

Action Selection in Intelligent Systems & Research at the AI Group, UPF

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, Spain

Autonomous Behavior in AI

How to develop systems or 'agents'
that can make decisions on their own?

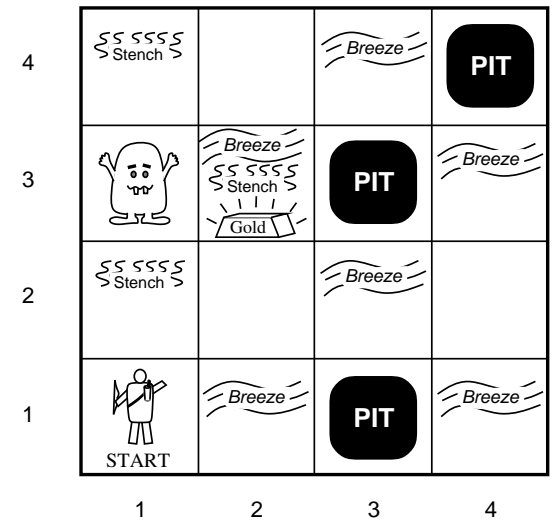
Wumpus World PEAS description

Performance measure

gold +1000, death -1000
 -1 per step, -10 for using the arrow

Environment

Squares adjacent to wumpus are smelly
 Squares adjacent to pit are breezy
 Glitter iff gold is in the same square
 Shooting kills wumpus if you are facing it
 Shooting uses up the only arrow
 Grabbing picks up gold if in same square
 Releasing drops the gold in same square



Actuators Left turn, Right turn,
 Forward, Grab, Release, Shoot

Sensors Breeze, Glitter, Smell

Autonomous Behavior in AI: The Control Problem

The key problem is to select **the action to do next**. This is the so-called **control problem**. Three approaches to this problem:

- **Programming-based:** Specify control by hand
- **Learning-based:** Learn control from experience
- **Model-based:** Specify problem by hand, derive control automatically

Approaches not orthogonal though; and successes and limitations in each . . .

Settings where greater autonomy required

- **Robotics**
- **Video-Games**
- **Web Service Composition**
- **Aerospace**
- **Manufacturing**
- **⋮**

Solution 1: Programming-based Approach

Control specified by programmer; e.g.,

- *don't move into a cell if not known to be safe (no Wumpus or Pit)*
- *sense presence of Wumpus or Pits nearby if this is not known*
- *pick up gold if presence of gold detected in cell*
-

Advantage: domain-knowledge easy to express

Disadvantage: cannot deal with situations not anticipated by programmer

Solution 2: Learning-based Approach

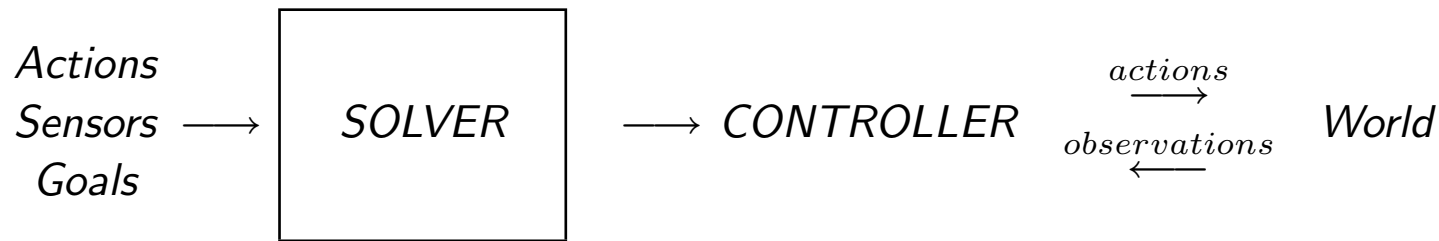
- **Unsupervised** (Reinforcement Learning):
 - ▷ penalize agent each time that it 'dies' from Wumpus or Pit
 - ▷ reward agent each time it's able to pick up the gold, . . .
- **Supervised** (Classification)
 - ▷ learn to classify actions into good or bad from info provided by teacher
- **Evolutionary**:
 - ▷ from pool of possible controllers: try them out, select the ones that do best, and mutate and recombine for a number of iterations, keeping best

Advantage: does not require much knowledge in principle

Disadvantage: in practice though, right features needed, incomplete information is problematic, and unsupervised learning is slow . . .

