Intelligent Agents

CHAPTER 2

Outline

- PAGE (Percepts, Actions, Goals, Environment)
- Environment types
- Agent functions and programs
- Agent types
- Vacuum world
PAGE

Must first specify the setting for intelligent agent design

Consider, e.g., the task of designing an automated taxi:

Percepts??

Actions??

Goals??

Environment??

PAGE

Must first specify the setting for intelligent agent design

Consider, e.g., the task of designing an automated taxi:

Percepts?? video, accelerometers, gauges, engine sensors, keyboard, GPS, ...

Actions?? steer, accelerate, brake, horn, speak/display, ...

Goals?? safety, reach destination, maximize profits, obey laws, passenger comfort, ...

Environment?? US urban streets, freeways, traffic, pedestrians, weather, customers, ....
**Internet shopping agent**

- Percepts??
- Actions??
- Goals??
- Environment??

---

**Rational agents**

Without loss of generality, "goals" specifiable by performance measure defining a numerical value for any environment history

Rational action: whichever action maximizes the expected value of the performance measure given the percept sequence to date

Rational ≠ omniscient
Rational ≠ clairvoyant
Rational ≠ successful
### Environment types

<table>
<thead>
<tr>
<th>Accessible??</th>
<th>Solitaire</th>
<th>Backgammon</th>
<th>Internet shopping</th>
<th>Taxi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic??</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Episodic??</td>
<td>Yes</td>
<td>No</td>
<td>Partly</td>
<td>No</td>
</tr>
<tr>
<td>Static??</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Discrete??</td>
<td>Yes</td>
<td>Semi</td>
<td>Semi</td>
<td>No</td>
</tr>
</tbody>
</table>

The environment type largely determines the agent design

The real world is (of course) inaccessible, stochastic, sequential, dynamic, continuous
Agent functions and programs

An agent is completely specified by the agent function mapping percept sequences to actions.

(In principle, one can supply each possible sequence to see what it does. Obviously, a lookup table would usually be immense.)

One agent function (or a small equivalence class) is rational.

Aim: find a way to implement the rational agent function concisely.

An agent program takes a single percept as input, keeps internal state:

```lisp
(defun SKELETON-AGENT(percep) returns action
  state: memory, the agent's memory of the world
  memory <- UPDATE-MEMORY(memory, percep)
  action <- CHOOSE-BEST-ACTION(memory)
  memory <- UPDATE-MEMORY(memory, action)
  return action
```

AIMA code

The code for each topic is divided into four directories:
- agents: code defining agent types and programs
- algorithms: code for the methods used by the agent programs
- environments: code defining environment types, simulations
- domains: problem types and instances for input to algorithms

(Often run algorithms on domains rather than agents in environments.)

```lisp
(setq joe (make-agent :name 'joe :body (make-agent-body)
                      :program (make-dumb-agent-program)))

(defun make-dumb-agent-program ()
  (let ((memory nil))
    #'(lambda (percept)
       (push percept memory)
       'no-op)))
```
Agent types

Four basic types in order of increasing generality:
- simple reflex agents
- reflex agents with state
- goal-based agents
- utility-based agents
Simple reflex agents

function SIMPLE-REFLEX-AGENT(percept) returns action
static: rules, a set of condition-action rules
state ← INTERPRET-INPUT(percept)
rule ← RULE-MATCH(state, rules)
action ← RULE-ACTION[rule]
return action

Reflex agents with state

State
How the world evolves
What the world is like now
What my actions do
Condition-action rules
What action I should do now

Agent
Sensors
Environment

Effectors
Reflex agents with state

```plaintext
function REFLEX-AGENT-WITH-STATE (percept) returns action

static: state, a description of the current world state
rules, a set of condition-action rules

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

Goal-based agents

The agent receives inputs from the environment through sensors and interacts with the environment through effectors. The agent maintains a state that describes the current condition of the world, including what the world is like now and how it will be like if the agent takes certain actions. The agent also has goals that guide its actions. The decision-making process involves evaluating what actions the agent should take based on the current state and goals.
**Utility-based agents**

- **State**
- How the world evolves
- What the world is like now
- What my actions do
- What it will be like if I do action A
- Utility
  - How happy I will be in such a state
  - What action I should do now

**Environment**

**Agent**

**The vacuum world**

code/agents/environments/vacuum.lisp

**Percepts**

- `<bump>`
- `<dirt>`
- `<home>`

**Actions**

- shutoff
- forward
- suck
- (turn left)
- (turn right)

**Goals**

- (performance measure on environment history)
  - +100 for each piece of dirt cleaned up
  - -1 for each action
  - -1000 for shutting off away from home

**Environment**

- grid, walls/obstacles, dirt distribution and creation, agent body
- movement actions work unless bump into wall
- suck actions put dirt into agent body (or not)

**Accessible?** Deterministic? Episodic? Static? Discrete?