pi$

- $\pi$ **solves** $P$ if all max trajectories compatible with $\pi$ reach goal of $P$

- $\pi$ **solves** collection of problems $Q$ if it solves each $P \in Q$
Example: Delivery

- Pick packages spread in $n \times m$ grid, one by one, to target location

- **Features** $\Phi = \{H, p, t, n\}$: hold, dist. to nearest pkg & target, $\#$ undelivered

- Policy $\pi$ that solves class $Q_D$: any $\#$ of pkgs and distribution, any grid size

\[
\begin{align*}
\{\neg H, p > 0\} & \mapsto \{p\downarrow, t?\} & \text{go to nearest package} \\
\{\neg H, p = 0\} & \mapsto \{H, p?\} & \text{pick it up} \\
\{H, t > 0\} & \mapsto \{t\downarrow, p?\} & \text{go to target cell} \\
\{H, t = 0\} & \mapsto \{\neg H, n\downarrow, p?\} & \text{drop package}
\end{align*}
\]
General Policies: Three Questions

1. How to prove that general policy solves potentially infinite class of instances $Q$?

2. How to learn policies (and the features involved) to solve $Q$?

3. How to learn policies that are guaranteed to solve infinite $Q$?

We consider idea of learning first and move then to 1. Not much to say about 3.
Given a known domain $D$, training instances $P_1, \ldots, P_k$, over $D$, and a finite pool of domain features $\mathcal{F}$, each with a cost, find the cheapest policy $\pi$ over $\mathcal{F}$ such that $\pi$ solves all $P_i, i = 1, \ldots, k$

- Problem cast and solved as **combinatorial opt. task** [Francès et al., 2021]
- Pool of **features** $\mathcal{F}$ generated from domain predicates using **2-variable** (description) logic grammar; feature cost given by syntax tree size
- **Deep learning** approaches [Toyer et al., 2018; Garg et al., 2020] do not need $\mathcal{F}$ but not 100% correct in general
- Recent DL approach also avoids $\mathcal{F}$ and nearly 100% correct when **2-variable logic** features suffice; exploits relation between **GNNs** and 2-variable logic [Ståhlberg et al., 2022a and 2022b]
How to prove that this policy $\pi$ achieves $\text{clear}(x)$ in all Block problems?

$\{\neg H, n > 0\} \mapsto \{H, n\downarrow\}$ \quad ; \quad $\{H, n > 0\} \mapsto \{\neg H\}$

- **Soundness**: policy $\pi$ applies in every non-goal state $s$
  
  $\rightarrow$ for any such $s$, there is $(s, s')$ compatible with $\pi$

- **Acyclicity**: no sequence of transitions $(s_i, s_{i+1})$ compatible with $\pi$ cycle

**Theorem**: If $\pi$ is sound and acyclic in $Q$, and no dead-ends, $\pi$ solves $Q$

**Exercise**: Show that policy for $\text{clear}(x)$ is sound and acyclic in Blocks
Acyclicity, Termination, and QNPs

- **Termination**: criterion that ensures that policy is **acyclic** over any domain

- A policy $\pi$ is **terminating** if for all infinite trajectories $s_0, \ldots, s_i, \ldots$ compatible with $\pi$, there is a **numerical feature** $n$ such that:
  
  $\triangleright$ $n$ is **decremented** in some recurrent transition $(s, s')$; i.e., $n(s') < n(s)$
  
  $\triangleright$ $n$ is **not incremented** in any recurrent transition $(s, s')$; i.e., $n(s') \not> n(s)$

- Every such trajectory deemed **impossible** or **unfair** ($n$ can’t decrement below 0), thus if $\pi$ terminates, $\pi$-trajectories **terminate**

- **Termination** notion is from **QNPs**; verifiable in time $O(2^{\mid \Phi \mid})$ by **SIEVE** algorithm [Srivastava et al., 2011], where $\Phi$ is set of features involved in the policy

More about QNPs later on . . .
Third Task: Subgoal Structure

Subgoal structure important in planning and RL ("intrinsic rewards", hierarchies)

