Today: Intelligent Agents

Chapter 2

Outline

• Agents and environments
• Rationality
• PEAS (Performance measure, Environment, Actuators, Sensors)
• Environment types
• Agent types

Reading & Homework

• For today: read chapter 1-2
• For Monday: read chapter 3
• Homework: Due Wed, April 6th:
  – Book problems: 2.1, 2.3, 2.7, 2.8, 2.9
  – Turn in HWs in class or directly to our TA

Agents

• An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators
• Human agent:
  – sensors: eyes, ears, and other organs
  – actuators: hands, legs, mouth, and other body parts
• Robotic agent:
  – sensors: cameras and infrared range finders;
  – actuators: various motors and switches

Agents and environments

- The agent function maps from percept histories to actions:
  \[ f: \mathcal{P}^n \rightarrow \mathcal{A} \]
- The agent program runs on the physical architecture to produce \( f \)
- agent = architecture + program
- Q: Is this a realistic function to implement? What if at each time step, there were 100 possible percepts (which is quite small)? T times steps, 100^T possible histories.
Ex: Vacuum-cleaner world

- Percepts: location and contents, e.g., [A, Dirty]
- Actions: Left, Right, Suck, NoOp

A vacuum-cleaner agent

<table>
<thead>
<tr>
<th>Percept sequence</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, Clean</td>
<td>Right</td>
</tr>
<tr>
<td>A, Dirty</td>
<td>Suck</td>
</tr>
<tr>
<td>B, Clean</td>
<td>Left</td>
</tr>
<tr>
<td>B, Dirty</td>
<td>Suck</td>
</tr>
<tr>
<td>A, Clean, [A, Clean]</td>
<td>Right</td>
</tr>
<tr>
<td>A, Clean, [A, Dirty]</td>
<td>Suck</td>
</tr>
</tbody>
</table>

function REPALEA(Percept, Action)
returns an action
if location = Dirty then return Suck
else if location = A then return Right
else if location = B then return Left

- What is the right function? Can it be implemented in a small agent program?
- For this agent, are all histories necessary?

Rational agents

- An agent should strive to "do the right thing", based on what it can perceive and the actions it can perform. The right action is the one that will cause the agent to be most "successful" in its environment, where "successful" is itself a concept that is relative and needs explicit definition.
- Performance measure: An objective criterion for success of an agent's behavior
- E.g., performance measure of a vacuum-cleaner agent could be amount of dirt cleaned up, amount of time taken, amount of electricity consumed, amount of noise generated, etc.

Rational agents

- Rational Agent: 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.

Rational agents

- Rationality is distinct from omniscience (all-knowing with infinite knowledge)
- Agents can perform actions in order to modify future percepts so as to obtain useful information (information gathering, exploration)
- An agent is autonomous if its behavior is determined by its own experience (with ability to learn and adapt)

PEAS

- A way to set the "parameters" of an agent
  - what it must do, where it must do it, how it must do it, etc.
- PEAS:
  - Performance measure
  - Environment
  - Actuators
  - Sensors
- Must first specify the setting for intelligent agent design
PEAS: Example 1
• We consider a domain more realistic than the vacuum, for example:
  • Consider, e.g., the task of designing an automated taxi driver:
    – Performance measure: Safe, fast, legal, comfortable trip, maximize profits (of driver)
      • note some of these might be in conflict (e.g., fast vs. safe, fast vs. comfortable, legal vs. profit maximization, etc.)
    – Environment: Roads, other traffic, pedestrians, customers
    – Actuators: Steering wheel, accelerator, brake, signal, horn
    – Sensors: Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard

PEAS: Example 2
• Agent: Medical diagnosis system
• Performance measure: Healthy patient, minimize costs, lawsuits
• Environment: Patient, hospital, staff
• Actuators: Screen display (questions, tests, diagnoses, treatments, referrals)
• Sensors: Keyboard (entry of symptoms, findings, patient’s answers), perhaps microphone (for audio input)

PEAS: Example 3
• Agent: Part-picking robot
• Performance measure: Percentage of parts in correct bins
• Environment: Conveyor belt with parts, bins
• Actuators: Jointed arm and hand
• Sensors: Camera, joint angle sensors

PEAS: Example 4
• Agent: Interactive English tutor
• Performance measure: Maximize student’s score on test
• Environment: Set of students
• Actuators: Screen display (exercises, suggestions, corrections)
• Sensors: Keyboard

Environment types
• There are different ways to characterize what an agent’s environment will consist of.
• These “constraints” significantly affect how we implement an agent (i.e., we are being fairly abstract on purpose right now).
• In future chapters, we will see how different environment types will require potentially completely different implementations (e.g., search, temporal aspects, computational demands, etc.)
• Environment types also affect scalability. This is crucial, since in the 1950s AI researchers assumed that algorithms would scale up fairly easily (before computational complexity theory became well developed).
• Fully observable (vs. partially observable): An agent’s sensors give it access to the complete state of the environment at each point in time.
  • Deterministic (vs. stochastic): The next state of the environment is completely determined by the current state and the action executed by the agent. (If the environment is deterministic except for the actions of other agents, then the environment is strategic)
    – stochastic might result simply from being partially observable
    – do we live in a fully deterministic but partially observable world? Does ignorance cause intelligence?
• Episodic (vs. sequential): The agent’s experience is divided into atomic “episodes” (each episode consists of the agent perceiving and then performing a single action independently of other episodes), and the choice of action in each episode depends only on the episode itself.
  – This could also be called “independent” vs. “temporal”
Environment types

- **Static** (vs. dynamic): The environment is unchanged while an agent is deliberating. (The environment is **semidynamic** if the environment itself does not change with the passage of time but the agent's performance score does)
- **Discrete** (vs. continuous): A limited number of distinct, clearly defined percepts and actions (vs. a continuum of percepts, e.g., $R^3$)
- **Single agent** (vs. multiagent): An agent operating by itself in an environment.

