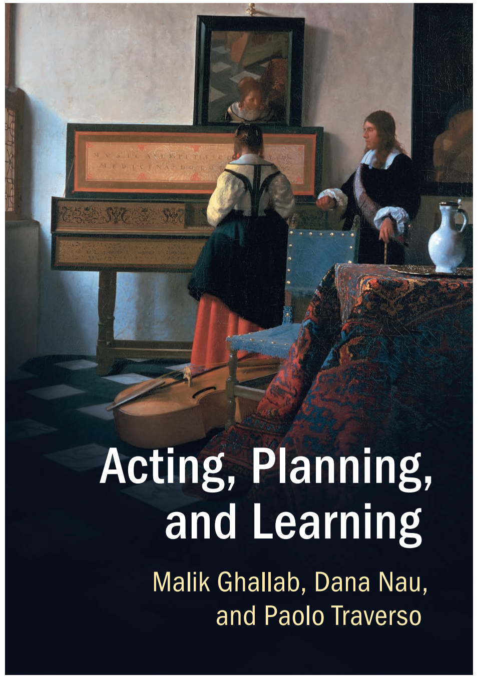
Last update: 11:09 PM, March 7, 2025



Chapter 4
Learning with Deterministic Models
(brief summary)

Dana S. Nau
University of Maryland

Outline and Assumptions

I'll briefly summarize the following sections:

4.1 Learning Heuristics

4.2. Learning Action Specifications

4.2.2 Online Action Learning

- Classical planning assumptions:
  - Finite, static world, just one actor
  - No concurrent actions, no explicit time
  - Determinism, no uncertainty
  - Sequence of states and actions $\langle s_0, a_1, s_1, a_2, s_2, \ldots\rangle$

4.1. Learning Heuristics

LRTA*(Σ, s_0, S_g, h_0)
- $s \leftarrow s_0$; $\pi \leftarrow \langle\rangle$ *// initialize current state and plan*
- $h(s) \leftarrow h_0(s)$ for every $s \in S$ *// initialize the heuristic*
- **while** $s \notin S_g$ **do**
  - **for each** a in *Applicable*(s) **do**
    - $Q(s,a) \leftarrow \text{cost}(s,a) + h(\gamma(s,a))$
  - $h(s) \leftarrow \min_a Q(s,a)$
  - $a \leftarrow \text{argmin}_a\, Q(s,a)$
  - $\pi \leftarrow \pi \cdot a$
  - $s \leftarrow \gamma(s,a)$

- Assume the domain is *safely explorable*
  - At every state s there is a path to S_g
- In each state s, $Q(s,a)$ is the estimated cost of getting to S_g if we start with action a
- LRTA* finds a path from s_0 to S_g
  - Updates $h(s)$ for every s along the path
- Call it again, it finds another path, updates h along that path
- After enough calls, it will find an optimal path
  - Along that path, $h(s) = h^*(s)$
  - But not along other paths

- Other algorithms to find a heuristic h that is close to h^* at every state
  - Based on *value iteration*

4.2. Learning Action Specifications

- *Action trace*: a triple $(s, \text{head}(a), s')$ such that $\gamma(s,a) = s'$
  - ({loc(r1) = d3, cargo(r1) = nil, loc(c1) = d1},
    move(r1,d3,d1),
    {loc(r1) = d1, cargo(r1) = nil, loc(c1) = d1})
- Let Σ be a state-transition system,
  T_Σ = {all possible action traces for Σ}
- Action model learning problem:
  - Given a set of action traces $T \subseteq T_\Sigma$
    - examples of what the actions do
  - Create an *action model* $\mathcal{AM} = (Xsym, Asym, Schema)$
    - *Xsym* and *Asym* are sets of state-variable names and action names (without arguments)
    - *Schema* is a function that maps each action name $\alpha \in Asym$ into an action schema

- $\mathcal{AM}$ defines a state-transition system Σ'
- Let $T_{\Sigma'}$ = {all possible action traces for Σ'}
- $\mathcal{AM}$ is *sound* if $T_{\Sigma'} \subseteq T_\Sigma$
  - $T_{\Sigma'}$ doesn't include any incorrect action traces
- $\mathcal{AM}$ is *complete* if $T_\Sigma \subseteq T_{\Sigma'}$
  - $T_{\Sigma'}$ includes every correct action trace
- Three ways to get the traces in T:
  - offline
  - online
  - from "informative states"

