CS 188: Artificial Intelligence
Spring 2006

Lecture 2: Agents
1/19/2006

Dan Klein – UC Berkeley
Many slides from either Stuart Russell or Andrew Moore

Administrivia

- Reminder:
  - Drop-in Python/Unix lab
  - Friday 1-4pm, 275 Soda Hall
  - Optional, but recommended
- Accommodation issues
- Project 0 will be up by the weekend
- Newsgroup: ucb.class.cs188 (link from course page)
- Course workload curve

Today

- Agents and Environments
- Reflex Agents
- Environment Types
- Problem-Solving Agents

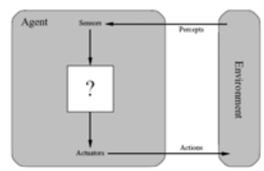
Agents and Environments

- Agents include:
  - Humans
  - Robots
  - Softbots
  - Thermostats
  - …
- The agent function maps from percept histories to actions:

$$\mathcal{P}^* \to \mathcal{A}$$

- An agent program running on the physical architecture to produces the agent function.

The line between agent and environment depends on the level of abstraction.

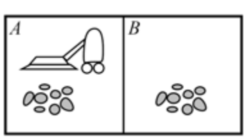


Always think of the environment as a black box, completely external to the agent – even if it's simulated by local code.

Vacuum-Cleaner World

- We'll start with a VERY simple world…

Vacuum World!



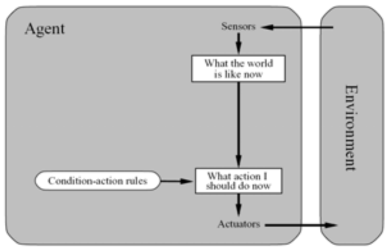
- Percepts: location and contents, e.g., [A, Dirty]
- Actions: Left, Right, Suck, No p

A Reflex Vacuum-Cleaner

```
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
```

Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Dirty]$	$Suck$
⋮	⋮

Simple Reflex Agents



- Does this ever make sense as a design?

Table-Lookup Agents?

- Complete map from percept (histories) to actions

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
⋮	⋮

- Drawbacks:
  - Huge table!
  - No autonomy
  - Even with learning, need a long time to learn the table entries
- How would you build a spam filter agent?
- Most agent programs produce complex behaviors from compact specifications

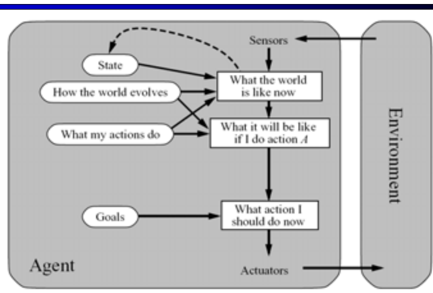
Rationality

- A 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?
- Reward should indicate success, not steps to success
- A rational agent chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date
- Rational ≠ omniscient: percepts may not supply all information
- Rational ≠ clairvoyant: action outcomes may not be as expected
- Hence, rational ≠ successful

Rationality and Goals

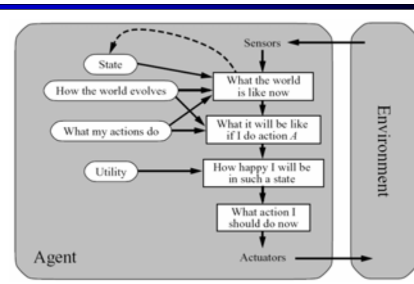
- Let's say we have a game:
  - Flip a biased coin (probability of heads is h)
  - Tails = loose $1
  - Heads = win $1
- What is the expected winnings?
  - (1)(h) + (-1)(1-h) = 2h - 1
- Rational to play?
  - What if performance measure is total money?
  - What if performance measure is spending rate?
  - Why might a human play this game at expected loss?

Goal-Based Agents



- These agents usually first find plans then execute them.

Utility-Based Agents



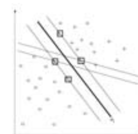
- How is this different from a goal-based agent?

