An Introduction to Planning in Artificial Intelligence

Kostas N. Oikonomou

Email: ko56@winlab.rutgers.edu

December 2023

Outline

I. Overview of AI

II. Intelligent Agents

III. Planning in AI

IV. Hierarchical Planning

V. Expressiveness & Complexity

VI. Planning & Learning

VII. Learning theory

I Overview of AI

Artificial Intelligence

The goal of the field of Artificial Intelligence (AI) can be summarized as

Create an intelligent entity

- When is an entity intelligent?

Acting vs. Thinking and Rational vs. Human

- Acting vs. thinking: judge by 'outward' behavior, or by its 'inner' laws?
- Rational vs. human: should follow principles of rationality, or try to imitate humans[1]?

The Turing test (Alan Turing, 1950): acting humanly.

> A system is intelligent if a human, who communicates with it by exchanging written messages, cannot tell if it is a human being or not.

1. Not always irrational!

Topics in AI

- STRUCTURE OF INTELLIGENT AGENTS
- PROBLEM SOLVING
  - Searching by heuristic methods
  - Optimization-related methods
  - Constraint programming
  - Searching with partial observations
  - Online search
  - Adversarial search, games
- KNOWLEDGE, REASONING, PLANNING
  - Logical agents, first-order logic
  - Planning and acting
  - Representation of knowledge

- Uncertain Knowledge and Reasoning
  - Quantifying uncertainty
  - Probabilistic reasoning, Bayesian networks, causal networks
  - Making decisions
  - Multiple agents
- Machine Learning
  - Learning from examples[2]
  - Knowledge in learning
  - Learning probabilistic models
  - Deep learning
  - Reinforcement learning

2. Supervised, unsupervised, decision trees, neural networks, ...

- Communicating, Perceiving, and Acting
  - Natural language processing (e.g. LLMs)
  - Perception
  - Robotics
- Philosophical Issues, Ethical Issues ...

Reference:

Artificial Intelligence: A Modern Approach

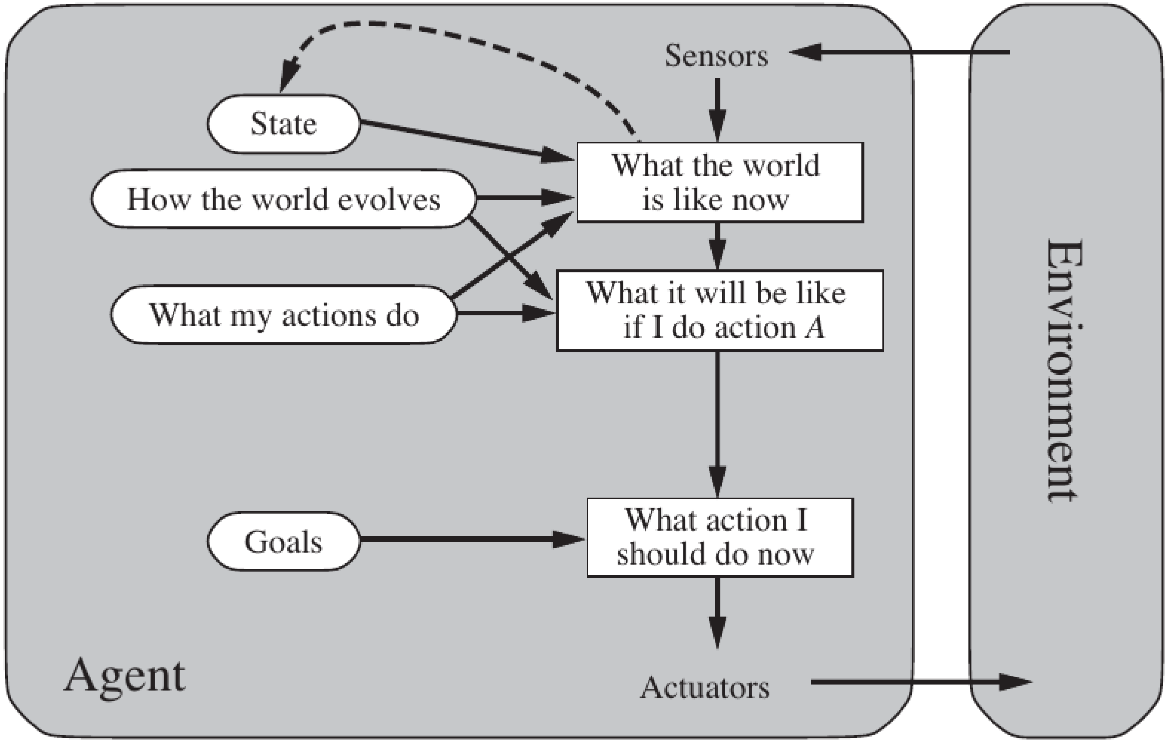
Stuart Russell, Peter Norvig[3]

3d ed. 2010, 4th ed. 2021

3. UC Berkeley and Google Research

II Intelligent Agents

An intelligent agent



A model-based, goal-based agent
(for simplicity, no learning)

Environment (world):

- fully/partially observable:
  w.r.t. all relevant aspects
- deterministic/uncertain:
  its next state is uniquely determined by its current state + agent's action, or not
- known/unknown:
  how much the agent knows about "how the world evolves" (independently, and due to its actions)

Agent's state:
how agent keeps track of the part of the world it can't see.

world fully-observable, deterministic, and known
$\Rightarrow$
precepts (from sensors) don't provide any add'l information

Driving a taxi in a downtown area. Environment is

- Fully/**partially** observable: e.g. cars in blind spots.
- Single/**multi** agent: partially **cooperative**, partially **competitive**.
- Deterministic/**uncertain**[4]: other traffic, lights, roadwork, police, pedestrians ...
- Episodic/**sequential**: present decision may affect future ones.
- Static/**dynamic**: environment changes while agent is ‘thinking’.
- Discrete/**continuous**: time is continuous.
- Known/unknown: mostly known.

Even harder case if the driver is a tourist in a new country.