Solution 3: Model-Based Approach

- specify model for problem: actions, initial situation, goals, and sensors
- let a solver compute controller automatically



Advantage: flexible, clear, and domain-independent

Disadvantage: need a model; computationally intractable

*Model-based approach to intelligent behavior called **Planning** in AI*

A basic planning model

Model for problems where actions to be selected for driving a system from some initial state s_0 to a target state s_g :

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

A **solution** is a sequence of applicable actions that maps s_0 into $s_g \in S_G$

A more complex planning model: POMDPs

POMDPs are **partially observable, probabilistic** state models:

- states $s \in S$
 - actions $A(s) \subseteq A$
 - transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
 - initial **belief state** b_0
 - final **belief states** b_F
 - **sensor model** given by probabilities $P_a(o|s)$, $o \in Obs$
-
- **Belief states** are probability distributions over S
 - **Solutions** are policies that map belief states into actions
 - **Optimal** policies minimize **expected** cost to go from b_0 to b_F

Example: A logistics problem in PDDL

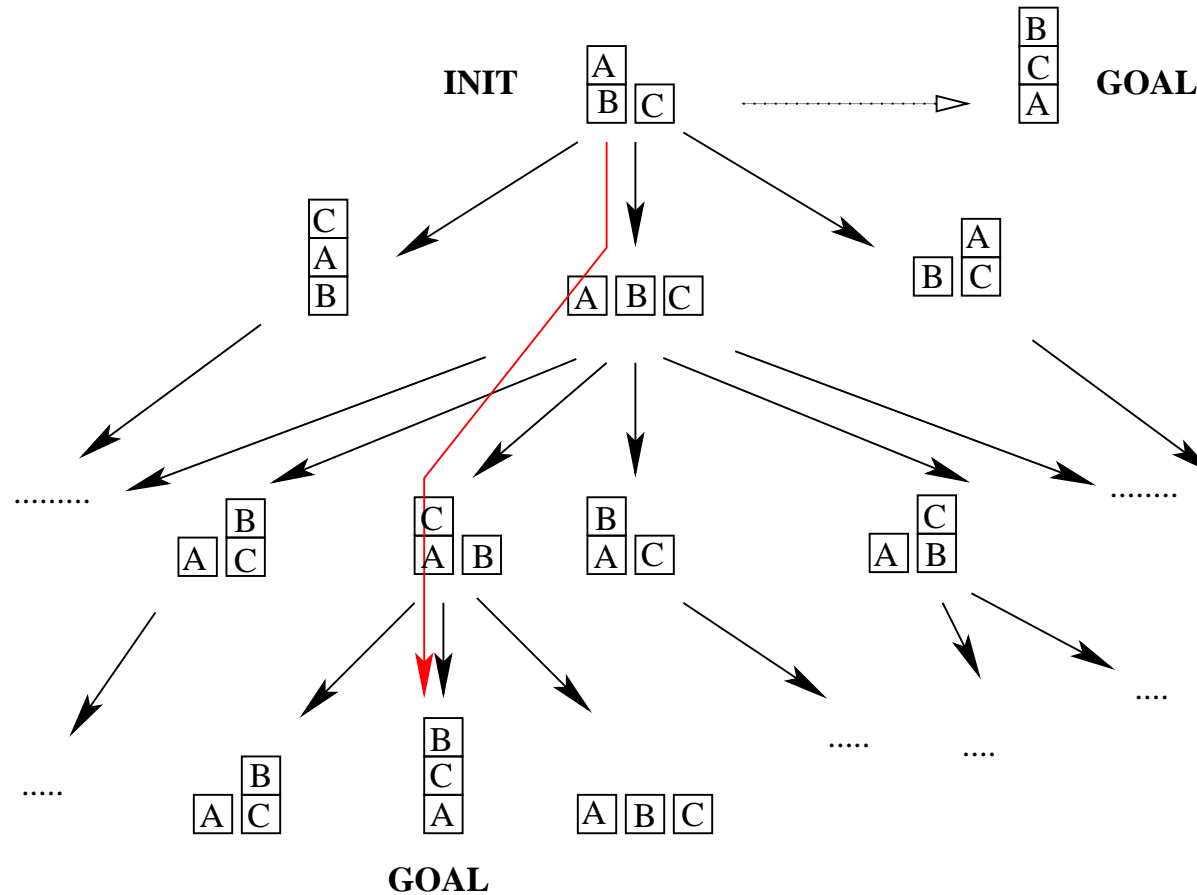
```
(define (domain logistics)
  (:types airport - location  truck plane - vehicle  vehicle packet - thing ...)
  (:predicates (loc-at ?x - location ?y - city) (at ?x - thing ?y - location) ...)

  (:action load-pkg-in-vehicle
    :parameters (?x - packet ?y - vehicle)
    :vars (?z - location)
    :precondition (and (at ?x ?z) (at ?y ?z))
    :effect (and (not (at ?x ?z)) (in ?x ?y)))

  (:action drive-vehicle
    :parameters (?x - truck ?y - location)
    :vars (?z - location ?c - city)
    :precondition (and (loc-at ?z ?c) (loc-at ?y ?c) (not (= ?z ?y)) (at ?x ?z))
    :effect (and (not (at ?x ?z)) (at ?x ?y)))
  ...)

(define (problem log3_2)
  (:domain logistics)
  (:objects packet1 packet2 - packet  truck1 truck2 truck3 - truck  plane1 - plane)
  (:init (at packet1 office1) (at packet2 office3) ...)
  (:goal (and (at packet1 office2) (at packet2 office2))))
```

