Intelligent Agents

Chapter 2

Reminders

Assignment 0 (lisp refresher) due 1/27

Lisp/emacs/AIMA tutorial: Friday 1–3 and Monday 10–12, 271 Soda

Outline

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

Agents and environments

Agents include humans, robots, softbots, thermostats, etc.
The agent function maps from percept histories to actions:

\[ f : \mathcal{P}^* \rightarrow A \]

The agent program runs on the physical architecture to produce \( f \)

Vacuum-cleaner world

Percepts: location and contents, e.g., \([A, \text{Dirty}]\)
Actions: \(\text{Left, Right, Suck, NoOp}\)

A vacuum-cleaner agent

<table>
<thead>
<tr>
<th>Percept sequence</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>([A, \text{Clean}])</td>
<td>\text{Right}</td>
</tr>
<tr>
<td>([A, \text{Dirty}])</td>
<td>\text{Suck}</td>
</tr>
<tr>
<td>([B, \text{Clean}])</td>
<td>\text{Left}</td>
</tr>
<tr>
<td>([B, \text{Dirty}])</td>
<td>\text{Suck}</td>
</tr>
<tr>
<td>([A, \text{Clean}], [A, \text{Clean}])</td>
<td>\text{Right}</td>
</tr>
<tr>
<td>([A, \text{Clean}], [A, \text{Dirty}])</td>
<td>\text{Suck}</td>
</tr>
</tbody>
</table>

\textbf{function} \text{REFLEX-VACUUM-AGENT}(\text{location, status}) \textbf{returns} an action

\begin{align*}
\text{if} \; \text{status} = \text{Dirty} \; \text{then return} \; \text{Suck} \\
\text{else if} \; \text{location} = A \; \text{then return} \; \text{Right} \\
\text{else if} \; \text{location} = B \; \text{then return} \; \text{Left} 
\end{align*}

What is the \textbf{right} function?
Can it be implemented in a small agent program?
Rationality

Fixed performance measure evaluates the environment sequence
– one point per square cleaned up in time $T$?
– one point per clean square per time step, minus one per move?
– penalize for $> k$ dirty squares?

A rational agent chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date.

Rational $\neq$ omniscient
– percepts may not supply all relevant information
Rational $\neq$ clairvoyant
– action outcomes may not be as expected
Hence, rational $\neq$ successful
Rational $\Rightarrow$ exploration, learning, autonomy

Internet shopping agent

Performance measure??
Environment??
Actuators??
Sensors??

PEAS

To design a rational agent, we must specify the task environment.

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

Performance measure??
Environment??
Actuators??
Sensors??

Environment types

<table>
<thead>
<tr>
<th>Solitaire</th>
<th>Backgammon</th>
<th>Internet shopping</th>
<th>Taxi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observable??</td>
<td>Deterministic??</td>
<td>Episodic??</td>
<td>Static??</td>
</tr>
<tr>
<td>Discrete??</td>
<td>Single-agent??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Environment types</td>
<td>Solitaire</td>
<td>Backgammon</td>
<td>Internet shopping</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Observable??</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Deterministic??</td>
<td>Yes</td>
<td>No</td>
<td>Partly</td>
</tr>
<tr>
<td>Episodic??</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Static??</td>
<td>Yes</td>
<td>Semi</td>
<td>Semi</td>
</tr>
<tr>
<td>Discrete??</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes (except auctions)</td>
</tr>
<tr>
<td>Single-agent??</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The environment type largely determines the agent design.

The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent.
Agent types

Four basic types in order of increasing generality:

- simple reflex agents
- reflex agents with state
- goal-based agents
- utility-based agents

All these can be turned into learning agents.

Example

```lisp
(defun make-reflex-vacuum-agent-program ()
  #'(lambda (percept)
     (let ((location (first percept)) (status (second percept)))
       (cond ((eq status 'dirty) 'Suck)
             ((eq location 'A) (if (> last-B 3) 'Right 'NoOp))
             ((eq location 'B) (if (> last-A 3) 'Left 'NoOp))))))
```

Goal-based agents
Utility-based agents


Several basic agent architectures exist: reflex, reflex with state, goal-based, utility-based.