2-0: Overview

- What makes an agent?
- Defining an environment
- Types of agent programs
What makes an agent?
Defining an environment
Types of agent programs
2-2: What is an agent?

- There are lots of potential definitions ...

  R & N: An agent is anything that can be viewed as perceiving its environment through sensors and acting on that environment through actuators.

  Woolridge: An agent is a computer system that is located in an environment and is capable of autonomous action.
2-3: Qualities of an agent

- Autonomy
- Adaptation
- Goal-directed behavior
- Has “beliefs” and “intentions”
- Proactive
- Situated within an environment
Autonomy is a quality often attributed to agents.

An autonomous agent is able to rely on its percepts and past experience to make decisions, rather than asking a human for help.

This is a thorny area - most agents will not have complete autonomy.

Challenge: Designing an agent that can reason about its own autonomy and know when to ask for help.
Agents can be thought of as the next logical step after objects.

- “Objects do it for free, agents do it for money.”

Objects are *receivers* of actions, agents are actors.

It’s less useful to think of agent as an *objective* label than as a *subjective* description.

Agency is a useful abstraction for us as programmers.

- Allows us to think about a program at a higher level.

Treat something as an agent if that helps to understand, predict, or explain its behavior.

- Thermostats as agents
Why bother with all this? We already know how to write programs.

Agents tend to be open-ended programs

- Difficult to specify in advance what they should do in all cases.

It’s helpful to talk about them as if they were intelligent.

“The robot wants to find the power supply.”

“The server believes that the client has reset.”

This assigning of mental states to programs is called the intentional stance.
2-7: Agents and the Environment

- **Percepts:** Information delivered to an agent’s sensors. (light, sound, EM waves, signals)

- **Sensors:** An agent’s mechanisms for gathering data about its environment. (eyes, ears, photoelectric cells, ...)

- **Actuators:** An agent’s mechanisms for affecting its environment. (Wheels, arms, radios, lights, etc)

- **Actions:** Actual changes to the environment. (running, rolling)
We can describe our agent’s behavior as a function $F$:

$$\text{Action} = F(\text{current-percept, percept-history}).$$

Maps a percept sequence to an action.

Actually implementing this function is the work of an agent program.

That’s what we’ll spend most of our time on.
2-9: Example: Vacuum-cleaner World

- Robotic vacuum cleaners are actually on the market.
- $200 at Amazon
- A reflex agent
2-10: Example: Vacuum-cleaner World

Let’s start with a very simple approximation.

Two rooms, A and B. Each room can be either clean or dirty.

This is the agent’s environment.

Sensors: Dirt sensor, location.

Actuators: Vacuum, wheels.

Percepts: Clean, Dirty

Actions: Move left, move right, suck, do nothing.
In this simple world, we could list all the possible percept sequences and associated actions. This is known as a table-based or lookup agent.

Question: How do we fill in the best action for each percept sequence?

Great for simple worlds, but doesn’t scale.

We need a more compact representation for this table.
Roughly, rationality means “doing the right thing”
More precision is needed - what is “the right thing”? We need a definition of success.
Begin with a performance measure
- This is a condition or state of the world we’d like the agent to achieve.
- “Both rooms are clean.” (perhaps more criteria, such as minimizing time, power consumed, or number of actions taken)
- We might prefer a scalar measure or a boolean condition.
Notice that this is a specification of an *outcome*, rather than how an agent should behave.

A rational action is one that tries to maximize an agent’s performance measure, given its percepts and actions.

R & N: Rational agents: For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.
“expected” vs. actual. We don’t require that our agent be able to predict the future, or predict unlikely events.

Information gathering might also be a rational action.
  ▲ Crossing the street without looking is irrational

Rational agents must be able to learn (except in very simple, well-understood environments).
  ▲ Learning is defined as improving an agent’s performance.
  ▲ This could mean reducing uncertainty, or taking observations into account.
What makes an agent?
Defining an environment
Types of agent programs
A criterion for being an agent is existing in an environment.

Not necessarily physical - could be a software environment.

R & N refer to the *task environment*

Consists of:

- Performance measure
- Environment
- Actuators available to the agent
- Sensors available to the agent
2-17: Characteristics of the Environment

- Observability
- Deterministic/stochastic
- Episodic vs sequential
- Static vs Dynamic
- Discrete vs continuous
- Single-agent vs multi-agent
The environment is fully observable if the agent’s sensors always give it complete information about the relevant parts of the environment.

Is there anything the agent needs to know that it cannot sense?

