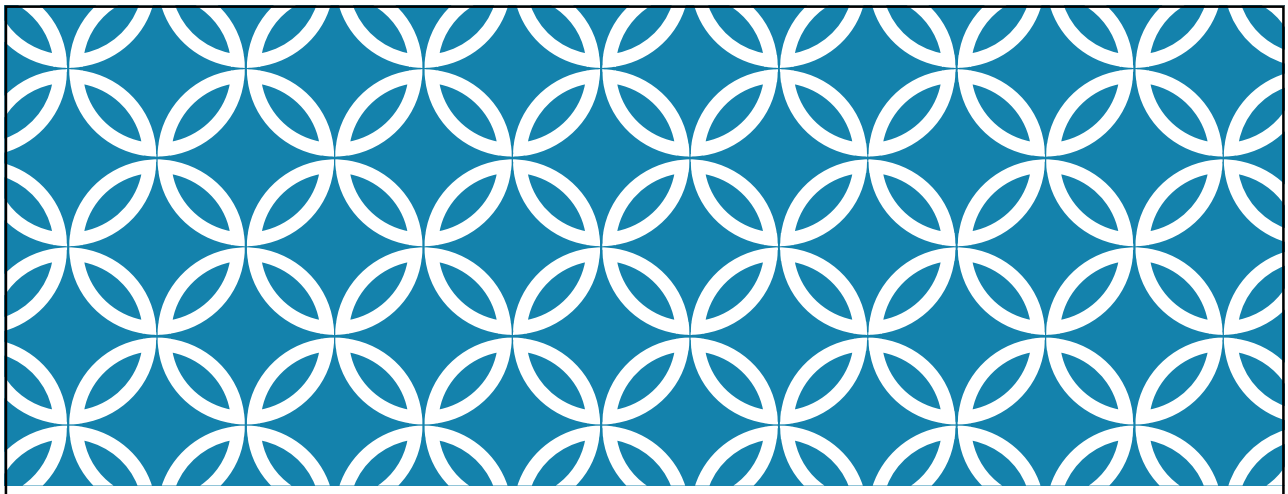


ARTIFICIAL INTELLIGENCE

1

RECAP

AI?

Agent?

2

RECAP

Fully observable vs. partially observable

Single agent vs. multiagent

Deterministic vs. stochastic

Episodic vs. sequential

Static vs. dynamic

Discrete vs. continuous

Known vs. unknown

AGENTS: STRUCTURE

The job of AI is to design an **agent program** that implements the agent function—the mapping from percepts to actions

We assume this program will run on some sort of computing device with physical sensors and actuators—we call this the **architecture**

Agent = Program + Architecture

AGENT TYPES

simple reflex agents

reflex agents with state

goal-based agents

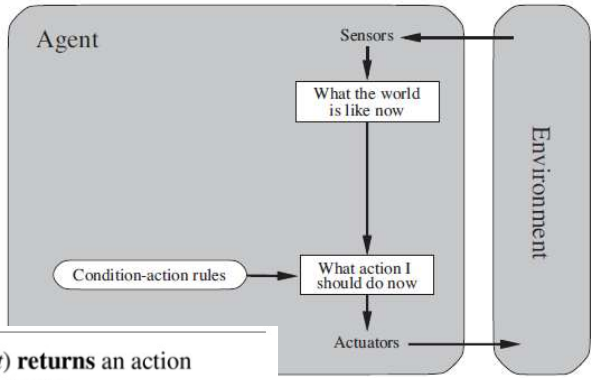
utility-based agents

All of these 4 types can be turned into learning agents

5

SIMPLE REFLEX AGENT

if *car-in-front-is-braking* **then** *initiate-braking*



function SIMPLE-REFLEX-AGENT(*percept*) **returns** an action
  persistent: *rules*, a set of condition–action rules

  state ← INTERPRET-INPUT(*percept*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*

A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

6

CATCH:

"only if the correct decision can be made on the basis of just the current percept—that is, only if the environment is fully observable."

The most effective way to handle partial observability is for the agent to *keep track of the part of the world it can't see now*.

That is, the agent should maintain some sort of **internal state** that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state

Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program in some form.

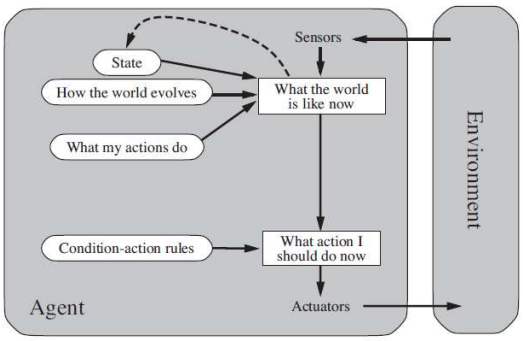
- First, we need some information about how the world changes over time, which can be divided roughly into two parts:
- the effects of the agent's actions and
- how the world evolves independently of the agent.

