Environments, Problem-Solving Agents

CS156

Chris Pollett

August 27, 2014

Outline

- Environments for Agents
- Types of Agents Programs
- Problem Solving Agents

Introduction

- On Monday, we gave four possible answers to what is AI, one of which was acting rationally (like a rational agent).
- We said this is the approach we were going to consider first this semester.
- We then introduced the notion of agent which we said is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
- We gave a vacuum cleaner example of an agent.
- Finally, we gave a definition of what it means for an agent to be rational: For each possible percept sequence, the rational agent selects an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge it has.
- Today, we are going to begin by looking a the different kinds of environments that an agent might exist within, look at different kinds of agents, and then start looking at problem solving agents in detail.

Environments for Agents

- When we described the vacuum cleaner world, we had to specify the performance measure (P), the environment (E), the agent's actuators (A), and the sensors (S).
- We group these under the heading of the **task environment** and call the above the **PEAS** description.
- In designing an agent, the first task is always to define the task environment as completely as possible.
- As an example, consider the task of making an automated taxi driver:
  - ***Performance*** *-- safe, fast, legal, comfortable trip, maximize profits*
  - ***Environment*** *-- roads, other traffic, pedestrians, customers*
  - ***Actuators*** *-- steering, accelerator, brake, signal, horn, display*
  - ***Sensors*** *-- cameras, sonar, GPS, odometer, accelerometer, engine sensors, keyboard*

Properties of Task Environments

- **Fully observable vs partially observable** -- whether one can see the whole environment at any given step (such as in chess) or just part of it (such as Taxi case). In the latter case, the agent needs to keep track of what it has seen of the environment
- **Single agent versus multiagent** -- if you have an agent when does it have to treat other objects in its environment as agents? Cross-word puzzle solving is single agent, so was our vacuum environment, chess is a **competitive** two agent environment, taxi-driving is a **co-operative** multiagent environment. **Communication** is often involved in a multi-agent environment, as is **randomized behavior** to avoid predictability.
- **Deterministic versus stochastic** -- if the next state is completely determined by the current state and the given action then it is said to be deterministic; otherwise, it is stochastic.
- **Episodic versus sequential** -- in an episodic environment, an agent's experience is divided into episodes. In each episode the agents receives a percept and then performs a single action. For example, a quality control robot might need to classify a bottle as okay or defective. This does not depend on previous bottles it has looked at. Sequential environments are ones where the agents will affect what its future actions can be. For example, chess or taxi-driving.
- **Static versus dynamic** -- Can the environment change while the agent is deciding what to do next? If so, the environment is dynamic; otherwise, it is static. For example, chess is static, but taxi driving is dynamic.
- **Discrete versus continuous** -- Are the time, percepts, and actions of the agent divided into a fixed finite number of distinct values? (discrete) Or is it better to think of these values as being floating point of arbitrary precision? (continuous)
- **Known versus unknown** -- In a known environment, the outcomes for all actions are given. For example, the game of solitaire is a known environment which is only partially observable. If the environment is unknown, the agent has to figure out how it works in order to make good decisions. For example, a new video game for a human agent may present an unknown environment which is fully observable.

Agent Programs

- The goal of AI is to design agent programs. These implement the agent function described last day.
- The kind of computing device (with sensors and actuators) on which these our program runs is called the **architecture** of the agent.
- A complete agent = architecture + program.
- The program for table agent we discussed last day might be written in pseudo-code as:

```
function TABLE-DRIVEN-AGENT (percept) returns an action
    persistent: percepts, a sequence, initially empty;
                #can think of this as a static var
                table, a table of actions,
                   indexed by percept sequences;
                #think of as a const
    append percept to the end of percepts;
    action := Lookup(percepts, table);
    return action;
```

- Usually, the table driven approach is not used because the size of the table is huge for many task environments
- For example, suppose we had an automated taxi with a 640 x 480 px camera taking 30 frames a second. Each pixel has 24 bits of color information. This gives a lookup table of size / . $^{03.\,*...\,*...\,*...}$ for an hour's driving.

Kinds of Agent Programs

The key challenge of AI is to find out how to write programs that produce rational behavior from a smallish program rather than a vast table. We next look at four kinds of agent programs that embody most intelligent systems.

- **Simple Reflex Agent** -- these kinds of agents select actions on the basis of the current percept, ignoring the rest of the percept history. They are often defined by a sequence of **condition-action rules** (aka if then rules or **productions**). For example,

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

- **Model-based Reflex Agent** -- This kind of agent uses **internal states** to keep track of the parts of the world it can't see now. It updates its internal state based on information about how the world evolves independent of the agent and based on how the agent's actions affect the world. The former is called the **model** of the agent. After updating its state, the agent finds a rule which matches the state it is in and does the rules action. Some kinds of model-based agents can be implemented with finite-automata.
- **Goal-based Agents** -- Sometimes the current state of the environment is not enough to decide what to do next. One also needs to know a target **goal**. i.e, where do we want the Taxi to go to? **Search** and **planning** algorithms might be involved in goal-based agents.
- **Utility-based Agents** -- A goal is a binary condition on a state: For the taxi, we are either where we wanted to go or not. You could also imagine agents that keep track of there performance by applying a **utility function**. The agent then wants to move in ways that will increase the expected utility of its future states.

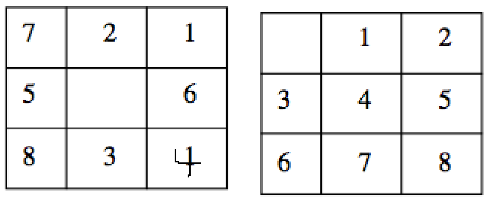
Problem Solving Agents

- We are now going to look at a particular kind of goal-based agent, called a **problem-solving agent**, in more detail.
- Problem solving agents use **atomic** representations for the environment, that is, states of the world are considered as whole, with no internal structure to the problem solving algorithm.
- Goal-based agents that use more advanced **factored** or **structured** representations are called **planning agents**.
- Basically, problem solving agents consider different sequences of actions until they find one that achieves the desired goal. For example, one might have a game and the goal of a monster in the game is to get from where it is to where the trapped player is. However, there are obstacles in between the monster can't go through. The monster might consider several potential paths until it finds one that works.
- The process of looking over sequences of actions is called **search**.
- A **solution** is an action sequence that achieves the goal.
- To discuss problem solving we want to make precise the notion of **problems** and their **solutions**.

A Problem

- An **initial state**.
- A set of **possible actions**
- A **successor function** which maps an action and a state to a new state. (state space)
- A **goal test** (to check if problem is solved)
- A **cost** associated with performing a given action. This cost of a single action induces an associated with a path in the state space. (A **path** is a sequence of (action, states))

Examples -- 8 Puzzle



7	2	1
5		6
8	3	4

	1	2
3	4	5
6	7	8

- States -- description of the location of each of 8 tiles.
- The initial state -- the position we want to solve from
- The successor function -- says how the hole can be moved.
- Goal Test -- Does the board look like (final solution)
- Path Cost- 1 for each move of hole in path.

Examples -- 8-Queens Problem

- States -- any arrangement of 0 to 8 queens on a chess board.
- Initial State -- no queens on the board.
- Successor function -- add a queen to an empty square.
- Goal Test -- 8 queens on the board none attacking each other.

Above state space could be as large as 64 x 63 x 62 x 61 x 60 x 59 x 58 x 57

Can reduce search space by requiring newly placed queens not to be attack-able by existing queens. (2,057 possible)