Chess-playing: fully observable.

Vacuum cleaner: partially observable (can’t see if there’s dirt in the adjacent square)
We can think of the world as going through state transitions.

\[ \text{CurrentState} \times \text{agentActions} \rightarrow \text{newState} \]

If the state transition is unique, the world is deterministic.

Does an action always produce the same result?

- chess-playing: deterministic
- Vacuum world: deterministic
- Driving a car: stochastic
2-20: Deterministic/stochastic

- Note: we’re avoiding splitting hairs with quantum-mechanical questions here - it’s possible that the world could be deterministic, but appear stochastic due to its complexity.

- How does the world appear to the agent.
Episodic vs Sequential

Episodic: each action is independent.
- Agent perceives, decides, acts. Start again.
- Next decision does not depend on previous states.
- A spam-filtering agent is episodic.

If the agent must take a series of actions to accomplish a task or achieve a goal, the environment is sequential.
- Future decisions must be considered.
- Driving a car is sequential.
A static environment “holds still” while the agent is deciding on an action.

Agent is not under time pressure to come to a decision.
- Spam-filtering is static.
- Chess-playing is static.

A dynamic environment changes while the agent is deciding what action to take.

Harder: the agent must act “quickly enough”
- Driving a car is dynamic
Semidynamic: the environment doesn’t change, but the performance measure changes over time.

- Taking a timed test
- Playing chess with a clock.

Still pressure to act quickly.
Discrete vs. Continuous

We can talk about discrete vs continuous in terms of the agent’s percepts, actions, or possible states of the environment.

If the possible values for any of these are a discrete set, then the environment is discrete wrt that characteristic.

- Discrete is not the same as finite.
- A spam-filtering environment is discrete, even though there’s a (countably) infinite number of possible emails.
A continuous environment has continuously-changing variables.
- Steering angles in a car-driving environment.
- Real-valued sensor readings. (we can split hairs about precision here; the point is whether or not there’s a distinguishable change between two values)

Time is the element we’ll often be concerned with.
Single-agent. Our agent is acting on its own.
  △ World may still be stochastic
  △ Spam-filtering is single-agent.

Multi-agent: The actions/goals/strategies of other agents must be taken into account.
  △ Chess-playing, bidding in an auction

Issues
  △ Even though a world may have other agents, we may choose to treat it as single-agent and stochastic for complexity reasons.
    e.g. an agent controlling traffic signals.
  △ Cooperative vs. Competitive
2-27: Some examples

- Chess-playing, Monopoly-playing, slot-machine playing
- Robot getting me coffee in Harney
- Mars orbiter
- Web-crawling agent
- Conversational agent
- Medical diagnosis agent
What makes an agent?
Defining an environment
Types of agent programs
Typically, we can’t enumerate every possible percept sequence and action.

- Too many possibilities.
- We as programmers may not know what to do for each sequence.
- We may not understand the problem well enough.

Need to create a more compact representation.
2-30: Types of agent programs

- Table-driven agent
- Reflex agent
- Model-based reflex agent
- Goal-based agent
- Utility-based agent
- Learning agent
Keep a dictionary that maps percept sequences to actions.

```python
class TableDrivenAgent(Agent):
    """This agent selects an action based on the percept sequence.
    It is practical only for tiny domains.
    To customize it you provide a table to the constructor. [Fig. 2.7]"

    def __init__(self, table):
        """Supply as table a dictionary of all {percept_sequence:action} pairs."
        # The agent program could in principle be a function, but because
        # it needs to store state, we make it a callable instance of a class
        percepts = []
        def program(percept):
            percepts.append(percept)
            action = table.get(tuple(percepts))
            return action
        self.program = program
```

2-31: Table-driven agent
2-32: Examples of table-driven “agents”

- Exception handlers
- Square root/logarithm tables
- Won’t scale to AI environments.
  - Chess: $10^{150}$ percept sequences.
  - And this is an “easy” environment. Deterministic, fully observable.
  - Taxi driving: $10^{250,000,000,000}$ entries for one hour of driving.
- Also: highly redundant and uninformative
  - Many percept sequences may lead to the same action.
  - Designer has no guidance as to how to fill in the table.
Given the current percept, select an action.

Ignore history.
Given the current percept, select an action.

class ReflexVacuumAgent(Agent):
    "A reflex agent for the two-state vacuum environment. [Fig. 2.8]"

    def __init__(self):
        def program((location, status)):
            if status == 'Dirty': return 'Suck'
            elif location == loc_A: return 'Right'
            elif location == loc_B: return 'Left'
        self.program = program

This agent will only be rational if the best action can be chosen based only on the current percepts.
Examples of reflex agents
- Thermostat agent
- Spam-filtering agent (sometimes)

Insight: table can be represented more compactly as a set of condition-action rules.

