Learning planning representations as a combinatorial optimization problem

Andrés Occhipinti Liberman

Universitat Pompeu Fabra (Barcelona, Spain)

Plan for the seminar

1. Planning & learning planning representations
   - A classical definition of planning
   - Action model learning (AML), vs. learning policies, sketches, etc.
   - Intuitive idea of AML

Plan for the seminar

1. Planning & learning planning representations
    - A classical definition of planning
    - Action model learning (AML), vs. learning policies, sketches, etc.
    - Intuitive idea of AML
2. Symbolic planning representations: STRIPS/PDDL
    - Motivation & examples

Plan for the seminar

1. Planning & learning planning representations
   - A classical definition of planning
   - Action model learning (AML), vs. learning policies, sketches, etc.
   - Intuitive idea of AML
2. Symbolic planning representations: STRIPS/PDDL
   - Motivation & examples
3. AML: brief overview of existing approaches

Plan for the seminar

1. Planning & learning planning representations
   - A classical definition of planning
   - Action model learning (AML), vs. learning policies, sketches, etc.
   - Intuitive idea of AML
2. Symbolic planning representations: STRIPS/PDDL
   - Motivation & examples
3. AML: brief overview of existing approaches
4. Some of our work
   - Learning simultaneously a state representation + action dynamics
   - AML as a combinatorial optimization problem

1. Planning & learning planning representations learning planning representations

Planning

- Planning problem: given an **initial state** s_0, and a **goal** g, is there some **action sequence** $(a_0, \ldots, a_t)$ that can take me from s_0 to a state s_g which satisfies g?

Planning

- Planning problem: given an **initial state** s_0, and a **goal** g, is there some **action sequence** $(a_0, \ldots, a_t)$ that can take me from s_0 to a state s_g which satisfies g?
- Examples: navigating from point A to point B, making dinner...

Planning

- Planning problem: given an **initial state** s_0, and a **goal** g, is there some **action sequence** $(a_0, \ldots, a_t)$ that can take me from s_0 to a state s_g which satisfies g?
- Examples: navigating from point A to point B, making dinner...
- **Model-based**: uses a **model** of how the world changes in response to actions.

Planning

- Planning problem: given an **initial state** s_0, and a **goal** g, is there some **action sequence** $(a_0, \ldots, a_t)$ that can take me from s_0 to a state s_g which satisfies g?
- Examples: navigating from point A to point B, making dinner...
- **Model-based**: uses a **model** of how the world changes in response to actions.
- Model specifies **action preconditions** (when is action a applicable?) and **action effects** (how does state s change if I take action a?)

Planning

- Planning problem: given an **initial state** s_0, and a **goal** g, is there some **action sequence** $(a_0, \ldots, a_t)$ that can take me from s_0 to a state s_g which satisfies g?
- Examples: navigating from point A to point B, making dinner...
- **Model-based**: uses a **model** of how the world changes in response to actions.
- Model specifies **action preconditions** (when is action a applicable?) and **action effects** (how does state s change if I take action a?)
- Planning is **simulation**: simulate state trajectories induced by action sequences before acting in the real world

High-level planning vs low-level control

- In this seminar: planning = **high-level** planning
- Example:
    - **Planning goal**: robot make eggs for breakfast.
    - **Possible plan**: go to fridge, grab eggs from fridge, put eggs on table, brush pan with olive oil, crack eggs...
- High-level plan will need to be mapped to the robot's low-level sensorimotor space (low-level sensing and control)

High-level planning vs low-level control

- In this seminar: planning = **high-level** planning
- Example:
  - **Planning goal**: robot make eggs for breakfast.
  - **Possible plan**: go to fridge, grab eggs from fridge, put eggs on table, brush pan with olive oil, crack eggs...
- High-level plan will need to be mapped to the robot's low-level sensorimotor space (low-level sensing and control)
- Assumptions high-level planning problems:
  - Finite, discrete state-space
  - Interaction in discrete time-steps
  - Actions are deterministic

Action model learning: intuitive idea

- Humans aren't born with internal representations of the world.
- Starting from a young age, they learn them via exploration.
- Children form hypotheses about how actions works, and engage in exploration to test and refine them [5, 4]. Then use them for planning.