Sketches powerful language for expressing subgoal structure [B. and G., 2021]

- Goal serialization and full policies expressible as sketches
- Semantics in terms of subgoals to be achieved; not so with HTNs, LTL
- Sketches split problems into subproblems

If subproblems have a bounded width, problems solved in polytime
Example: Sketches for Delivery

- **Width=0** Sketch (full policy)

  \[
  \{\neg H, p > 0\} \mapsto \{p\downarrow, t?\} \quad \text{go to nearest package}
  \]

  \[
  \{\neg H, p = 0\} \mapsto \{H, p?\} \quad \text{pick it up}
  \]

  \[
  \{H, t > 0\} \mapsto \{t\downarrow, p?\} \quad \text{go to target cell}
  \]

  \[
  \{H, t = 0\} \mapsto \{\neg H, n\downarrow, p?\} \quad \text{drop package}
  \]

- **Width=2** Sketch:

  \[
  \{n > 0\} \mapsto \{n\downarrow\} \quad \text{deliver package}
  \]

- **Width=1** Sketch:

  \[
  \{\neg H\} \mapsto \{H\} \quad \text{go and pick package}
  \]

  \[
  \{H\} \mapsto \{\neg H, n\downarrow\} \quad \text{go and deliver package}
  \]

**Features:** Holding \((H)\); Dist. to nearest Pkg \((p)\), Target \((t)\); \# Undeliv Pkgs \((n)\)
Syntax and Semantics of Sketch Rules

- **Syntax**: For Boolean and numerical features $p$ and $n$:
  - $p, \neg p, n > 0, n = 0$ can appear in $C$
  - $p, \neg p, n\uparrow, n\downarrow, n?$ can appear in $E$

- **Semantics**: State pair $(s, s')$ satisfies sketch rule $C \mapsto E$ if
  - $f(s)$ satisfies $C$
  - $(f(s), f(s'))$ satisfies $E$

Syntax of sketches and policies the same, and so with semantics, except that $(s, s')$ is not a 1-step state transition necessarily

**Interpretation**: When in state $s$, the set of subgoal states $G_R(s)$ to aim at is:

$$G_R(s) = \{ s' \mid (s, s') \text{ satisfies sketch rule or } s' \text{ is goal} \}$$
Sketch Width

- Sketch $R$ splits problems $P$ in $Q$ into collection of subproblems $P[s, G_R(s)]$:
  - Initial state $s$: reachable state $s$ in $P$
  - (Sub) goal states $G_R(s) = \{ s' \vert (s, s')$ satisfies sketch rule or $s'$ is goal $\}$

- Width of sketch $R$ over $Q = \max_{s, P \in Q} \text{width}(P[s, G_R(s)])$
  - for definition in presence of dead-ends, see refs

**Theorem:** Any $P$ in $Q$ is solvable in $O(b \cdot N^{\Phi} + 2^k - 1)$ time by $\text{SIW}_R$ algorithm if sketch $R$ is terminating and has width over $Q$ bounded by $k$ [B. and G., 2021]
  - $N$: Number of atoms in problem $P$  ;  $\Phi$: Set of features in sketch

$\text{SIW}_R$ is like $\text{SIW}$ but subgoal to achieve next given by sketch

- $\text{SIW}$ is $\text{SIW}_R$ with sketch $R$ with single rule: $\{ \#g > 0 \} \mapsto \{ \#g \downarrow \}$
Another Example: IPC Grid [Drexler et al., 2021]

This sketch is **terminating** and has width 1 for IPC domain Grid (pick and deliver keys spread in grid where cells can be locked and opened with other keys):

- **Sketch:**
  - \( r_1 : \{ l > 0 \} \mapsto \{ l \downarrow, k?, o?, t? \} \) (if locked cells, unlock them)
  - \( r_2 : \{ l = 0, k > 0 \} \mapsto \{ k \downarrow, o?, t? \} \) (else, place keys in targets)
  - \( r_3 : \{ l > 0, \neg o \} \mapsto \{ o, t? \} \) (if locked cells, pick key to open locked cell)
  - \( r_4 : \{ l = 0, \neg t \} \mapsto \{ o?, t \} \) (if all locks open and misplaced keys, pick up such key)