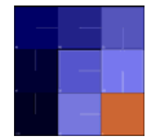
More Rationality

- Remember: rationality depends on:
  - Performance measure
  - Agent's (prior) knowledge
  - Agent's percepts to date
  - Available actions
- Is it rational to inspect the street before crossing?
- Is it rational to try new things?
- Is it rational to update beliefs?
- Is it rational to construct conditional plans in advance?
- Rationality gives rise to: exploration, learning, autonomy

The Road Not (Yet) Taken

- At this point we could go directly into:
  - Empirical risk minimization *(statistical classification)*
  - Expected return maximization *(reinforcement learning)*
- These are mathematical approaches that let us derive algorithms for rational action for reflex agents under nasty, realistic, uncertain conditions
- But we'll have to wait until week 5, when we have enough probability to work it all through
- Instead, we'll first consider more general goal-based agents, but under nice, deterministic conditions





PEAS: Automated Taxi

- Before designing an agent, we must specify the task
  - We've done this informally so far…
- Consider, e.g., the task of designing an automated taxi:
  - Performance measure: safety, destination, profits, legality, comfort…
  - Environment: US streets/freeways, traffic, pedestrians, weather…
  - Actuators: steering, accelerator, brake, horn, speaker/display…
  - Sensors: video, accelerometers, gauges, engine sensors, keyboard, GPS…

PEAS: Internet Shopping Agent

- Specifications:
  - Performance measure: price, quality, appropriateness, efficiency
  - Environment: current and future WWW sites, vendors, shippers
  - Actuators: display to user, follow URL, fill in form
  - Sensors: HTML pages (text, graphics, scripts)

PEAS: Spam Filtering Agent

- Specifications:
  - Performance measure: spam block, false positives, false negatives
  - Environment: email client or server
  - Actuators: mark as spam, transfer messages
  - Sensors: emails (possibly across users), traffic, etc.

Environment Simplifications

- 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.
- Episodic (vs. sequential): The agent's experience is divided into independent atomic "episodes" (each episode consists of the agent perceiving and then performing a single action)

Environment Simplifications

- Static (vs. dynamic): The environment is unchanged while an agent is deliberating.
- Discrete (vs. continuous): A limited number of distinct, clearly defined percepts and actions.
- Single agent (vs. multi- agent): An agent operating by itself in an environment.
- What's the real world like?

Environment Types

	Peg Solitaire	Back-gammon	Internet Shopping	Taxi
Observable	✓	✓	✗	✗
Deterministic	✓	✗	?	✗
Episodic	✗	✗	✗	✗
Static	✓	✓	?	✗
Discrete	✓	✓	✓	✗
Single-Agent	✓	✗	✓	✗

- The environment type largely determines the agent design
- The real world is partially observable, stochastic, sequential, dynamic, continuous, multi-agent

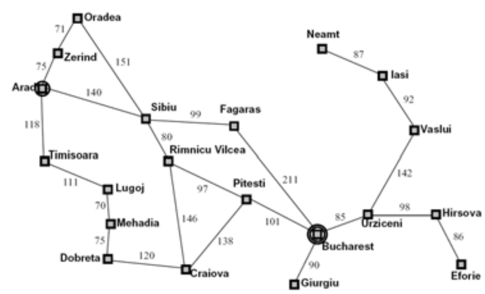
Problem-Solving Agents

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
   static: seq, an action sequence, initially empty
           state, some description of the current world state
           goal, a goal, initially null
           problem, a problem formulation