Action model learning: intuitive idea

- Humans aren't born with internal representations of the world.
- Starting from a young age, they learn them via exploration.
- Children form hypotheses about how actions works, and engage in exploration to test and refine them [5, 4]. Then use them for planning.

A sophisticated action model learner :)

- Similarly, an artificial agent may have no internal model for planning.

- Similarly, an artificial agent may have no internal model for planning.
- **Action model learning**: problem of learning action representations from data, gathered by taking actions and observing their results.

- Similarly, an artificial agent may have no internal model for planning.
- **Action model learning**: problem of learning action representations from data, gathered by taking actions and observing their results.
- Action representations should be **general**: model of action generalizes to unseen scenarios

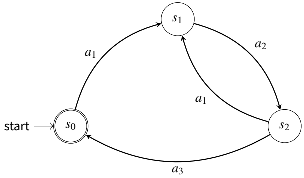
2. Symbolic planning representations: STRIPS/PDDL

Compact & general representations for planning

- A necessary input to any planning algorithm is a **description** of the problem to be solved: **states**, **actions**, **goals**

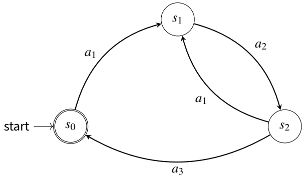
Compact & general representations for planning

- A necessary input to any planning algorithm is a **description** of the problem to be solved: **states**, **actions**, **goals**
- Simplest representation: state-space graph



Compact & general representations for planning

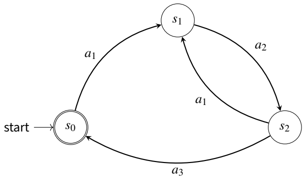
- A necessary input to any planning algorithm is a **description** of the problem to be solved: **states**, **actions**, **goals**
- Simplest representation: state-space graph



- In practice, explicit enumeration of possible states and state transitions impossible (typically, #states exponential in #objects)

Compact & general representations for planning

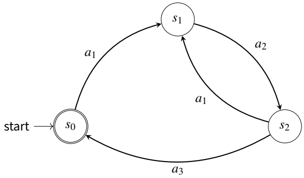
- A necessary input to any planning algorithm is a **description** of the problem to be solved: **states**, **actions**, **goals**
- Simplest representation: state-space graph



- In practice, explicit enumeration of possible states and state transitions impossible (typically, #states exponential in #objects)
- Representation not general (each graph tied to a specific instance)

Compact & general representations for planning

- A necessary input to any planning algorithm is a **description** of the problem to be solved: **states**, **actions**, **goals**
- Simplest representation: state-space graph



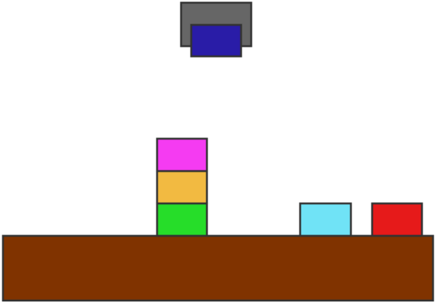
- In practice, explicit enumeration of possible states and state transitions impossible (typically, #states exponential in #objects)
- Representation not general (each graph tied to a specific instance)
- **Compact & general representation** is needed (avoid enumeration, makes it easy to compute transitions on-the-fly).

Compact & general representations

- Some “compact” action representations:
- Neural network $f^a_\theta(s) = s'$.
  - Action representation implicit in network parameters θ
  - Learned from data
  - Hard to interpret
- PDDL/STRIPS action schemas:
  - Explicit representation $a(x_1, \ldots, x_n) = (\text{pre}(a), \text{eff}(a))$: use declarative/logical language to define action preconditions and effects.
  - Typically hand-coded
  - Easy to interpret