- **Features:**
  - \( l \) is the number of unlocked grid cells
  - \( k \) is the number of misplaced keys
  - \( o \) is true iff robot holds key for which there is a closed lock
  - \( t \) is true iff robot holds key that must be placed at some target grid cell
Given a known domain $D$, training instances $P_1, \ldots, P_n$, and non-negative integer $k$, find simplest sketch $R$ over a pool of features $F$ such that

- Subproblems induced by $R$ on each $P_i$ have all \textbf{width bounded} by $k$,
- Sketch $R$ is \textbf{terminating}

Possibly first approach for \textbf{learning subgoal structure} based on crisp \textbf{principles}

\textbf{Many threads that come together:}

- Planning \textbf{width}
- Language of \textbf{general policies}
- Termination notion from \textbf{QNPs}
- Semantics of \textbf{sketches}
In the 1985 AIJ paper, *Macro-Operators: A Weak Method for Learning*, Rich Korf provides **macro-tables** for puzzles like Rubik Cube, 8-puzzle, and other hard puzzles that encode **policies** $\pi(s)$ for solving them from any initial state.

- Can these compact policies be replaced by even more compact **sketches** of **bounded width**?

- Can these sketches be **general**? That is, applicable to Rubik cubes and $n$-sliding puzzles of **different sizes**?

- Can such sketches be **learned** with current method? **Expressivity**? **Scalability**? Other methods?
Background 2:
Qualitative Numerical Planning Problems (QNPs)
Language for QNPs

- Language for planning involving **propositional** and **numerical variables**

- QNPs [Srivastava et al. 2011] different than **numerical planning**:
  - Numerical vars in QNPs are non-negative, **real-valued**
  - **Effects** on numerical variables: just **qualitative** increments/decrements
  - **Numerical literals**: whether variable is **zero** or **positive** only

- These differences make plan-existence for QNPs **decidable**

- QNPs provide language for **general policies and sketches**:
  - QNP actions similar to policy/sketch rules but **features** replaced by **variables**

- We follow [B. and G., 2020b]
Syntax for QNPs

A **qualitative numerical problem (QNP)** is tuple $Q = \langle F, V, I, O, G \rangle$:

- $F$ and $V$ are sets of propositional and numerical variables (not features!)
- $I$ and $G$ denote initial and goal states
- $O$: actions $a$ with precs, and prop. and numeric effects $Pre(a)$, $Eff(a)$, $N(a)$:
  - $F$-literals may appear in $I$, $G$, $Pre(a)$ and $Eff(a)$
  - $V$-literals may appear in $I$, $G$ and $Pre(a)$
  - $N(a)$ can only have expressions of the form $X^\uparrow$ and $X^\downarrow$ for var $X$ in $V$
- $V$-literal is either $X = 0$ or $X > 0$ for variable $X$ in $V$

**Example:** QNP $Q_{clear} = \langle \{H\}, \{n\}, I, O, G \rangle$

- $I = \{n > 0, \neg H\}$
- $G = \{n = 0\}$
- $O = \{a, b\}$ where $a = \{\neg H, n > 0\} \mapsto \{H, n^\downarrow\}$ and $b = \{H\} \mapsto \{\neg H\}$

- QNP actions like policy rules above but $H$ and $n$ not features but variables
Semantics and Solutions of QNPs

- Policy $\pi$ for a QNP is partial map from state $s$ into actions such that:
  - $\pi(s) = \pi(s')$ if $s$ and $s'$ qualitatively similar: same $F$ and $V$ true literals

- $\pi$ solves QNP if all maximal QNP-fair $\pi$-trajectories reach the goal
  - QNP fairness: trajectory unfair if numerical variable decremented infinite number of times and incremented finite number of times.