---

4. ‘uncertain’ can be further classified as non-deterministic or stochastic.

III Planning in AI

- AI planning has a particular world view:

  There is an agent, who is acting in the world.

- This is captured by a planning domain:
  - a model of the 'world': objects, their properties, relationships, ...
  - the actions available to the agent,
  - the goal that the agent wants to achieve.
- The planner (planning algorithm) finds a sequence of actions for the agent, the plan, that achieve the goal.

This is classical planning.

And in its simplest form: single-agent, discrete, non-temporal, deterministic, no interaction with acting (offline planning), ...

Detour: deep learning and big data

When people talk about 'AI' these days, they almost always mean some kind of machine learning or deep neural networks.

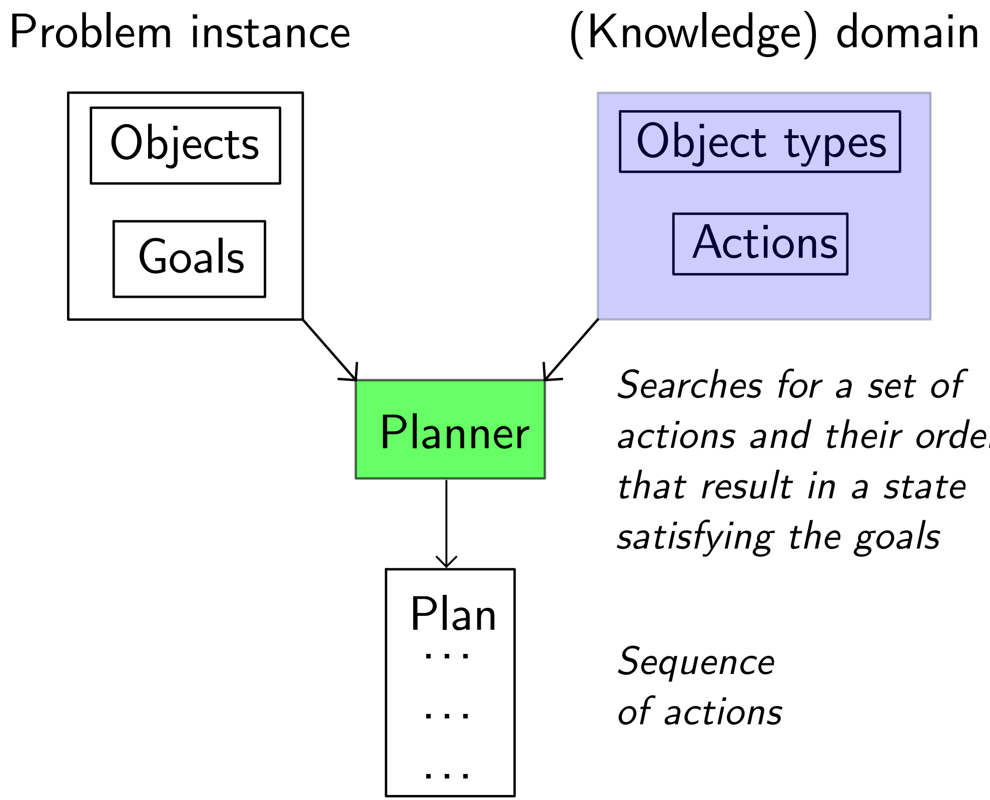
In the AI planning that we are talking about here

- there are no neural networks,
- there is no big data, and there is no training of anything on big data.

Why not? In cases where AI planning is applied,

- We typically have a lot of accumulated domain knowledge.
- We can reason about the domain.
- There isn't a large number of 'examples'.

Problem, domain, planner



HUMAN EXPERTS

There are many instances of a given domain

SOFTWARE TOOL

Searches for a set of actions and their ordering that result in a state satisfying the goals

Sequence of actions

Here environment is assumed fully observable, known, deterministic, static.

Example: the depo planning problem

DOMAIN: There are distributors that have depots. Pallets are there, loaded with crates. We have trucks to drive to/from depots, and can use hoists there to load and unload the crates. Trucks have load limits, and consume fuel.

GOAL: the crates must be taken from where they are and put onto specified pallets at specified depots.
(Optionally: with least fuel cost.)

POSSIBLE ACTIONS:

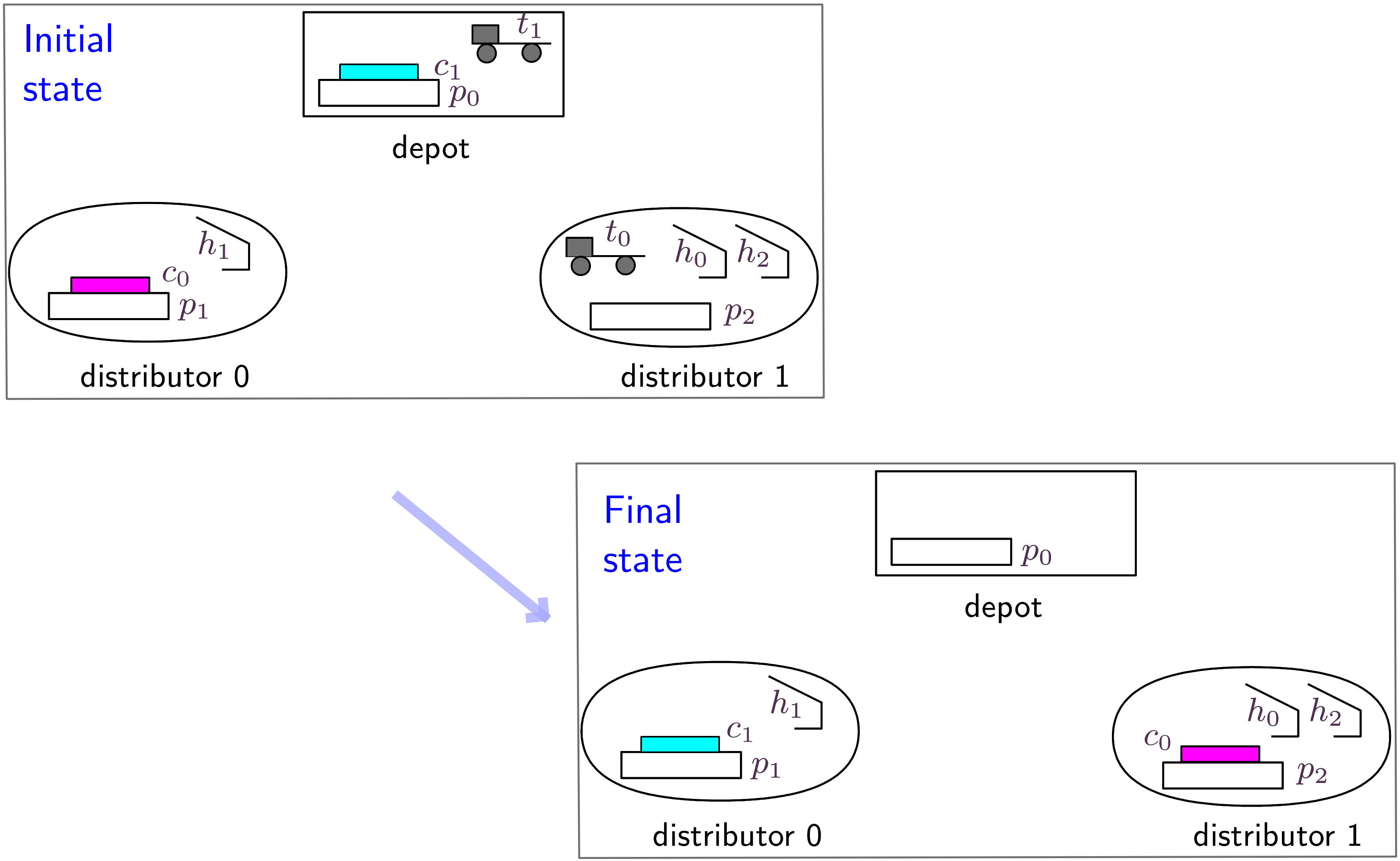
→ Drive a truck from one place to another,

→ Lift a crate with a free hoist,

→ Drop a crate onto a pallet with a hoist,

→ Load a crate on a truck with a hoist,

→ Unload a crate from a truck with a hoist.


Initial
state
t_1
c_1
p_0
depot
h_1
c_0
p_1
distributor 0
t_0
h_0
h_2
p_2
distributor 1
Final
state
p_0
depot
h_1
c_1
p_1
distributor 0
h_0
h_2
c_0
p_2
distributor 1

A 12-step 'satisficing' plan:

```
0.0: (Lift hoist0 crate1 pallet0 depot0 )
1.0: (Load hoist0 crate1 truck1 depot0 )
2.0: (Drive truck0 distributor1 distributor0 )
3.0: (Lift hoist1 crate0 pallet1 distributor0 )
4.0: (Load hoist1 crate0 truck0 distributor0 )
5.0: (Drive truck0 distributor0 depot0 )
6.0: (Drive truck1 depot0 distributor0 )
7.0: (Unload hoist1 crate1 truck1 distributor0 )
8.0: (Drive truck1 distributor0 depot0 )
9.0: (Drive truck0 depot0 distributor1 )
10.0: (Unload hoist2 crate0 truck0 distributor1 )
11.0: (Drop hoist2 crate0 pallet2 distributor1 )
12.0: (Drop hoist1 crate1 pallet1 distributor0 )
```

Optimal plan: minimize fuel cost

```
0.0: (Lift hoist0 crate1 pallet0 depot0 )
1.0: (Load hoist0 crate1 truck1 depot0 )
2.0: (Lift hoist1 crate0 pallet1 distributor0 )
3.0: (Drive truck1 depot0 distributor0 )
4.0: (Load hoist1 crate0 truck1 distributor0 )
5.0: (Unload hoist1 crate1 truck1 distributor0 )
6.0: (Drop hoist1 crate1 pallet1 distributor0 )
7.0: (Drive truck1 distributor0 distributor1 )
8.0: (Unload hoist2 crate0 truck1 distributor1 )
9.0: (Drop hoist2 crate0 pallet2 distributor1 )
```

9 steps and uses only one truck.

- PDDL: Planning Domain Definition Language
- Objects and their properties are represented by terms, predicates and atoms:

```
(:types place locatable - object
        depot distributor - place
        truck hoist surface - locatable
        pallet crate - surface)
(:predicates (located ?x - locatable ?p - place)
             (on ?c - crate ?s - surface)
             (in ?c - crate ?t - truck)
             (lifting ?h - hoist ?c - crate)
             (available ?c - hoist)
             (clear ?s - surface))
(:functions
        (load_limit ?t - truck)
        (current_load ?t - truck)
        (weight ?c - crate)
        (fuel-cost))
```

- Functions take terms as arguments and return other terms.

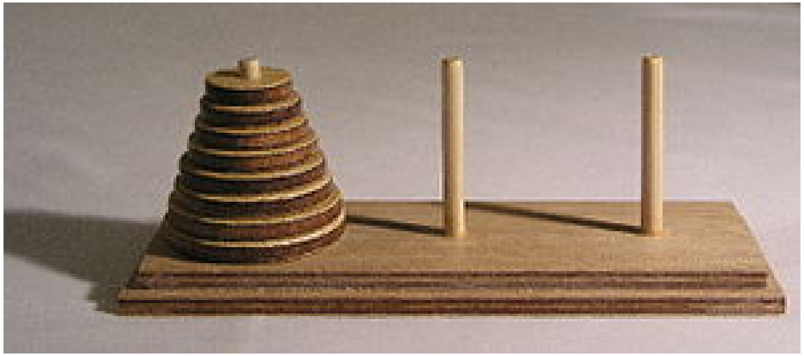
```
(:action drive
:parameters (?t - truck ?p1 ?p2 - place)
:precondition (and (located ?t ?p1))                ; 1st-order logic formula
:effect (and (not (located ?t ?p1)) (located ?t ?p2) ; 1st-order logic formula
              (increase (fuel-cost) 10)))

(:action lift
:parameters (?h - hoist ?c - crate ?s - surface ?p - place)
:precondition (and (located ?h ?p) (available ?h) (located ?c ?p) (on ?c ?s)
                    (clear ?c))
:effect (and (not (located ?c ?p)) (lifting ?h ?c) (not (clear ?c))
          (not (available ?h)) (clear ?s) (not (on ?c ?s))
          (increase (fuel-cost) 1)))

(:action drop
:parameters (?h - hoist ?c - crate ?s - surface ?p - place)
:precondition (and (located ?h ?p) (located ?s ?p) (clear ?s) (lifting ?h
 ?c))
:effect (and (available ?h) (not (lifting ?h ?c)) (located ?c ?p)
          (not (clear ?s)) (clear ?c) (on ?c ?s)))
```

There is no “main program”!