PDDL by example: Blocksworld

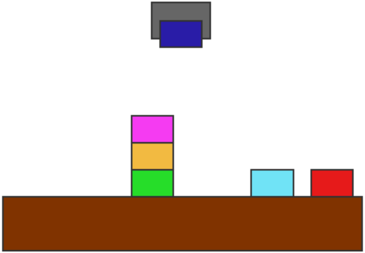
State representation:



```
objects and types
block(b-pink).
block(b-yellow).
robot(r).
table(t).
...
relations
clear(b-pink).
on(b-pink,b-yellow).
holding(b-blue).
...
```

PDDL by example: Blocksworld

Action representation:



```
stack block x on top of block y
(:action stack
      :parameters (?x ?y)
      :precondition (and
          holding(?x)
          clear(?y))
      :effect (and
          (not holding (?x))
          (not clear(?y))
          clear(?x)
          (handempty)
          on(?x,?y))
  )
```

Planning domains & problems

- Given action schema $a(x_1, \ldots, x_n)$ and objects $\overline{o} = (o_1, \ldots, o_n)$, the *instantiation* of the schema with $\overline{o}$ is the concrete action $a(o_1, \ldots, o_n)$.

Planning domains & problems

- Given action schema $a(x_1, \ldots, x_n)$ and objects $\overline{o} = (o_1, \ldots, o_n)$, the *instantiation* of the schema with $\overline{o}$ is the concrete action $a(o_1, \ldots, o_n)$.
- A **planning domain** $D = (L, A)$ is a pair where:
  - L is a set of *predicates* for describing states
  - A is a set of *action schemas*

Planning domains & problems

- Given action schema $a(x_1,\ldots,x_n)$ and objects $\overline{o} = (o_1,\ldots,o_n)$, the *instantiation* of the schema with $\overline{o}$ is the concrete action $a(o_1,\ldots,o_n)$.
- A **planning domain** $D = (L,A)$ is a pair where:
  - L is a set of *predicates* for describing states
  - A is a set of *action schemas*
- A **planning problem** $P = (D,I)$ is given by a planning domain $D = (L,A)$, and **instance information** $I = (O,s_0,g)$, where:
  - O is a set of *objects*
  - s_0 is an *initial state*
  - g is a *goal*
- Planning problem $P = (D,I)$ induces a labelled **planning graph** $G(P)$ where the nodes correspond to states and each edge (s,s') is labelled by action $\alpha = a(o_1,\ldots,o_n)$ if α is executable in s and leads to s'.

Pros and cons of PDDL (and of symbolic repr. in general)

Pros:

- General representation; size of action schemas constant across instances.
- Human-readable

Pros and cons of PDDL (and of symbolic repr. in general)

Pros:

- General representation; size of action schemas constant across instances.
- Human-readable

Cons:

- Typically hand-coded: error-prone, time-consuming task
- "Knowledge acquisition bottleneck": the excessive cost of human involvement in converting real-world problems into inputs for symbolic AI systems

Pros and cons of PDDL (and of symbolic repr. in general)

Pros:

- General representation; size of action schemas constant across instances.
- Human-readable

Cons:

- Typically hand-coded: error-prone, time-consuming task
- “Knowledge acquisition bottleneck”: the excessive cost of human involvement in converting real-world problems into inputs for symbolic AI systems

Way out? AML

3. AML: brief overview of existing approaches

AML: basic case

Let $D = (L, A)$ be an unknown domain, and let $P_1 = (P_1, \ldots, P_n)$ be problems over D.

Given:

1. The language L
2. Planning graphs $G(P_1), \ldots, G(P_1)$

AML: basic case

Let $D = (L, A)$ be an unknown domain, and let $P_1 = (P_1, \ldots, P_n)$ be problems over D.

Given:

1. The language L
2. Planning graphs $G(P_1), \ldots, G(P_1)$

Find: action schemas $\hat{A}$ defined with language L

AML: basic case

Let $D = (L, A)$ be an unknown domain, and let $P_1 = (P_1, \ldots, P_n)$ be problems over D.

Given:

1. The language L
2. Planning graphs $G(P_1), \ldots, G(P_1)$

Find: action schemas $\hat{A}$ defined with language L

s.t.: for *any* problem $P = ((L, A), (O, s_0, g))$ over D:

$G(P)$ and $G((L, \hat{A}), (O, s_0, g))$ are **isomorphic**

AML: beyond the basic case (1), incomplete graphs

Let $D = (L, A)$ be an unknown domain, and let $P_1 = (P_1, \ldots, P_n)$ be problems over D.