Theorem [Srivastava et al., 2011]: $\pi$ solves QNP $Q$ iff $\pi$ is strong cyclic solution of the FOND problem $T_D(Q)$ obtained from $Q$ that terminates

- $T_D(Q)$ replaces numerical $X$ by Boolean variables “$X=0$” and “$X>0$”
- Qualitative effects $X\uparrow$ replaced by effect $X>0$
- Qualitative effects $X\downarrow$ replaced by non-deterministic effect “$X>0 \mid X=0$”
- Strong-cyclic: every reachable state is connected to goal state by $\pi$

Polytime reduction from QNPs to FOND, but more complex than $T_D$ [B. and G., 2020b]
Termination, Sieve Algorithm [Srivastava et al., 2011]

Policy for QNP $Q$ **terminates** if no infinite **QNP-fair** $\pi$-trajectories

Sieve provides **sound** and **complete** polynomial termination test

- State $s$ **terminates** if
  - there is no cycle on state $s$,
  - every cycle on $s$ contains a state $s'$ that terminates, or
  - $\pi(s)$ decrements a variable $X$, and every cycle on $s$ that contains a state $s'$ such that $\pi(s')$ increments $X$, contains a state $s''$ that terminates

- Policy $\pi$ terminates iff every state reached by $\pi$ terminates

Recent FOND$^+$ planner handles strong FOND, strong cyclic FOND, QNPs, and hybrids by stating **fairness assumptions** explicitly [Rodriguez et al. 2021b]
Part III: Learning Dynamics, Policies, Sketches

- **Learning action models:**
  
  Given graphs $G_1, \ldots, G_k$, find **simplest** instances $P_i = \langle D, I_i \rangle$ such that graphs $G_i$ and $G(P_i)$ are isomorphic, $i = 1, \ldots, k$.

- **Learning general policies:**

  Given known domain $D$, training instances $P_1, \ldots, P_k$, over $D$, and **finite pool of domain features** $\mathcal{F}$, each with a cost, find the cheapest policy $\pi$ over $\mathcal{F}$ such that $\pi$ solves all $P_i$, $i = 1, \ldots, k$.

- **Learning sketches:**

  Given known domain $D$, training instances $P_1, \ldots, P_n$, and non-negative integer $k$, find simplest sketch $R$ over a pool of features $\mathcal{F}$ such that
  
  - Subproblems induced by $R$ on each $P_i$ have all **width bounded** by $k$,
  - Sketch $R$ is **terminating**
Learning Action Models: Encoding [Rodriguez et al., 2021a]

- Construct **answer set program**, bounding number of objects and action/predicate arities:
  - **Given** $G_1, \ldots, G_k$ as input graphs over **black-box states**, with edge labels,
  - **Check** whether there is STRIPS model $D$ and instances $I_1, \ldots, I_k$ such that graphs $G(P_i)$ and $G_i$ are **isomorphic**, $i = 1, \ldots, k$, where $P_i = \langle D, I_i \rangle$
  - **Optimize**: sum of action and predicate arities, . . .

- **Choice variables** in program:
  - Lifted precs/effects for each action schema (schemas determined by labels in input graphs)
  - Values of ground atoms at each state
  - Assignment of applicable grounded actions to edges in input graphs

- **Constraints** in program:
  - Different nodes in each input graph maps to different valuations of grounded atoms
  - Every edge in input graph “receives” a grounded action (establishing isomorphism)
  - Compliance of precs/effects of assigned grounded actions to edges

- **Clingo** program $\sim$ 400 lines [Rodriguez et al. 2021a]; more complex in SAT [B. and G., 2020a]
Learning General Policies: Encoding [Francès et al., 2021]

- **Input** is set of transitions $S$ from small instances, pool of features $F$, integer $\delta$
- **Output** is policy: rules obtained from selected features and (“good”) transitions
- **Combinatorial opt. task** $T(S, F, \delta)$: Solve constraints minimizing feature complexity