- A computational problem: integers x and u, $x \leqslant u$.
  - Initial state: $x = 2$, $u = 18$.
  - Actions: A_1: if $x \leqslant u - 3,$ add 3 to x, A_2: if $x \leqslant u - 5$, add 5 to x.
  - Find a sequence of A_1s and A_2s that accomplish $x = u$.
- A procedural problem: The towers of Hanoi.



"It is said that in the temple of Brahma in India, there is a tower with 64 golden disks. Monks have been moving them one-by-one, for a long time, one per second, ... and they are still working."

  - A_1: move a disk to an empty peg, A_2: put on the top of another stack.
  - Constraint: *no disk can ever be placed on a smaller disk!*
  - Task: move pile from first peg to last peg.

When/why do you consider AI planning?

1. When the viewpoint of an agent acting in the world is natural for the problem.
2. When there is no well-known mathematical formulation for the problem, or it is not easy to derive one:

   Problem is not well-studied/understood, has many complicated conditions/constraints, its formulation is not precise, is subject to change, there are multiple objectives, ...
3. If the problem
   - is well-studied, e.g. finding shortest paths in a graph, don't look at AI planning, there are much better ways to solve it.
   - is suited to deep learning/big data $\Rightarrow$ not suited to AI planning.
4. Dividing line is not always sharp: some problems can be approached in more than one way.
5. Some AI planners (formalisms + tools) allow including optimization algorithms or neural networks as black boxes.

Extensions to classical planning

Current research on classical planning focuses on domain-independent search heuristics.

There are also many extensions:

- Integrated planning and acting.
- Temporal planning: actions have durations, can overlap, plans are schedules.
- Planning with limited resources.
- Planning under uncertainty (e.g. Markov decision processes, MDPs).
- Planning with hybrid (discrete + continuous) systems.
- Multiple agents: cooperative, or competitive.

Hierarchical planning: methods on top of actions, tasks instead of goals.

Planning languages/formalisms

AI planning is an active research area:

Conferences

- ICAPS: International Conference on Automated Planning and Scheduling
- IPC: International Planning Competition
- IJCAI: International Joint Conference on Artificial Intelligence

Planning languages, and open-source planners

- PDDL, the Planning Domain Definition Language
  - Under development for more than 30 years.
  - There are specifications for versions 1.x and 2.x, and proposals for 3.x. Almost all address classical planning + extensions.
  - Widely used by planner developers, but still not an official standard.
- Other formalisms, some with extensions: planning tools written in Python, Java, Lisp.

Desirable properties of a planner (planning algorithm):

- Handle any domain, set of actions, and goals: domain independence.
- If a plan exists, it will be found: completeness[5].
- If a plan is found, it is correct: soundness.
- If there is a measure of optimality, and an optimal solution exists, it will be found: admissibility[6].

Not the kind of properties we talk about in machine learning!
Do any machine learning methods have any of these properties?

---

5. Subtlety: definition admits that a plan may not exist, but then planner may not terminate and return "failure". Such problems are undecidable. (But they cannot be formulated in classical planning.)

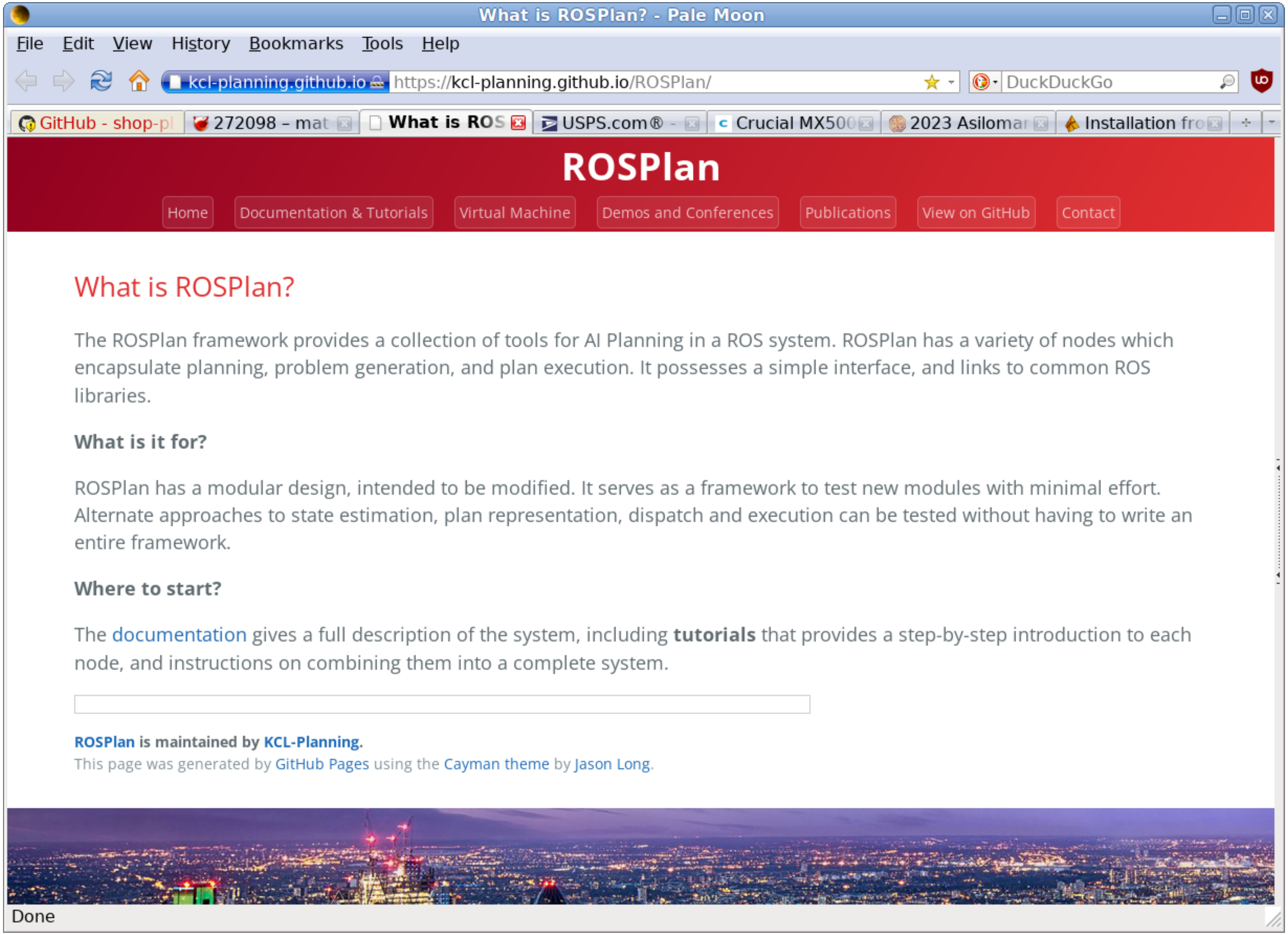
6. Of planning algorithm; not the same as the admissibility of a *solution*, plan.
Also note: admissibility $\Rightarrow$ soundness $\wedge$ completeness.