How planning problems are solved? (1)

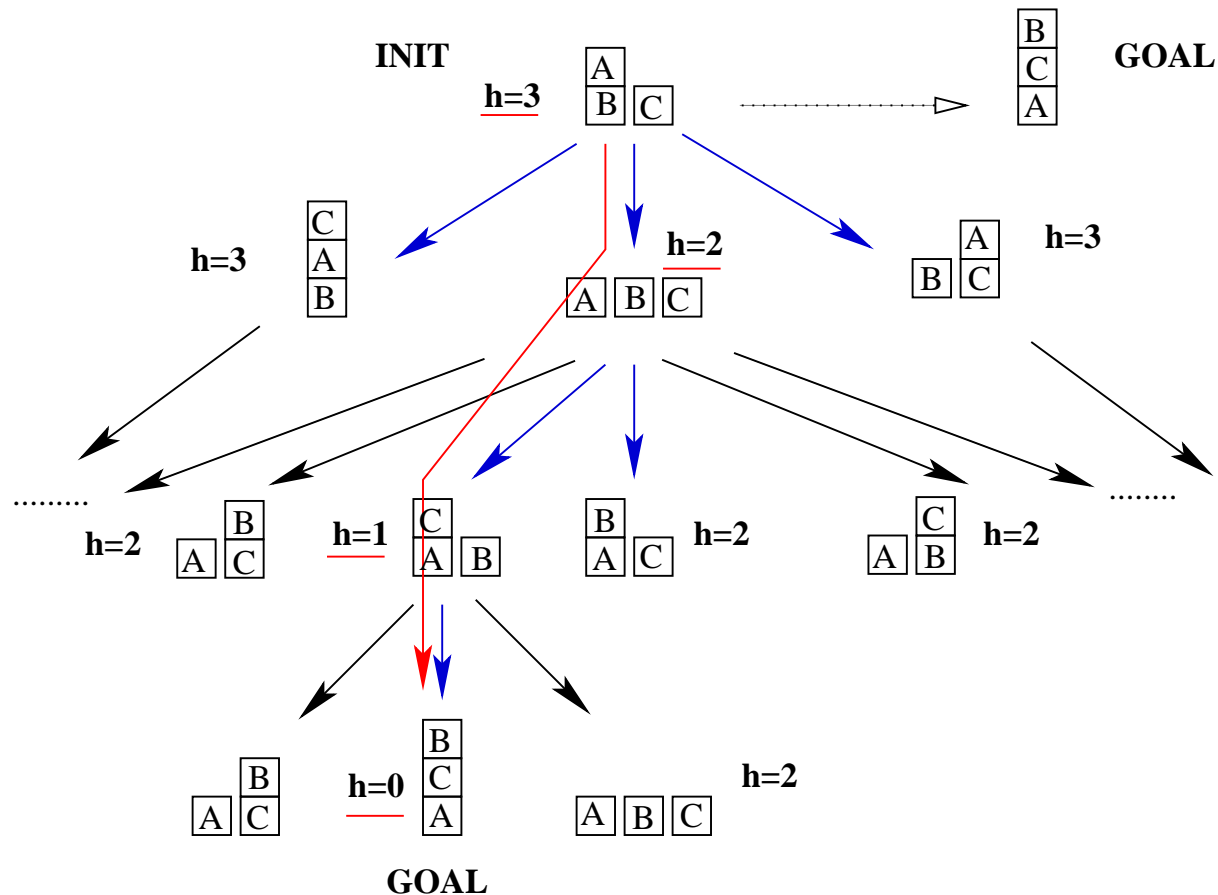


- Given the **actions** that move a 'clear' block to the table or onto another 'clear' block, **find a plan** to achieve the goal
- Problem becomes **finding a path** in a huge **directed graph**

How planning problems are solved? (2)

- How do we find a route in a map from Barcelona to Madrid?
- Need **sense of direction**: whether an action takes us towards the goal or not
- In AI, this is captured by **heuristic functions**: functions $h(s)$ that provide an **estimate of the cost** from any state s to the goal
- Key new idea in planning is that useful heuristics $h(s)$ can be obtained **automatically** from the problem encoding
- **How?** Solving a **relaxed problem** where '**deletes**' are dropped
- Heuristic $h(s)$ is cost of solution found for **relaxed problem** in **poly-time**

How is our problem solved?



- Provided with the **heuristic** h , plan found without search by **hill-climbing**
- Actually, only the states reached by the actions in **blue** evaluated

The appraisals $h(s)$ from a cognitive point of view

- they are **opaque** and thus cannot be **conscious**

meaning of symbols in the relaxation is not the normal meaning; e.g., objects can be at many places at the same time as old locations not deleted