▷ **Choice variables:** $select(f)$, $good(s, s')$ and $V(s, d)$

▷ **Constraints:**

1. $\bigvee \{ good(s, s') : (s, s') \in S \}$  
   (Good transition at each non-terminal $s$)
2. $\neg good(s, s')$  
   (No good reach dead-end $s'$)
3. $\bigvee \{ select(f) : f$ such $f[s] \neq f[s'] \}$  
   (Distinguish $\{s, s'\}$ when exactly-1 is goal)
4. Exactly-1 $\{ V(s, d) : V^*(s) \leq d \leq \delta V^*(s) \}$  
   (Set distances)
5. $good(s, s') \land V(s, d) \land V(s', d') \rightarrow d < d'$  
   (Distances avoid cycles)
6. $good(s, s') \land \neg good(t, t') \rightarrow D2(s, s'; t, t')$  
   (Distinguish good/bad transitions)

where $D2(s, s'; t, t') = \bigvee_{f : \Delta_f(s, s') \neq \Delta_f(t, t')} select(f)$ and $\Delta_f(s, s') \in \{\uparrow, \downarrow, =\}$
Learning General Sketches: Encoding [Drexler et al., 2022]

- **Input:** transitions $S$ in small instances, pool $\mathcal{F}$, width bound $k$, max # sketch rules $m$
- **Output:** sketch of width $\leq k$, acyclic in given instances, with up to $m$ rules
- **Combinatorial opt. task** $T(S, \mathcal{F}, k, m)$: solve constraints min complexity of selected features

- **Variables:**
  1. $\text{select}(f)$
  2. $\text{cond}(i, f, v)$
  3. $\text{eff}(i, f, e)$
  4. $\text{satisfies}(s, s', i)$
  5. $\text{subgoal}(s, t)$
  6. $\text{subgoals}(s, t, s')$

- **Constraints:**
  1. $\text{cond}(i, f, v)$ and $\text{eff}(i, f, e)$ use unique $v$, imply $\text{select}(f)$
  2. $\bigvee_t \text{subgoal}(s, t)$
  3. $\text{subgoal}(s, t) \leftrightarrow \text{subgoals}(s, t, s')$
  4. $\text{subgoals}(s, t, s') \rightarrow \bigvee_i \text{satisfies}(s, s', i)$
  5. $\text{satisfies}(s, s'', i) \rightarrow \bigvee \{\text{subgoal}(s, t) : t \text{ such } d(s, t) < d(s, s'')\}$
  6. $\text{satisfies}(s, s', i) \rightarrow \bigvee \{\text{subgoal}(s, t) : t \text{ such } d(s, t) \leq d(s, s')\}$
  7. $\text{satisfies}(s, s', i) \leftrightarrow "(s, s') \text{ satisfies rule } i"$
  8. Collection of rules is **terminating**

- **Paper to be presented at the conference (ICAPS 2022)**
About the Pool of Features $\mathcal{F}$ [B. et al., 2019]

- **Description logic grammar** allows generation of concepts and roles from domain predicates.
- Complexity of concept/role given by **size of its syntax tree**.
- Pool $\mathcal{F}$ obtained from concepts of complexity bounded by parameter.
- Denotation of concept $C$ in state $s$ is **subset of objects**.
- Each concept $C$ defines num and Bool features $n_C(s) = |C(s)|$; $p_C(s) = \top$ iff $|C(s)| > 0$.
- Grammar:
  - Primitive: $C_p$ given by unary predicates $p$ and unary “goal predicates” $p_G$.
  - Universal: $C_u$ contains all objects.
  - Nominals: $C_a = \{a\}$ for constants/parameter $a$.
  - Negation: $\neg C$ contains $C_u \setminus C$.
  - Intersection $C \cap C'$.
  - Quantified: $\exists R. C = \{x : \exists y[R(x, y) \land C(y)]\}$ and $\forall R. C = \{x : \forall y[R(x, y) \land C(y)]\}$.
  - Roles (for binary predicate $p$): $R_p$, $R_{p}^{-1}$, $R_{p}^{+}$, and $[R_{p}^{-1}]^+$.
- Additional **distance features** $\text{dist}(C_1, R, C_2)$ for concepts $C_1$ and $C_2$ and role $R$ that evaluates to $d$ in state $s$ iff minimum $R$-distance between object in $C_1$ to object in $C_2$ is $d$. 