7

MODEL-BASED REFLEX AGENTS

How the world evolves independently of agent
e.g., road is slippery during rain

What my actions do e.g., when steering wheel turns clockwise, car moves to the right



function MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action
  persistent: *state*, the agent's current conception of the world state
    model, a description of how the next state depends on current state and action
    rules, a set of condition–action rules
    action, the most recent action, initially none

  state ← UPDATE-STATE(*state*, *action*, *percept*, *model*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*

8

Knowing something about the current state of the environment is not always enough to decide what to do.

- For example, at a road junction, the taxi can turn left, turn right, or go straight on.
- The correct decision depends on where the taxi is trying to get to.

In other words,

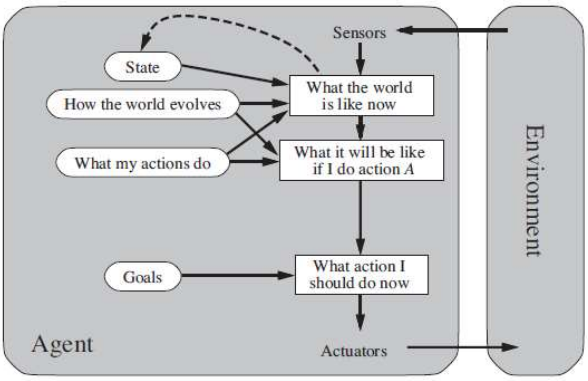
- as well as a current state description,
- the agent needs some sort of **goal** information that describes situations that are desirable

9

GOAL-BASED AGENTS

Sometimes an agent needs some sort of **goal** information that describes situations that are desirable—for example, being at the passenger's destination.

"Searching" and "Planning" are subfields of AI that describe such agents.



A model-based, goal-based agent

10

Goals alone are not enough to generate high-quality behavior in most environments.

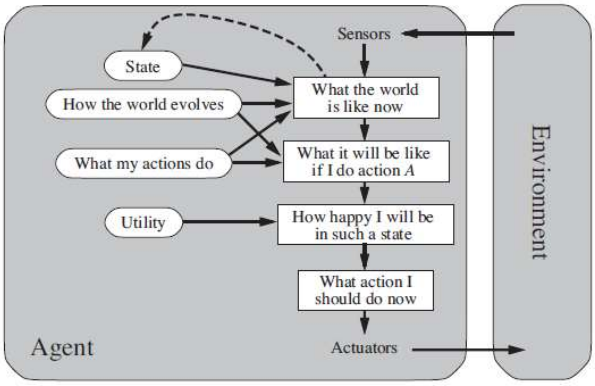
For example, many action sequences will get the taxi to its destination (thereby achieving the goal), but some are quicker, safer, more reliable, or cheaper than others

UTILITY-BASED AGENTS

Many ways to achieve a "goal" e.g. to reach a particular destination but which one is quicker, cheaper and more reliable?

An agent's **utility function** is essentially an internalization of the performance measure.

A model-based, utility-based agent chooses the action that leads to the best expected utility, that is, the utility the agent expects to derive, on average, given the probabilities and utilities of each outcome.



A model-based, utility-based agent

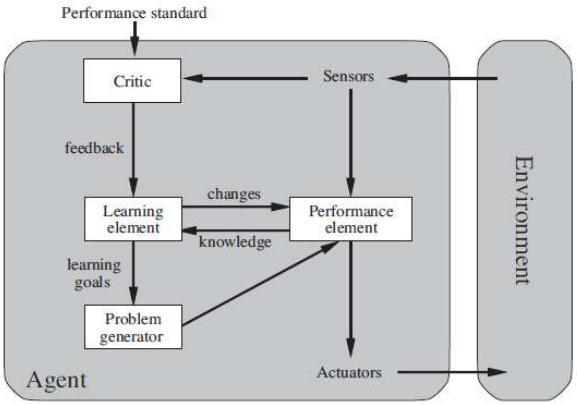
LEARNING AGENTS

Learning Element: making improvements

Performance Element (Agent in previous slides): selecting external actions.

The critic tells the learning element how well the agent/performance element is doing with respect to a fixed performance standard.

Problem Generator: suggesting actions that will lead to new and informative experiences, doing suboptimal tasks in shortrun for better overall experience in longrun.



13

ENVIRONMENT REPRESENTATION

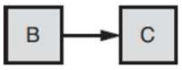
Atomic

Factored

Structured

14

ATOMIC ENVIRONMENT



Environment State B or C represented as a single entity

Algorithms underlying search and game-playing (Chapters 3–5), HiddenMarkov models (Chapter 15), and Markov decision processes (Chapter 17) all work with atomic representations.