Problems:
- Writing rules that apply in all cases is hard.
- Often it’s necessary to remember some of the past.
A model-based reflex agent maintains an internal representation of the world.

Actions are selected based on the model and the current percepts.
A model-based reflex agent maintains an internal representation of the world.

Actions are selected based on the model and the current percepts.

class ModelBasedVacuumAgent(Agent):
    "An agent that keeps track of what locations are clean or dirty."
    def __init__(self):
        model = {loc_A: None, loc_B: None}
        def program((location, status)):
            "Same as ReflexVacuumAgent, except if everything is clean, do NoOp"
            model[location] = status ## Update the model here
            if model[loc_A] == model[loc_B] == 'Clean': return 'NoOp'
            elif status == 'Dirty': return 'Suck'
            elif location == loc_A: return 'Right'
            elif location == loc_B: return 'Left'
            self.program = program
Maintaining a representation of the environment is extremely useful.
- Allows the agent to remember things.
- Can anticipate future events.
- Can make predictions about unseen parts of the environment.

Still uses rules, conditioned on the model and the sensors.

Much of our time will be spent constructing and manipulating models.
Examples of model-based agents

- Vacuum-cleaner agent (with a map)
- Spam-filtering agent (maybe)
- Factory robots
2-40: Types of models

- Attributes and values
- Probability distributions over variables
- Data structures
  - Maps
  - Graphs
  - Finite State Machines
- Facts about the world
Knowing the current state of the environment is not always enough.

The right action may also depend upon what the agent is trying to accomplish.

Select actions that will help accomplish goals.

Search and planning are used to solve this problem.
Goal-based reasoning is very useful for sequential environments.

- Chess-playing
- Taxi driving
- Spaceship piloting

The right action to take for a given percept sequence depends upon the agent’s knowledge (its model), its current state (percepts) and what it is trying to achieve currently.

Next week’s lectures will look at using search to accomplish goals.
Goals may not be enough in high-complexity environments.

There may be many action sequences that achieve a goal.

*Utility* is used to compare the relative desirability of action sequences.

Maps states to real numbers.

Can be an estimate of cost, time, or relative value of different outcomes.

Utilities are very useful in dealing with partially observable or stochastic environments.
Utility-based agent

Agent

Environment

State

What the world is like now

What it will be like if I do action A

What the world is like now

How happy I will be in such a state

How my actions do

What action I should do now

Utility

How the world evolves

Sensors

Actuators
Utilities are sometimes a controversial issue in AI.

Assumption: outcomes can be linearly ordered (with ties) on a consistent scale.

- Example: Taxi driving agent. What is the utility of driving off the Bay Bridge?
- Requires designer to explicitly evaluate outcomes. (Qualitative vs quantitative)

Utilities are very useful in domains with probability and/or money.

- Online trading, exploration in uncertain environments, gambling.
Often, an agent may need to update its agent program.
Programmers may not completely understand the environment.
The environment may change over time.
Coding by hand may be tedious.
A learning agent is one that improves its performance wrt a set of tasks over time.
Learning (or adaptation) is essential in complex environments.
A learning agent needs both a *performance element* and a *learning element*

- Performance element: Select current action(s)
- Learning element: evaluate the correctness of the performance element.
2-48: Learning Agent

Performance standard

Critic

feedback

Sensors

changes

Learning element

knowledge

Performance element

learning goals

Problem generator

Environment

Agent

Actuators
Learning can happen *offline* or *online*.

Learning can be *passive* or *active*.

Learning can be *supervised* or *unsupervised*.

*Credit assignment* is a big problem when learning in sequential environments.

Often, learning is treated as a separate topic in AI; we’ll try to integrate it in with other topics.
An agent is an *autonomous* program situated in an *environment*.

An agent behaves *rationally* if it acts to optimize its expected *performance measure*.

Characterizing the environment can help us decide how to build an agent.

More complex environments often require more sophisticated agent programs.
We get into our first “meaty” topic: search.

We’ll look at how agents can perform goal-directed behavior by searching through a space of possible outcomes.

Uninformed search.

Applying domain knowledge - heuristic search.

Game-tree search (adversarial search)

Searching enormous spaces: genetic algorithms and simulated annealing.