B. Bonet, H. Geffner. Language-based Representation Learning for Acting and Planning. ICAPS/IJCAI 2022 Tutorial
Exploits correspondence between graph neural networks (GNNs) and two-variable logic $C_2$ to learn policy without requiring pool of $C_2$ features $\mathcal{F}$

- **Value function** $V$ learned that yields general policy $\pi_V$ greedy in $V$

- For generalization, based on GNN arch. for MaxCSP($\Gamma$) [Toenshoff et al., 2021]
  - **Input** given by the states $s$ extended with “goal predicates” $p_G$
  - **Output** $V(s)$ is non-linear aggregation of object embeddings
  - **Min Loss:** $|V^*(s) - V(s)|$ for supervised learning of optimal policies
  - **Min Loss:** $\max\{0, [1 + \min_{s' \in N(s)} V(s')] - V(s)\}$ unsupervised/non-optimal

- Nearly as good as policies based on explicit pool $\mathcal{F}$ of $C_2$ features

- Complexity of “latent features” not explicitly bounded

- Paper to be presented at the conference (ICAPS 2022)
GNN Architecture [Ståhlberg et al., 2022a,b]

Algorithm 1: GNN maps state \( s \) into scalar \( V(s) \)

**Input:** State \( s \): set of atoms true in \( s \), set of objects

**Output:** \( V(s) \)

1. \( f_0(o) \sim 0^{k/2} \mathcal{N}(0, 1)^{k/2} \) for each object \( o \in s \);
2. for \( i \in \{0, \ldots, L - 1\} \) do
   3. for each atom \( q := p(o_1, \ldots, o_m) \) true in \( s \) do
      4. // Msgs \( q \rightarrow o \) for each \( o = o_j \) in \( q \)
      5. \( m_{q,o} := [\text{MLP}_p(f_i(o_1), \ldots, f_i(o_m))]_j \);
   6. for each \( o \) in \( s \) do
      7. // Aggregate, update embeddings
      8. \( f_{i+1}(o) := \text{MLP}_U(f_i(o), \text{agg}(\{m_{q,o} \mid o \in q\})) \);
   9. // Final Readout
   10. \( V := \text{MLP}_2(\sum_{o \in s} \text{MLP}_1(f_L(o))) \)
Wrap Up: Representation Learning for Acting and Planning

- **Background 1**: Classical planning, planning width

- **Languages** for
  - representing general dynamics
  - representing general policies
  - representing general subgoal structures (sketches; ‘intrinsic rewards’)

- **Background 2**: Qualitative numerical planning problems (QNPs)

- **Learning** representations over these languages:
  - learning general dynamics
  - learning general policies
  - learning general subgoal structures

- **Wrap up; Challenges**
Wrap Up

- To learn representations that generalize due to structure, don’t play with low-level neural architecture; choose suitable (domain-independent) **target language** and learn representations over it:
  - generalization
  - transparency
  - powerful, meaningful bias
  - distinction between **what** and **how**

- Examples of learning language-based representations to **act** and **plan**:
  - general action **dynamics**
  - general **policies**
  - general **subgoal structures** (sketches)
Challenges: Language-based Representation Learning

- Scalability of combinatorial optimization approaches
- Use of deep learning (learning lifted dynamics, policies, sketches).
- Alternative target languages for learning (e.g., vs lifted STRIPS)
- Continuous domains, space, time
- Stochastic and non-deterministic domains
- States in the input: black-box, parsed images, images, videos
- Grounded vs. ungrounded representations
- Learning and reusing “skills”, hierarchies
- . . .

Plenty to do; if seriously interested, reach us
References