Given:

1. The language L
2. ~~Planning graphs~~ **execution traces over** $G(P_1), \ldots, G(P_1)$:

$$s_0, a_0(o_1, \ldots, o_n), s_1, a_1(o'_1, \ldots, o'_n), \ldots$$

Find: action schemas $\hat{A}$ defined with language L

s.t.: for *any* problem $P = ((L, A), (O, s_0, g))$ over D:

$G(P)$ and $G((L, \hat{A}), (O, s_0, g))$ are **isomorphic**

AML: beyond the basic case (2), partial state observability

Let $D = (L, A)$ be an unknown domain, and let $P_1 = (P_1, \ldots, P_n)$ be problems over D.

Given:

1. The language L
2. ~~Planning graphs~~ **partially observed execution traces**:

$$\textbf{obs}(s_0), a(o_1, \ldots, o_n), \textbf{obs}(s_1), a_1(o'_1, \ldots, o'_n), \ldots$$

Find: action schemas $\hat{A}$ defined with language L

s.t.: for *any* problem $P = ((L, A), (O, s_0, g))$ over D:

$G(P)$ and $G((L, \hat{A}), (O, s_0, g))$ are **isomorphic**

AML: beyond the basic case (3), no state observability

Let $D = (L, A)$ be an unknown domain, and let $P_1 = (P_1, \ldots, P_n)$ be problems over D.

Given:

1. The language L
2. ~~Planning graphs~~ **possible action sequences**:

$$\cancel{\textbf{obs}(s_0)}, a_0(o_1, \ldots, o_n), \cancel{\textbf{obs}(s_1)}, a_1(o'_1, \ldots, o'_n), \ldots$$

Find: action schemas $\hat{A}$ defined with language L

s.t.: for *any* problem $P = ((L, A), (O, s_0, g))$ over D:

$$G(P) \text{ and } G((L, \hat{A}), (O, s_0, g)) \text{ are } \textbf{isomorphic}$$

AML: beyond the basic case (4), no state observability, no action parameters, no language

Let $D = (L, A)$ be an unknown domain, and let $P_1 = (P_1, \ldots, P_n)$ be problems over D.

Given:

1. ~~The language L~~

AML: beyond the basic case (4), no state observability, no action parameters, no language

Let $D = (L, A)$ be an unknown domain, and let $P_1 = (P_1, \ldots, P_n)$ be problems over D.

Given:

1. ~~The language L~~
2. ~~Planning graphs~~ **execution traces** over $G(P_1), \ldots, G(P_n)$:

$$\mathbf{id}(s_0),\ a_0\cancel{(o_1, \ldots, o_n)},$$

AML: beyond the basic case (4), no state observability, no action parameters, no language

Let $D = (L, A)$ be an unknown domain, and let $P_1 = (P_1, \ldots, P_n)$ be problems over D.

Given:

1. ~~The language L~~
2. ~~Planning graphs~~ **execution traces** over $G(P_1), \ldots, G(P_n)$:

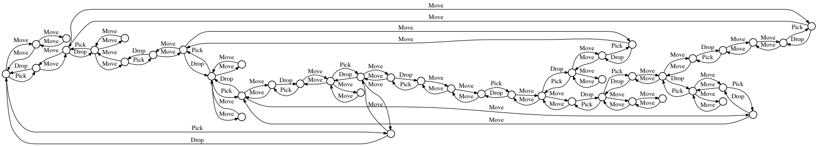
$$\textbf{id}(s_0),\ a_0\cancel{(o_1, \ldots, o_n)},\textbf{id}(s_1),a_1\cancel{(o'_1, \ldots, o'_n)},\ldots$$

Find: **language** $\hat{L}$ and action schemas $\hat{A}$

s.t.: for *any* problem $P = ((L, A), (O, s_0, g))$ over D:

$$G(P) \text{ and } G((\hat{L}, \hat{A}), (O, s_0, g)) \text{ are } \textbf{isomorphic}$$