Applications of AI planning

- Space missions (NASA's Europa planner: Deep Space 1, Hubble, Mars rovers)
- Military mission planning (Army, Air Force)
- Cognitive robotics:
  - ROSPlan system
  - Ergo and Goal languages
- Logistics and scheduling
- Configuring equipment
- Planning manufacturing processes
- Crisis and emergency management (e.g. evacuation)
- Our Indigo project: slice creation in multi-operator, open RAN networks.

Classical planning used mostly in academic work, some form of hierarchical planning used in most real applications.

ROSPLAN: ROS system ∪ AI planning



What is ROSPlan?

The ROSPlan framework provides a collection of tools for AI Planning in a ROS system. ROSPlan has a variety of nodes which encapsulate planning, problem generation, and plan execution. It possesses a simple interface, and links to common ROS libraries.

What is it for?

ROSPlan has a modular design, intended to be modified. It serves as a framework to test new modules with minimal effort. Alternate approaches to state estimation, plan representation, dispatch and execution can be tested without having to write an entire framework.

Where to start?

The documentation gives a full description of the system, including **tutorials** that provides a step-by-step introduction to each node, and instructions on combining them into a complete system.

ROSPlan is maintained by KCL-Planning.

This page was generated by GitHub Pages using the Cayman theme by Jason Long.

IV Hierarchical Planning

- Adds two concepts to classical planning:
  1. tasks, also known as "high-level actions",
  2. methods for performing them.
- The hierarchy:
  - A method specifies how a task can be performed by executing a set of simpler (sub) tasks[7].
  - For a given task, different methods may apply, depending on conditions.
  - Under some conditions, many methods may be applicable to a given task.
  - The decomposition by methods is repeated until a task can be performed by the available 'primitive' actions.
- The goal: a set of tasks to perform.
- The plan: sequence of primitive actions, just as in classical planning.

7. This set can have some organization, as a task network, explained later.

1. Hierarchical planning organizes the action knowledge base in the way human domain experts would do it:

   - For every task: a set of methods/recipes for how to perform it in terms of simpler tasks.

   - Depending on conditions, different methods are applicable.

   A library of methods for the domain. Can be periodically updated (learning).

2. Methods

   – $\Rightarrow$ more intuitively-understandable plans.

   – $\Rightarrow$ drastic (exponential) reduction in search for applicable actions.
     Plan generation can be much faster.

   – From the algorithmic viewpoint, can be viewed as heuristics for the domain.

3. Hierarchical planning is strictly more expressive than classical planning[8].

---

8. More on this in the Expressiveness & Complexity part.

- A method has the form

method *name*	task's name
parameters	get bound to objects/attributes
pre-condition	logical formula involving the parameters
task network	decomposition of task into sub-tasks

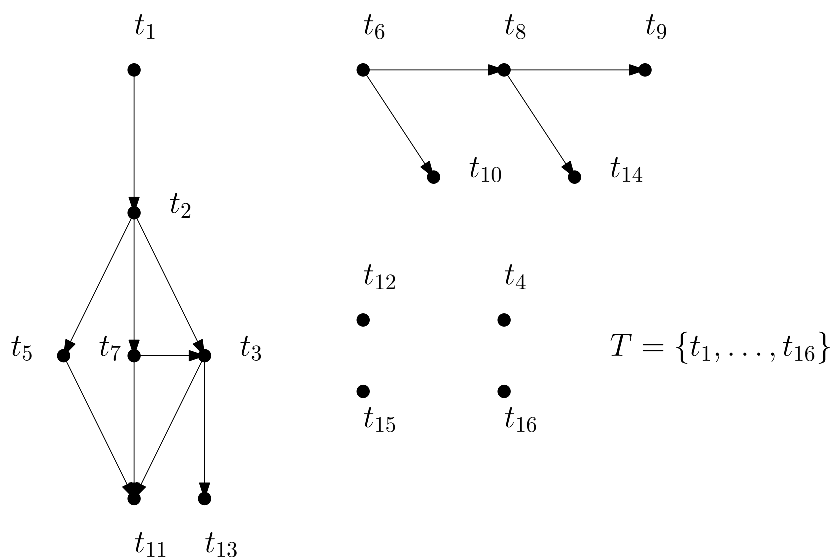
- Methods/decompositions can be recursive: e.g. $T \rightarrow T_1 T_2 T$.