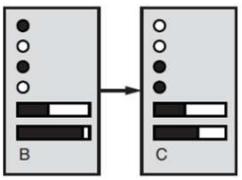
Example:

- Consider the problem of finding a driving route from one end of a country to the other via some sequence of cities

15

FACTORED ENVIRONMENT

Each state of the environment has multiple entities



Example:

- Same as in previous slide except that now we are also interested in amount of fuel left in the tank, current GPS coordinates etc.

Each state can share some attributes with another state

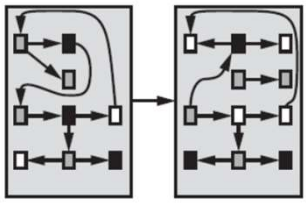
A good representation for uncertainty.

Constrained Satisfaction (Ch 6)
Propositional Logic (Ch 7)
Planning (Ch 10, 11)
Bayesian Networks (Ch 13, 16)
Machine Learning (Ch 18, 20, 21)

16

STRUCTURED ENVIRONMENT

Structured representation: a state includes objects, each of which may have attributes of its own as well as relationships to other objects.



First Order Logic (Ch 8,9, 12)

First Order Probability Models (Ch 14)

Knowledge based learning (Ch 19)

Natural Language Understanding (Ch 22, 23)

BENCHMARKING

benchmarking—running the algorithms on a computer and measuring speed in seconds and memory consumption in bytes.

measures the performance of a particular program written in a particular language, running on a particular computer, with a particular compiler and particular input data.

We want to measure the complexity of algorithm itself, not of the machine in which that algorithm runs.

ASYMPTOTIC ANALYSIS

How many times the loop will be executed ?

Let 'n' be the length of the sequence. The complexity of the code on the right is O(n).

```
function SUMMATION(sequence) returns a number
  sum ← 0
  for i = 1 to LENGTH(sequence) do
      sum ← sum + sequence[i]
  return sum
```


COMPLEXITY ANALYSIS

O() -> Complexity of algorithms. How about complexity of the problem itself?

The first gross division is between problems that can be solved in polynomial time and problems that cannot be solved in polynomial time, no matter what algorithm is used.

The class of polynomial problems—those which can be solved in time $O(n^k)$ for some k—is called P.

NP PROBLEM

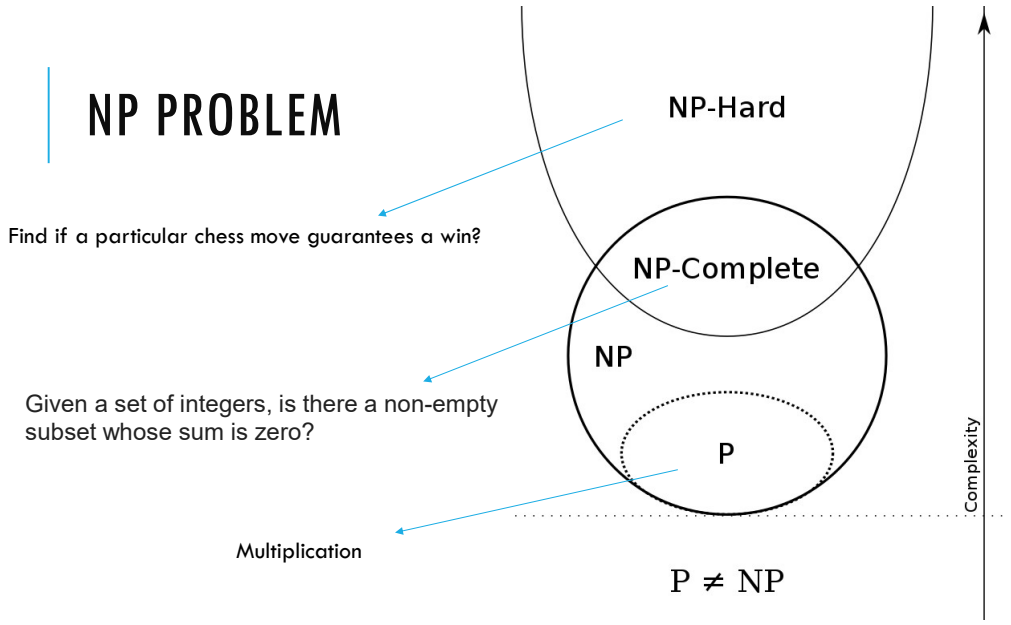
Another important class of problems is NP, the class of nondeterministic polynomial problems.

A problem is in this class if there is some algorithm that can guess a solution and then verify whether the guess is correct in polynomial time.

NP Hard Problem: Even if a solution is available, it is sometimes impossible to determine in polynomial time if it's a right solution.

21

NP PROBLEM



22