Input: State graph G of agent in 1×3 grid, moving/picking/dropping 2 pkgs



Output: Simplest domain $D = (L, A)$ that **generates** G:

```
Move(?to,?from):
  Pre: neq(?to,?from), p5(?to,?from)
  Pre:  p2(?from), -p2(?to)
  Eff: -p2(?from),  p2(?to)

Pick(?p,?x):
  Pre:  p2(?x),   p1, -p3(?p),  p4(?p,?x)
  Eff:           -p1,  p3(?p), -p4(?p,?x)

Drop(?p,?x):
  Pre:  p2(?x), -p1,   p3(?p), -p4(?p,?x)
  Eff:            p1, -p3(?p),  p4(?p,?x)
```

Interpretation of learned predicates:

1. p_1: gripper empty
2. $p_2(x)$: agent at cell x,
3. $p_3(p)$: agent holds pkg p,
4. $p_4(p,x)$: pkg p in cell x
5. $p_5(x,y)$: cell x adj to y

- Domain D correct for **any** grid, **any** # of packages. Structure of nodes uncovered.

Key features of this work[1]

- Learn **simultaneously** *state representation language* L and *domain dynamics* A.

[1] [1, 3]

Key features of this work[1]

- Learn **simultaneously** *state representation language* L and *domain dynamics* A.
- From **topology of graph alone** (can also accommodate partial graphs)

[1] [1, 3]

Key features of this work[1]

- Learn **simultaneously** *state representation language* L and *domain dynamics* A.
- From **topology of graph alone** (can also accommodate partial graphs)
- Casted as a **combinatorial optimization problem**:
  - **Given**: graph G
  - **Find**: domain $D = (L, A)$
  - **S.t.**: D induces graph isomorphic to G
  - **Minimize**:
    1. Sum of action schema parameters (prefer simpler actions)
    2. Sum of predicates arities (prefer simpler state repr.)
    3. Number of action effects
    4. Number of preconditions

[1] [1, 3]

Key features of this work[1]

- Learn **simultaneously** *state representation language* L and *domain dynamics* A.
- From **topology of graph alone** (can also accommodate partial graphs)
- Casted as a **combinatorial optimization problem**:
  - **Given**: graph G
  - **Find**: domain $D = (L, A)$
  - **S.t.**: D induces graph isomorphic to G
  - **Minimize**:
    1. Sum of action schema parameters (prefer simpler actions)
    2. Sum of predicates arities (prefer simpler state repr.)
    3. Number of action effects
    4. Number of preconditions
- Solved using Answer Set Programming (CLINGO solver)
- Learns solutions that generalize for several standard planning domains

[1] [1, 3]

Limitations:

- Input minimal and search relatively unconstrained; limits scalability to more complex domains
- Learned predicates are **ungrounded**; symbol grounding needs to be done manually

```
Move(?to, ?from):
  Pre: neq(?to, ?from),  p5(?to, ?from)
  Pre:  p2(?from), -p2(?to)
  Eff: -p2(?from),  p2(?to)

Pick(?p, ?x):
  Pre:  p2(?x),  p1, -p3(?p),  p4(?p, ?x)
  Eff:          -p1,  p3(?p), -p4(?p, ?x)

Drop(?p, ?x):
  Pre:  p2(?x), -p1,  p3(?p), -p4(?p, ?x)
  Eff:           p1, -p3(?p),  p4(?p, ?x)
```

Interpretation of learned predicates:

1. p_1: gripper empty
2. $p_2(x)$: agent at cell x,
3. $p_3(p)$: agent holds pkg p,
4. $p_4(p,x)$: pkg p in cell x
5. $p_5(x,y)$: cell x adj to y

Learning representations that are grounded [2]

Overcoming limitations:

- Augment input with information about states expressed in a simple **domain-independent** language

Learning representations that are grounded [2]

Overcoming limitations:

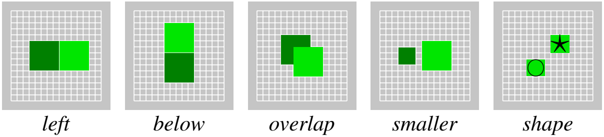
- Augment input with information about states expressed in a simple **domain-independent** language
- **O2D**: simple language that captures basic **spatial relations** amongst objects

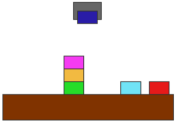
Learning representations that are grounded [2]

Overcoming limitations:

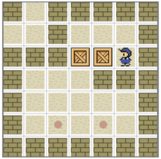
- Augment input with information about states expressed in a simple **domain-independent** language
- **O2D**: simple language that captures basic **spatial relations** amongst objects
- Learned predicates correspond to logical conditions over O2D language: i.e., learned symbols are **grounded** over spatial information

Visual language for states: O2D language





```
% objects and types
robot(r). table(t).
block(b0). block(b1).
...
% relations
overlap(b0,r).
below(t,b1).
smaller(b1,r).
...
% shapes
shape(r,rectangle).
shape(b0,rectangle).
...
```



```
% object and types
sokoban(s).
crate(c1). crate(c2).
cell(c1_1).
...
% relations
overlap(s,c3_6).
overlap(c2, c3_5).
below(c3_6,c_2_6).
...
% shapes
shape(c1,rectangle).
...
```

States represented as **objects**, their **types**, and 5 qualitative, fixed **spatial relations**.
Grounded predicate: derived from O2D language. Example:
$clear(\texttt{pink\_block}) := \neg\exists x(below(\texttt{pink\_block}, x) \wedge block(x))$

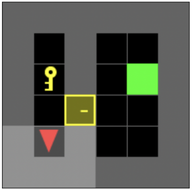
O2D & grounded predicates

1. **Pool** of grounded predicates obtained from primitive O2D predicates using a (description logic) grammar.
2. The actual **description logic grammar** used is given by:

$$
\begin{aligned}
C &\leftarrow U \mid \top \mid \bot \mid \exists R.C \mid C \sqcap C' \mid C \sqsubseteq C' \\
R &\leftarrow R \mid R^{-1} \mid R \circ R'
\end{aligned}
$$

   - Where U and R are primitive O2D unary and binary predicates, respectively.
   - Nullary predicates $C \sqsubseteq C'$ are true iff the denotation of C is a subset of the denotation of C'.
3. Finite pool generated by constructing all predicates up to a given grammar complexity, pruning syntactic variants

Experimental results: some models learned



[Grid] Pickup(p,k):

pre: $\mathit{armempty}$, $\mathit{at}(\mathtt{R},p)$, $\mathit{at}(p,k)$

eff: $\neg\mathit{armempty}$, $\neg\mathit{somecell}(k)$, $\neg\mathit{at}(p,k)$, $\neg\mathit{at}(k,p)$

groundings:

$\mathit{armempty} :=$ SUBSET[key,ER[overlap,Top]]

$\mathit{somecell} :=$ INTER[key,ER[overlap,Top]]

$\mathit{at} := \mathit{overlap}$

[Sokoban] Pushdown(x,y,z,c):

static: $\mathit{below}(z,y)$, $\mathit{below}(y,x)$

pre: $\mathit{at}(\mathtt{Sok},x)$, $\mathit{at}(c,y)$, $\neg\mathit{nempty}(z)$

eff: $\neg\mathit{nempty}(x)$, $\mathit{nempty}(z)$, $\mathit{at}(\mathtt{Sok},y)$, $\mathit{at}(y,\mathtt{Sok})$, $\neg\mathit{at}(\mathtt{Sok},x)$
$\neg\mathit{at}(x,\mathtt{Sok})$, $\neg\mathit{at}(y,c)$, $\neg\mathit{at}(c,y)$, $\mathit{at}(c,z)$, $\mathit{at}(z,c)$