- An action has the same form as in classical planning:

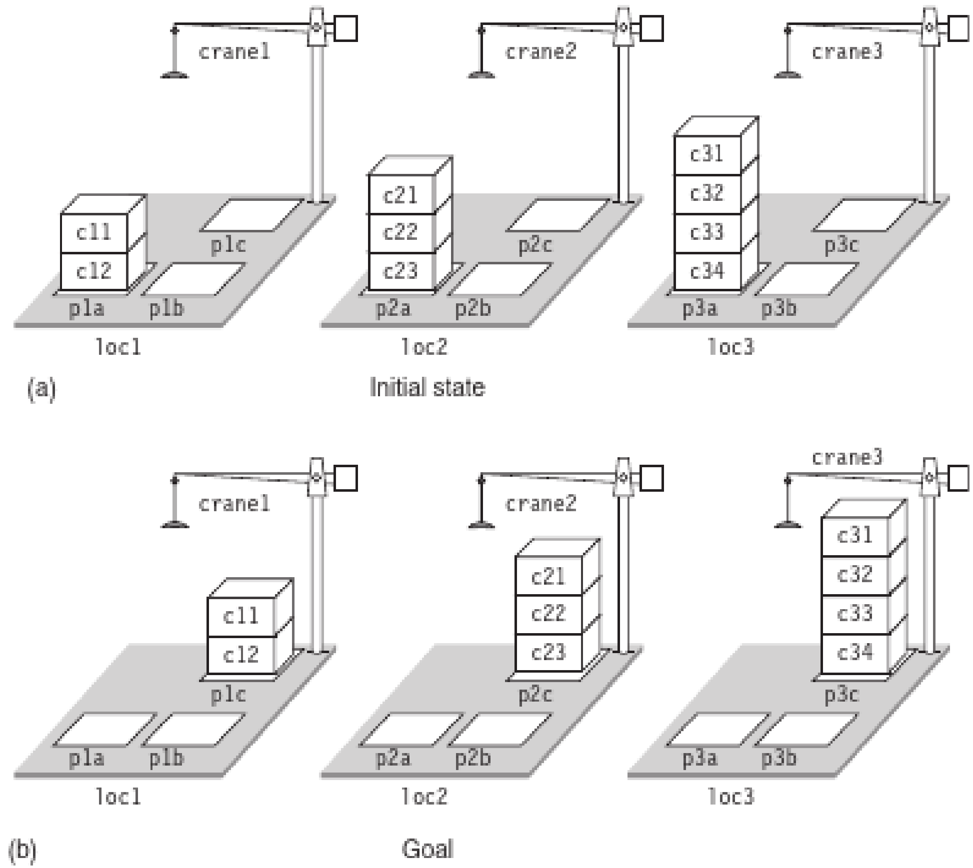
action *name*	action's name
parameters	get bound to objects/attributes
pre-condition	logical formula involving the parameters
effects	update facts in knowledge base (modify object attributes)

- The goal is a task network, describing a set of tasks to be accomplished.

- A set of tasks we want to accomplish: $T = \{t_1, t_2, \ldots, t_n\}$.
- Set can be structured as a task network. Likewise, a method's sub-tasks.
- A task network imposes a partial order on T: we can require that some tasks have to precede, or follow, some others.
- Task network: acyclic directed graph with nodes $t_1, \ldots, t_n$, where presence of edge $t_i \longrightarrow t_j$ means that t_i has to performed before t_j.
  No edge between t_k and t_l: we don't care in what order these two tasks are executed.



Example: dock worker robots



In the final state the crates must be in the same order.

Human expertise put into methods:

1. T : move a stack of crates from one pile to another, preserving order of crates.
2. To do T for a stack:

   T_1: move crates to to an intermediate pile, then
   T_2: move the crates to the final pile

3. To move pile p to pile q, do

   $M(p, q)$: if p is not empty,
   $C(p, q)$: move top-most crate of p to q
   $M(p, q)$

4. To do $C(p, q)$: use the `take(p)` action to remove the top crate from p and the `put(q)` action to put it on q.

Without the methods, a classical planner would have to do a lot of search here.

Simple comparison of the worst cases:

take a planning problem that has a plan of length ℓ.

- Classical planner: a actions, all applicable in each state. Plan $\leftrightarrow$ leaf of search tree.

  $\therefore$ worst-case no. of search nodes generated $= 1 + a + a^2 + \cdots + a^\ell = \dfrac{a^{\ell+1} - 1}{a - 1}$.

- Hierarchical planner: m methods, each decomposes a task into t sub-tasks, all methods apply at all times. Plan $\leftrightarrow$ decomposition tree.

  $\therefore$ Decomposition tree has ℓ leaves, hence $\log_t \ell$ levels. So it has
  $$1 + t + t^2 + \cdots + t^{\log_t \ell - 1} = \frac{\ell - 1}{t - 1}$$
  internal/search nodes. All m methods apply at each node,
  $\therefore$ total of $m^{(\ell-1)/(t-1)}$ decomposition trees/plans generated in the worst case.

- Worst-case no. of generated plans: a^ℓ vs. $m^{\frac{\ell-1}{t-1}}$.

$\longrightarrow$ Say $m \approx a$, and let $t \geqslant 3$ ...

$\longrightarrow$ Ideal hierarchy has small m, large t: few, long methods.

- Methods for tasks
  - hierarchical planning much more efficient than classical, and
  - hierarchy improves understandability/explainability.
- Downside: the domain's designer must specify methods for every task of interest.
- Classical planning: designer specifies only primitive actions, planner does the rest.

1. Can we mix classical and hierarchical planning?
   Methods for some tasks, just a set of primitive actions for some others?[9]
2. Methods library: plans found by a classical planner can, after some processing, be added to the methods library of a hierarchical planner for the domain.