- they are **fast and frugal** (linear-time), but unlike the 'fast and frugal heuristics' of Gigerenzer et al. are **general**

they apply to all problems fitting the model (planning problems)

- they play the role of '**gut feelings**' or '**emotions**' according to De Sousa 87, Damasio 94, Evans 2002, Gigerenzer 2007

providing a guide to action while avoiding infinite regresses in the decision process

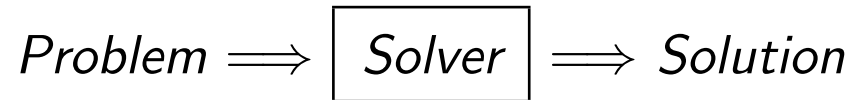
Research at the AI Group, UPF

Recent issues of AIJ, JAIR, AAI or IJCAI shows papers on:

1. **SAT and Constraints**
2. **Search and Planning**
3. **Probabilistic Reasoning**
4. **Probabilistic Planning**
5. Inference in First-Order Logic
6. Machine Learning
7. Natural Language
8. Vision and Robotics
9. Multi-Agent Systems

Areas 1–4 often deemed about **techniques**, but more accurate to regard them as **models** and **solvers** . . .

Example: Solver for Linear Equations

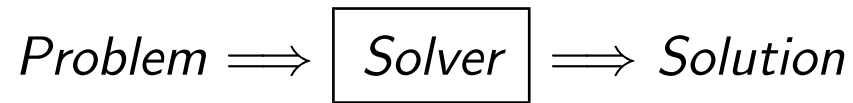


- **Problem:** The age of John is 3 times the age of Peter. In 10 years, it will be only 2 times. How old are John and Peter?
- **Expressed as:** $J = 3P$; $J + 10 = 2(P + 10)$
- **Solver:** Gauss-Jordan (Variable Elimination)
- **Solution:** $P = 10$; $J = 30$

Solver is **general** as deals with any problem expressed as an instance of **model**

Linear Equations Model, however, is **tractable**, AI models are not . . .

