

Artificial Intelligence Programming

Agents and Environments

Chris Brooks

Department of Computer Science
University of San Francisco

Overview

- What makes an agent?
- Defining an environment
- Types of agent programs

Overview

- What makes an agent?
- Defining an environment
- Types of agent programs

Department of Computer Science — University of San Francisco — p.1/77

Department of Computer Science — University of San Francisco — p.1/77

Intelligent Agents

- The idea of developing *intelligent agents* has been a unifying one for AI.
 - Previously, subdisciplines were fairly balkanized.
 - Learning, knowledge representation, search, vision, etc.
- Agent programs that may require several of these abilities.

Department of Computer Science — University of San Francisco — p.3/77

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 *situated in an environment* and is capable of *autonomous action*.

Department of Computer Science — University of San Francisco — p.4/77

Qualities of an agent

- Autonomy
- Adaptation
- Goal-directed behavior
- Has “beliefs” and “intentions”
- Proactive
- Situated within an environment

Department of Computer Science — University of San Francisco — p.5/77

Autonomy

- 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.
 - When might we not want an agent to have complete autonomy?
- Challenge: Designing an agent that can reason about its own autonomy and know when to ask for help.

Department of Computer Science — University of San Francisco — p.6/77

Agent-oriented Programming

- We can also think of agents as a programming paradigm.
 - 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

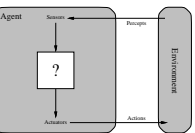
Department of Computer Science — University of San Francisco — p.7/77

Usefulness of the agent metaphor

- Why bother with all this? We already know how to write programs.
- Agents tend to be *open-ended* programs
 - Difficult to specify all cases in advance.
 - Instead, write programs that can work with a wide set of cases.
 - Separate out the knowledge from the reasoning mechanism.
- 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*.

Department of Computer Science — University of San Francisco — p.8/77

Agents and Environments



- One shift from other courses: we'll think explicitly about an agent's *environment* and how that affects execution.
- 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)

Department of Computer Science — University of San Francisco — p.9/77

Agent Programs

- We can describe our agent's behavior as a function F :
 - Action = $F(\text{current-percept}, \text{percept-history})$.
 - 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.

Department of Computer Science — University of San Francisco — p.10/77

Example: Vacuum-cleaner World

- Robotic vacuum cleaners are actually on the market.

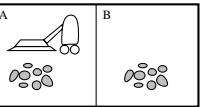


- \$150 at Amazon
- A *reflex* agent (mostly)

Department of Computer Science — University of San Francisco — p.11/77

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.

Example: Vacuum-cleaner World

- 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.

Rationality

- 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.

Rationality

- 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.

Rationality

- “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.

Overview

- What makes an agent?
- Defining an environment
- Types of agent programs

Environments

- 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

Characteristics of the Environment

- Observability
- Deterministic/stochastic
- Episodic vs sequential
- Static vs Dynamic
- Discrete vs continuous
- Single-agent vs multi-agent

Observability

- 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.
 - What about the other player?
- Vacuum cleaner: partially observable (can't see if there's dirt in the adjacent square)

Deterministic/stochastic

- We can think of the world as going through *state transitions*.
- $CurrentState \times agentActions \rightarrow newState$
- 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

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.

Static vs Dynamic

- 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

Static vs 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.

Discrete vs. Continuous

- 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 vs. Multiagent

- 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

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

Overview

- What makes an agent?
- Defining an environment
- Types of agent programs

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.

Types of agent programs

- Table-driven agent
- Reflex agent
- Model-based reflex agent
- Goal-based agent
- Utility-based agent
- Learning agent

Table-driven agent

- Keep a dictionary that maps percept sequences to actions.

```
class TableDrivenAgent(Agent):
    """This agent selects an action based on the percept sequence.
    It is practical only for tiny domains."""

    def __init__(self, table):
        #Supply as table a dictionary of all {perceptSequence:action} pairs.
        self.percepts = []

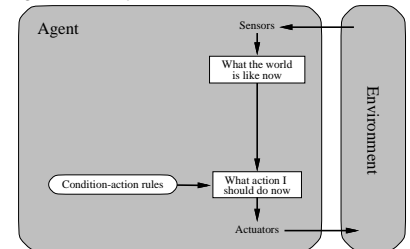
    def program(percept):
        self.percepts.append(percept)
        action = table.get(tuple(self.percepts))
        return action
```

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.

Reflex agent

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



Reflex agent

- Given the current percept, select an action.

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

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

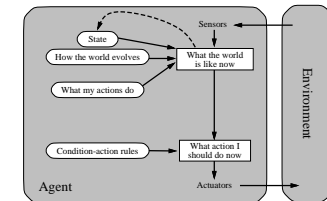
- This agent will only be rational if the best action can be chosen based only on the current percepts.

Reflex Agents

- 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.

Model-based reflex agent

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



Model-based reflex agent

- A model-based reflex agent maintains an internal representation of the world.
- We'll refer to this as keeping *state* about 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'
```

Model-based reflex agent

- 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.

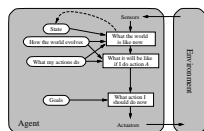
Model-based reflex agent

- Examples of model-based agents
 - Vacuum-cleaner agent (with a map)
 - Spam-filtering agent (maybe)
 - Factory robots

Types of models

- Attributes and values
- Probability distributions over variables
- Data structures
 - Maps
 - Graphs
 - Finite State Machines
- Facts about the world

Goal-based agent



- 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.

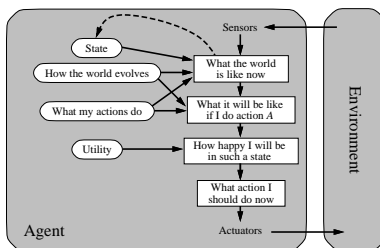
Goal-based agent

- 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*.
- Future lectures will look at using search to accomplish goals.

Utility-based agent

- 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



Utility-based agent

- 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.

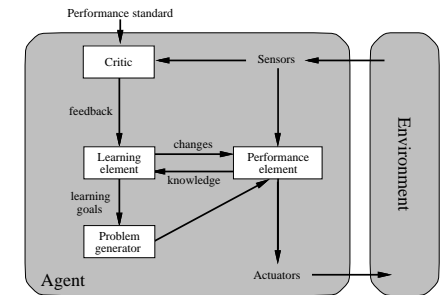
Learning Agent

- 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.

Learning Agent

- 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.

Learning Agent



Learning Agent

- 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.

Summary

- 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.

Next week(s) ...

- 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, constraint satisfaction
- Searching enormous spaces: genetic algorithms and simulated annealing.