9. Yes. Doable with any hierarchical planner: 'wrap' the primitive actions by trivial methods, and transform goal formulas into special tasks.

V Expressiveness & Complexity

1. Any classical planning problem can be easily expressed as a hierarchical planning problem[10].
2. Hierarchical task networks (HTNs) can express undecidable problems[11].
3. Classical planning is decidable[12], so hierarchical planning is strictly more expressive.

What if we put some restrictions on HTNs?

- Bounds on the number of times recursion can occur (acyclicity constraint): equivalent classical problem has exponentially larger size.
- All tasks in an HTN are just primitive actions: PLAN-EXISTENCE is NP-complete.

10. Wrap the primitive actions by trivial methods, and transform goal formulas into special tasks.

11. This depends on allowing *recursion* in methods.

12. i.e. there is an algorithm that given a problem $\mathcal{P}$ returns whether it is solvable or not: PLAN-EXISTENCE.

1. Recall: domain is represented in 1st-order logic.
2. Two problems: PLAN-EXISTENCE, PLAN-LENGTH(k).
3. Decidability:

Function symbols?	PLAN-EXISTENCE	PLAN-LENGTH(k)
No	Decidable	Decidable
Yes	Semi-decidable	Decidable

4. Complexity[13]:

All atoms ground?	PLAN-EXISTENCE	PLAN-LENGTH(k)
No	EXPSPACE-complete	NEXPTIME-complete
Yes	PSPACE-complete	PSPACE-complete

"All atoms ground" is a severe restriction: representation size increases exponentially.

Recall:

$$\text{NLOGSPACE} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME} \subseteq \text{EXPSPACE}.$$

Reference: *Automated Planning: Theory and Practice*, M. Ghallab et al, Elsevier, 2004.

13. With no function symbols allowed.

- Hierarchical planning is more expressive than classical planning.
- So the complexity results for classical planning provide lower bounds for the complexity of hierarchical planning.

There are also much more detailed complexity results:

Complexity results for HTN planning, K. Erol et al. Annals of Mathematics and Artificial Intelligence 18 (1996).

Tight Bounds for HTN Planning, R. Alford et al. 25th Int'l Conference on Planning and Scheduling (ICAPS), 2015.

Assessing the Expressivity of Planning Formalisms through the Comparison to Formal Languages, D. Höller et al. 26th Int'l Conference on Planning and Scheduling (ICAPS), 2016.

VI Planning & Learning

Not unrelated: can be used for some of the same problems, e.g. game playing.

Otherwise, comparison isn't straightforward…

- Training: none vs. extensive.
- Hierarchical planning takes more human investment than deep learning.
- Understandability and explainability:
  - Specification of a planning domain (objects, methods, actions) can be made as understandable as well-documented code can.
  - All sorts of efforts to make deep learning 'explainable'…
- Desirable properties of planning algorithms have no counterpart in deep learning.
- Planning is a search problem, learning is an optimization problem.
  Isn't optimization better than search?

- Planning, where agent's actions have uncertain effects on the world[14].
- A Markov decision process, MDP, consists of a set of (world) states S and a set of (agent) actions A:
  - If action $a \in A$ is taken in state $s \in S$, the process moves to $s' \in S$ with probability $P(s'|s,a)$.
  - A policy $\pi: S \rightarrow A$ defines what action from A to take in each state of S.
  - Also, costs can be associated with pairs (s,a), and rewards with states s.
- A history h of the process is a sequence of states $h = s_0, s_1, \ldots, s_n, \ldots$
- So given π we can assign a probability $P(h|\pi)$ to a history, and a value $V(h|\pi)$ to it.
- Then the expected value of policy π is the sum over histories
$$E(\pi) = \sum_{h \in H} V(h|\pi)\, P(h|\pi).$$
- Algorithms, based on dynamic programming, can compute (learn) an optimal π.

14. The world/environment is fully observable, but uncertain.

- We don't stress optimality in classical planning because even without it, finding a plan is in the worst-case a computationally hard problem.
- But a classical planning problem can be easily expressed as a determinized[15] MDP with rewards associated only with goal states, and then

  optimal MDP policy = classical deterministic plan.

- Thus classical planning can be viewed as a special case of MDP policy-finding, an optimization problem.
- Therefore solving MDPs is at least as hard as solving classical planning problems! But this is not often mentioned, and people routinely solve MDPs . . .

(Aside: MDPs can also be solved non-iteratively, by a transformation into a linear program. So can they be solved in polynomial time?)

15. 'determinized': for each state, any action applicable to it leads to exactly one next state.

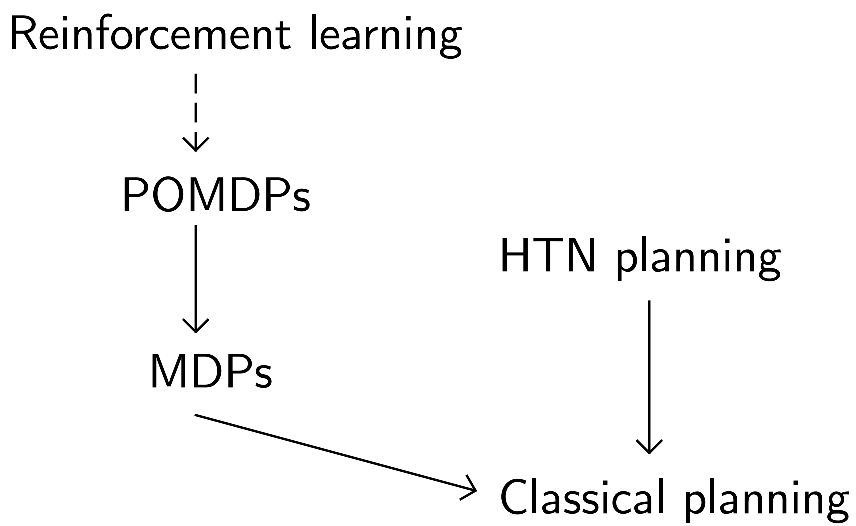
- There is much more known about the complexity of probabilistic planning . . .
- The complexity results say that there are problem instances that are hard. Methods for solving MDPs are an active research area.