Three Kinds of Learning

- *Offline learning:*
  - Given a fixed set of action traces $T \subseteq T_\Sigma$
  - Advantage: quick access to examples
  - Disadvantage: The examples might not show you conclusively what each action does
- *Online learning:* learner generates T by acting in Σ
  - Observe current state, choose action a, send it to the execution platform, see what state it produces, …
  - Advantage: If you're unsure what a does in the current state, you can try it and see
  - Disadvantage: Each action execution takes time. Collecting enough observations may take lots of time.
- *Learning from informative states:*
  - Suppose the learner has an oracle for Σ
    - e.g., a quick simulator
  - Given (s,a), it either returns $\gamma(s,a)$ or says that a isn't applicable
  - Advantage: Learner can try a in many different states until it's sure what a does
  - Disadvantage: Not feasible unless you have a simulator that's both fast and accurate
- In all three cases, observations give information about *actions*
  - To get action schemas, must generalize
  - Use techniques based on lifting

Lifted Preconditions and Effects

- Recall what an action schema looks like
  - head: $name(z_1, \ldots, z_n)$
  - pre: atoms in which every object variable is one of $z_1, \ldots, z_n$
  - eff: assignments in which every object variable is one of $z_1, \ldots, z_n$

- Assume the action schemas contain no constants
  - Then the action schemas are fully lifted
  - In pre and eff,
    - every state variable contains only $z_1, \ldots, z_n$ as parameters
    - every state variable's value is one of $z_1, \ldots, z_n$

- Suppose we're given an action trace

  ({loc(r1) = d3}, move(r1,d3,d1), {loc(r1) = d1})

- We want to figure out the preconditions and effects of move(z_1, z_2, z_3)
- For pre and eff, consider atoms such as loc(z_1) = z_2, loc(z_1) = z_3

This restriction can exclude useful action schemas, e.g.

load(r, c, l)
pre: cargo(r)=nil, loc(c)=l, loc(r)=l
eff: cargo(r)←c, loc(c)←r

Offline Learning (Basic Idea)

- Given $T \subseteq T_\Sigma$ – a set of triples $(s, \alpha(c_1,\ldots,c_k), s')$
- For each α, begin with action schema (*head,pre,eff*)
  - $head = \alpha(z_1,\ldots,z_k)$
  - pre = {atoms of the form $x(z'_1,\ldots, z'_n) = z'_{n+1}$ where $x \in Xsym$ and each z'_i is one of $z_1,\ldots,z_k$}
  - $eff = \emptyset$
- **for** every $(s, \alpha(c_1,\ldots,c_k), s') \in T$ **do**
  - **for** every atom $x(z'_1,\ldots, z'_n) = z'_{n+1}$ in pre$(\alpha(z_1,\ldots,z_k))$ that doesn't have a ground instance in s **do**
    - remove it from pre$(\alpha(z_1,\ldots,z_k))$
  - **for** every atom $x(z_1,\ldots, z_n) = z_{n+1}$ that has a ground instance in $s' \setminus s$ **do**
    - add $x(z_1,\ldots, z_n) \leftarrow z_{n+1}$ to eff$(\alpha(z_1,\ldots,z_k))$

- Need to fill in some additional details
- Can prove it produces a sound action model
  - $T_{\Sigma'} \subseteq T_\Sigma$
- Completeness depends on whether T contains enough action traces
- Can enhance the algorithm by allowing action traces of the form
  - $(s, \alpha(c_1,\ldots,c_k), \textit{inapplicable})$

- As given, the algorithm is very inefficient
  - each action schema starts with huge number of preconditions, must remove most of them
- Paolo will revise it to make it more efficient

4.2.2 Online Learning

- Learning actions by queries
  - Access to an oracle, e.g., a quick simulator
  - *Informative state*
    - A state s such that $\gamma(s,a)$ will provide needed information about a
  - Generate queries about such states
  - Can write an algorithm that is both correct and complete
- Online action learning
  - Observe current state, choose action a, send it to the execution platform, see what state it produces, …
  - Try to generate plans that will lead to informative states