   state ← UPDATE-STATE(state, percept)
   if seq is empty then
       goal ← FORMULATE-GOAL(state)
       problem ← FORMULATE-PROBLEM(state, goal)
       seq ← SEARCH(problem)        ← This is the hard part!
   action ← FIRST(seq); seq ← REST(seq)
   return action
```

- This offline problem solving!
- Solution is executed "eyes closed."
- When will offline solutions work? Fail?

Example: Romania



Example: Romania

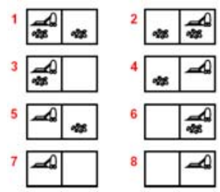
- Setup
  - On vacation in Romania; currently in Arad
  - Flight leaves tomorrow from Bucharest
- Formulate problem:
  - States: being in various cities
  - Actions: drive between adjacent cities
- Define goal:
  - Being in Bucharest
- Find a solution:
  - Sequence of actions, e.g. [Arad → Sibiu, Sibiu → Fagaras, …]

Problem Types

- Deterministic, fully observable → single-state problem
  - Agent knows exactly which state it will be in; solution is a sequence, can solve offline using model of environment
- Non-observable → sensorless problem (conformant problem)
  - Agent may have no idea where it is; solution is a sequence
- Nondeterministic and/or partially observable → contingency problem
  - Percepts provide new information about current state
  - Often first priority is gathering information or coercing environment
  - Often interleave search, execution
  - Cannot solve offline
- Unknown state space → exploration problem

Example: Vacuum World

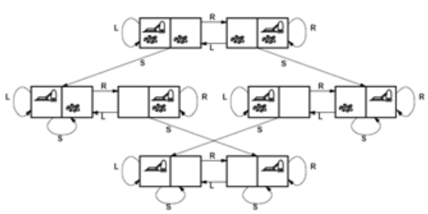
- States?
- Goal?
- Single State: Start in 5.
  - Solution?
  - [Right, Suck]
- Sensorless: Start in {1…8}
  - Solution?
  - [Right, Suck, Left, Suck]



Single State Problems

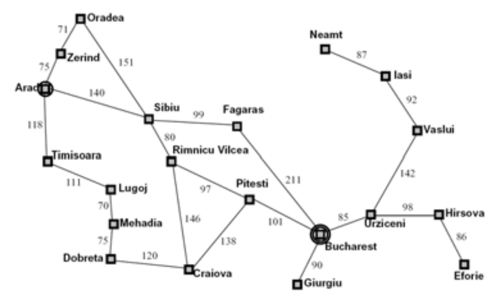
- A search problem is defined by four items:
  - Initial state: e.g. Arad
  - Successor function S(x) = set of action–state pairs:
    e.g., S(Arad) = {<Arad → Zerind, Zerind>, … }
  - Goal test, can be
    - explicit, e.g., x = Bucharest
    - implicit, e.g., Checkmate(x)
  - Path cost (additive)
    - e.g., sum of distances, number of actions executed, etc.
    - c(x,a,y) is the step cost, assumed to be ≥ 0
- A solution is a sequence of actions leading from the initial state to a goal state
- Problem formulations are almost always abstractions and simplifications

Example: Vacuum World

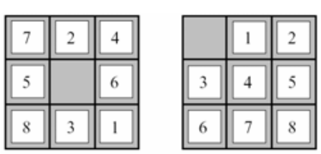


- Can represent problem as a graph
  - Nodes are states
  - Arcs are actions

Example: Romania



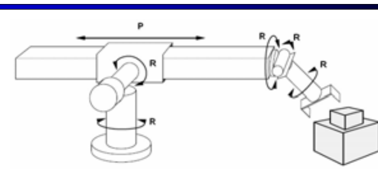
Example: 8-Puzzle



Start State Goal State

- What are the states?
- What are the actions?
- What states can I reach from the start state?
- What should the costs be?

Example: Assembly



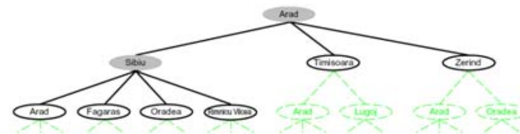
- What are the states?
- What is the goal?
- What are the actions?
- What should the costs be?

Tree Search

- Basic solution method for graph problems
  - Offline simulated exploration of state space
  - Searching a model of the space, not the real world

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
```

Tree Search Example



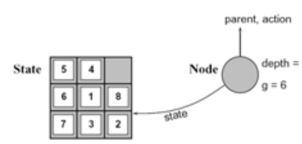
Tree Search

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
      if fringe is empty then return failure
      node ← REMOVE-FRONT(fringe)
      if GOAL-TEST(problem, STATE(node)) then return SOLUTION(node)
      fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
   successors ← the empty set; state ← STATE[node]
   for each action, result in SUCCESSOR-FN(problem, state) do
      s ← a new NODE
      PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
      PATH-COST[s] ← PATH-COST[node]+STEP-COST(state, action, result)
      DEPTH[s] ← DEPTH[node] + 1
      add s to successors
   return successors
```

States vs. Nodes

- Problem graphs have problem states
  - Have successors
- Search trees have search nodes
  - Have parents, children, depth, path cost, etc.
  - Expand uses successor function to create new search tree nodes
  - The same problem state may be in multiple search tree nodes



Summary

- Agents interact with environments through actuators and sensors
  - The agent function describes what the agent does in all circumstances
  - The agent program calculates the agent function
  - The performance measure evaluates the environment sequence
- A perfectly rational agent maximizes expected performance
- PEAS descriptions define task environments
- Environments are categorized along several dimensions:
  - Observable? Deterministic? Episodic? Static? Discrete? Single-agent?
- Problem-solving agents make a plan, then execute it
- State space encodings of problems