References:

Probabilistic Propositional Planning: Representations and Complexity, M. Littman. Proceedings AAAI/IAAI (1997).
The Computational Complexity of Probabilistic Planning, M. Littman et al. Journal of Artificial Intelligence Research 9 (1998).
On the undecidability of probabilistic planning and related stochastic optimization problems, O. Madani et al. Artificial Intelligence 147 (2003).

- Environment of MDP not fully observable $\Rightarrow$ partially-observable MDP: POMDP.
- Agent's knowledge about the environment's state s is only a probability distribution, the "belief state".
- Sensor model: $p(o|\, s, a)$, where o is the observation if in s and last action was a.
- There is an initial belief state, updated when observations are made.
- An MDP is a special case of a POMDP.
- Solution of POMDPs much more difficult than MDPs: active research area.
- POMDPs are a general model for learning in uncertain environments.
- Some algorithms for reinforcement learning (RL) can be modelled by POMDPs.

The learning problems that are currently attracting attention (and some hype) in AI sit at the top of the difficulty hierarchy:



Nevertheless, there is also a lot of work in the other areas.

VII Learning theory

- How well can we learn from examples?
- What are limits to learning?

A simple question[16]:

> given finite sets X, Y, how many observations $(x, y) \in X \times Y$ do we need to approximate an unknown function
>
> $$h : X \to Y$$
>
> to a prescribed accuracy?

16. 'Simple' compared to, say, recognizing an image.

- Hypothesis space $\mathcal{H}$: some subset of the space Y^X of all functions $h : X \to Y$.
- Have a sample $z \triangleq \{(x_1, y_1), \ldots, (x_n, y_n)\} \in (X \times Y)^n$.
- Assume z is generated i.i.d. by a p.d. $p(x, y)$, where
  p : unknown and arbitrary, but time-invariant.
- Evaluate h by 0-1 loss: $\ell : Y \times Y \to \{0, 1\}$.
  $\ell(y', y) = 1$ iff predicted output $y' \neq$ correct output y.
- Knowing the sample z, find an $h \in \mathcal{H}$ with minimum expected loss (risk)

$$L(h) \triangleq \textstyle\sum_{x \in X, y \in Y} \ell\big(h(x), y\big)\, p(x, y).$$

[$L(h)$ does not depend on z, and cannot be calculated! We don't know p.]

ML terminology: discriminative model, supervised learning, classification loss, generalization error $L(h) - L(h^*)$, where $h^* \triangleq \operatorname{argmin}_{h \in \mathcal{H}} L(h)$.

Approximately correct learning

- h is ε-approximately correct: $L(h) - L(h^*) \leqslant \varepsilon$.
- Suppose $\mathcal{H}$ is discrete and finite. Then

$$\Pr_p\big(\forall h \in \mathcal{H} : |L(h) - L_{\text{emp}}(h, z)| \leqslant \varepsilon\big) > 1 - 2\,|\mathcal{H}|\,\mathrm{e}^{-2n\varepsilon^2}, \tag{1}$$

and if we restrict h to the subset $V_{\mathcal{H}}(z)$ of $h \in \mathcal{H}$ with $L_{\text{emp}}(h, z) = 0$,

$$\Pr_p\big(\forall h \in V_{\mathcal{H}}(z) : L(h) \leqslant \varepsilon\big) > 1 - 2\,|\mathcal{H}|\,\mathrm{e}^{-n\varepsilon/4}. \quad [17] \tag{2}$$

Interpretation of these statements needs care!

- ε-approximate learning in $V_{\mathcal{H}}$ with 'confidence' $1 - \delta$:

$$n > \frac{4}{\varepsilon} \ln \frac{2\,|\mathcal{H}|}{\delta} \quad \Rightarrow \quad \Pr_p\big(\forall h \in V_{\mathcal{H}}(z) : L(h) \leqslant \varepsilon\big) > 1 - \delta. \tag{3}$$

- Sample complexity for this supervised learning problem is $\propto \ln |\mathcal{H}|$.

17. (1) is known as a "VC bound", and (2) as a "PAC bound". For some $\varepsilon, \mathcal{H}$ the bounds are vacuous.

Example: boolean functions

– A sample $(x_1, y_1), \ldots, (x_n, y_n)$ with $x_i \in \{0,1\}^m$, $y_i \in \{0,1\}$.

– Reasonable: this comes from a boolean function h of m arguments. Then

$$|\mathcal{H}| = 2^{2^m}. \tag{4}$$

[How do you get that? A boolean fn. of m bits has a truth table with 2^m rows and $m+1$ columns. Fix the first m columns of the table. To specify a function you need to fill out the last column with 0s and 1s in a particular way. How many ways are there?]

– Our deep learning algorithm yields an h that matches the sample exactly: $h \in V_{\mathcal{H}}(z)$.

- Say we want h to be 0.05-correct, with confidence 99.9%: from (3) and (4),

$$\hat{n} = \frac{4}{\varepsilon} \ln \frac{2\,|\mathcal{H}|}{\delta} \qquad \rightarrow \qquad \hat{n} = \frac{4}{0.05}\left(2^m + \ln \frac{1}{0.001}\right).$$

- So ...?

$$\hat{n} = \frac{4}{\varepsilon} \ln \frac{2\,|\mathcal{H}|}{\delta} \qquad \rightarrow \qquad \hat{n} = \frac{4}{0.05}\left(2^m + \ln \frac{1}{0.001}\right).$$

- m, how big can it be?
- Why do we have to look at almost all possible inputs?
- Accuracy ε and confidence δ: don't matter much if m is big.
- This result holds no matter what the learning algorithm is!
- How do we deal with this complexity bound?

[This is just the surface of learning theory. Much more is known about sample complexity.
E.g. D. McAllester, *A PAC-Bayesian Tutorial with A Dropout Bound (2013)*, improves the '4' above to '2'.
Also see P. Alquier, *User-friendly introduction to PAC-Bayes bounds*, ArXiV, March 2023.]