Agent functions and programs

- An agent is completely specified by the agent function mapping percept sequences to actions
- One agent function (or a small equivalence class) is (ideally) rational
- Aim: find a way to implement the rational agent function concisely
- Note: This simple idea of an agent function is extremely general. Much of this class is about how to implement these agent functions. They can be deterministic, rule based, algorithmic, based on probability and statistics (recently popular), or something else. We next characterize agents rather abstractly...

Environment types: Examples

<table>
<thead>
<tr>
<th>Fully observable</th>
<th>Chess with a clock</th>
<th>Chess without a clock</th>
<th>Taxi driving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Deterministic</td>
<td>Strategic</td>
<td>Strategic</td>
<td>No</td>
</tr>
<tr>
<td>Episodic</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Static</td>
<td>Semi</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Discrete</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Single agent</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic (we think), sequential, dynamic, continuous, and multi-agent

How to implement agent function?

- Conceptually the most simple:
  - **Table-Driven Agent**
    - just a big table that maps from inputs (percept histories) to action outputs
  - Is this realistic? Can it be intelligence?
  - **John Searle’s Chinese Room argument**

Table-Driven Agent

- Practical Drawbacks:
  - Huge table (not possible to implement)
  - Take a long (or infinite) time to build the table
  - No autonomy (as specified, how to learn this or to adapt)
  - Even with learning, need a long time to learn the table entries (curse of dimensionality, table is to large a system)
- Key idea: Occam’s Razor: We want the simplest possible system that generalizes well (given a choice between two systems that perform equally on a given data set, we choose the simpler one)

Agent types

- Four basic types in order of increasing generality:
  - Simple reflex agents
  - Model-based reflex agents
  - Goal-based agents
  - Utility-based agents
Ex: Agent program for vacuum-cleaner agent

```plaintext
function REFLEX-VACUUM-AGENT([location, status]) returns an action
  if status = Dirty then return Suck
  else if location = A then return Right
  else if location = B then return Left
```

- Note: not necessary to look back at history in this simple world
- Table size is therefore very small, this agent (in this particular env.) will work well.
- We can make this even more concise by using rules (basically “if-then” clauses) based on some function of (only) the current percept.

General Simple reflex agents

- State is only function of current percept (no memory)
- select and apply the one rule that matches current state
- only works if (at least) environment is fully observable at all times (and deterministic)
- Select a “condition-action” (or if-then) rule, that checks a condition and executes the action if the condition is true. – just like in any programming language

Simple reflex agents

- State is only function of current percept (no memory)
- select and apply the one rule that matches current state
- only works if (at least) environment is fully observable at all times (and deterministic)
- Select a “condition-action” (or if-then) rule, that checks a condition and executes the action if the condition is true. – just like in any programming language

Model-based reflex agents

- State: summary of percept history
- How the word evolves: knowledge of how, independent of the agent, the world might change (and used to predict best action)
- What my actions do: model of how the agents actions will effect the world (and also guides action to take)
- Condition-action rules: if-then rules, same as before.

Model-based reflex agents

- Agent now has memory (since new state is function of most recent action and most recent state), but doesn’t store all past percepts.
- New state also uses current percept.
Goal-based agents

- Goal: agent keeps track of a (future) goal that it is trying to reach.
- Each action might not achieve that goal, but (hopefully) will step towards achieving that goal.
  - each step might require much “thought” to determine which state might best (at the moment) lead to the goal. This is the reason for search and planning in AI.
- Diagram shows that agent “deliberates” before it takes a given action.

Utility-based agents

- Goals are too quantized
- Utilities are often numeric (measurable) ways to judge how far we are towards achieving a particular goal.
- Utility function:
  - maps (sequence of) states to real numbers
  - Allows tradeoffs to be made between conflicting goals (quick vs. safe, rich vs. nice, etc.)
  - allows for dealing with uncertainty (reach goals with greater “likelihood”)
  - Have probabilistic interpretations (as do everything)

Learning agents

- So far, agents are static, they don’t change in response to how they perform in their environment.
- Intelligence typically requires that there be some learning associated with it.
  - ex: dung beetle, if ball of dung is removed from grasp en route, beetle continues on pantomiming filling its nest (same plan, as if the dung ball sensor is turned off once the plan is in action)
  - ex: spex wasp: plan: burrow, sing caterpillar, drag to burrow, enter to check ok, drag caterpillar inside.
    - if caterpillar is moved during check, wasp will “reset” plan to drag stage of plan regardless of number of times it happens.

- So, perhaps even “living” things are not intelligent?
- Ideally, a real agent will learn (as hypothesized by Turing).
- Are humans intelligent enough to directly “code up” intelligent behavior?
- If not, are we intelligent enough to code up a machine that can learn intelligent behavior (even if once it does, we don’t understand why it does what it does)?
- We can represent general situation using agents
Learning agents

- Performance element: agent from before
- Learning element: changes the performance element via knowledge obtained.
- Critic: outside “teacher” of agent, that tells it how well it is doing.
  - Supervisory information: critic explicitly tells that it is doing well or poorly (relative to a performance standard)
  - Other learning methods have no supervisory info.
- Problem generator: suggest actions leading to new/informative experiences (even if they are locally or temporarily suboptimal)

Summary Agents

- You might think, what’s the difference between agents and any computer program (which has input, control, and output)
- We abstract these different types of agents since these characterize many situations in the real world, and also since these characterizations (as we will see in later weeks) greatly effect the possible set of solutions (implementations) of a given agent (some of which will be computationally easy, others of which will be computationally intractable).