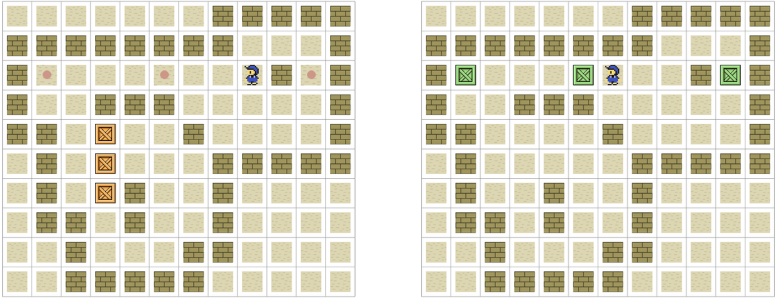
Experimental results for some domains: input data

						Predicate pool $\mathscr{P}$		
Domain (#inst.)	#obj.	#const.	$\|A\|$	$\|S\|$	#edges	$\|\mathscr{P}\|$	compl	time
Blocksworld (5)	5	2	4	1,020	2,414	79	4	9.13
Towers of Hanoi (5)	8	1	4	363	1,074	14	2	2.03
Sliding Tile (7)	11	1	4	742	1,716	16	2	0.96
IPC Grid (19)	11	1	10	9,368	23,530	164	4	316.64
Sokoban1 (95)	22	3	8	1,936	5,042	18	2	8.54
Sokoban2 (24)	27	3	8	12,056	36,482	18	2	160.48

Experimental results for some domains: learning stats

				Learning time in seconds			
Domain	#iter	#inst.	#states	solve	ground	verif.	total
Blocksworld	7	3	16	0.29	23.37	29.42	53.70
Towers of Hanoi	6	4	27	1.56	12.67	0.59	15.06
Sliding Tile	6	5	10	0.11	2.89	1.20	4.43
IPC Grid	27	12	127	693.44	3,536.23	2,404.87	6,653,03
Sokoban1	10	9	13	16.18	285.56	9.18	311.79
Sokoban2	11	8	56	7,250.67	5,314.35	165.19	12,740.43

Experimental results: planning with the learned models



Initial state Goal state

Figure: Sokoban instance. Optimal plans of length 156 are found using the original "hidden" domain and the learned grounded domain.

Wrap-up

- AML can be casted as a **combinatorial optimization problem**: find simplest planning domain given data

Wrap-up

- AML can be casted as a **combinatorial optimization problem**: find simplest planning domain given data
- State representation language and action dynamics can be learned **simultaneously**. **Meaning of learned symbols can be ground** in simple, **domain-independent language** capturing visual relationships

Wrap-up

- AML can be casted as a **combinatorial optimization problem**: find simplest planning domain given data
- State representation language and action dynamics can be learned **simultaneously**. **Meaning of learned symbols can be ground** in simple, **domain-independent language** capturing visual relationships
- Learned representations are **interpretable** and **general**

Wrap-up

- AML can be casted as a **combinatorial optimization problem**: find simplest planning domain given data
- State representation language and action dynamics can be learned **simultaneously**. **Meaning of learned symbols can be ground** in simple, **domain-independent language** capturing visual relationships
- Learned representations are **interpretable** and **general**
- Future work: obtain visual description directly from images; learn action representations with deep learning.

Thank you!

[1] B. Bonet and H. Geffner. "Learning first-order symbolic representations for planning from the structure of the state space". In: *Proc. ECAI*. 2020.

[2] Andres Occhipinti Liberman, Blai Bonet, and Hector Geffner. "Learning First-Order Symbolic Planning Representations That Are Grounded". In: *3rd ICAPS workshop on Bridging the Gap Between AI Planning and Reinforcement Learning* (2022).

[3] I. D. Rodriguez et al. "Learning First-Order Representations for Planning from Black-Box States: New Results". In: *KR*. arXiv preprint arXiv:2105.10830. 2021.

[4] Laura Schulz and Elizabeth Bonawitz. "Serious Fun: Preschoolers Engage in More Exploratory Play When Evidence Is Confounded". In: *Developmental psychology* 43 (Aug. 2007), pp. 1045–50.

[5] Aimee E. Stahl and Lisa Feigenson. "Observing the unexpected enhances infants' learning and exploration". In: *Science* 348.6230 (2015), pp. 91–94. DOI: 10.1126/science.aaa3799.