AI Solvers



- The basic models and tasks we study are
 - ▷ **Constraint Satisfaction/SAT**: find state that satisfies constraints
 - ▷ **Bayesian Networks**: find probability over variable given observations
 - ▷ **Planning Problems**: find actions that map given state into final state
 - ▷ **Planning with Feedback**: find strategy for mapping state into final state
- All of these models are **intractable**, and some extremely powerful (POMDPs)
- The challenge is computational: **how to scale up**
- For this, solvers must **recognize and exploit structure** of the problems
- Methodology is **empirical**: benchmarks and competitions
- Significant progress in recent years: experimental and conceptual

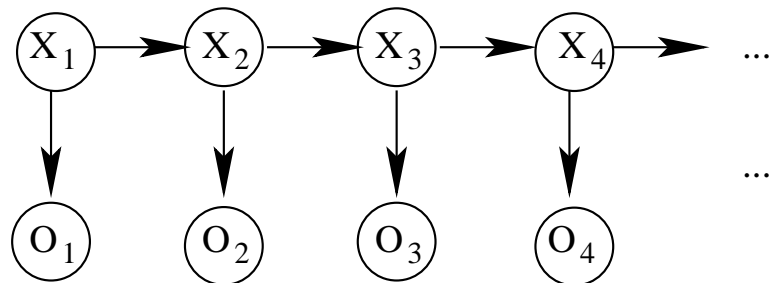
Bayesian Networks

A Bayesian Network (Pearl 1988) is a compact representation of a joint probability distribution over a set of variables X_1, \dots, X_n made up of:

- a DAG where the nodes are the variables X_1, \dots, X_n
- Conditional probability tables $prob(X_i|pa(X_i))$, $i = 1, \dots, n$, where $pa(X_i)$ refers to the parents of X_i in the DAG

The DAG implicitly defines a set of **independences** that result in joint distrib.

$$P(X_1, \dots, X_n) = \prod_{i=1, n} P(X_i|pa(X_i))$$



A **Hidden Markov Model** is a **Bayesian Tree** solvable in **linear-time**.

SAT and CSPs

- **SAT**: determine if there is a **truth assignment** that satisfies a set of clauses

$$x \vee \neg y \vee z \vee \neg w \vee \dots$$

- Problem is NP-Complete, which in practice means worst-case behavior of SAT algorithms is **exponential** in number of variables
- Yet current SAT solvers manage to solve problems with **thousands of variables and clauses**, and used widely (circuit design, verification, planning, etc)
- Key is **efficient (poly-time) inference** in every node of search tree: **unit resolution, conflict-based learning, . . .**
- Many other ideas **logically possible**, but **do not work** (don't scale up).
- Same for **Constraint Satisfaction Problems (CSPs)**

Related Tasks: From SAT to Bayesian Networks

- **Weighted MAX-SAT**: find assignment σ that minimizes total cost $w(C)$ of violated clauses

$$\sum_{C:\sigma \not\models C} w(C)$$

- **Weighted Model Counting**: Adds up 'weights' of satisfying assignments:

$$\sum_{\sigma:\sigma \models T} \prod_{L \in \sigma} w(L)$$

SAT methods extended to these other tasks, closely connected to **probabilistic** reasoning tasks over **Bayesian Networks**:

- **Most Probable Explanation (MPE)** easily cast as Weighted MAX-SAT
- **Probability Assessment** $P(X|Obs)$ easily cast as Weighted Model Counting

Current best BN solvers built over this formulation (ACE, Weighted Cachet)

Artificial Intelligent Group at UPF

- **Faculty:**

- ▷ Hector Geffner (Ph.D. UCLA, 1989; ICREA)
- ▷ Victor Dalmau (Ph.D. UPC, 2000)
- ▷ Hubie Chen (Ph.D. Cornell Univ. 2004)
- ▷ Anders Jonsson (Ph.D. Univ. of Massachusetts 2005)

- **Graduate Students**

- ▷ Hector Palacios
- ▷ Miquel Ramirez
- ▷ Alex Albore
- ▷ Emil Keyder
- ▷ Nir Lipovetzky

- **Research:** Action Selection, Planning, Search and Inference, Bayesian Networks, Reinforcement Learning, (Weighted) Constraint Satisfaction, Algorithms and Complexity, . . .