Last time: Summary

- Definition of AI?
- Turing Test?
- Intelligent Agents:
  - Anything that can be viewed as
  - perceiving its environment
    - through sensors and
  - acting upon that environment
    - through its effectors to maximize progress towards its goals.
  - Described as a Perception (sequence) to Action
    Mapping: $f: \mathcal{P}^* \rightarrow \mathcal{A}$
  - Using look-up-table, closed form, etc.

Agent Types: Reflex, state-based, goal-based, utility-based

Rational Action: The action that maximizes the expected value of the performance measure given the percept sequence to date
Outline: Problem solving and search

- Introduction to Problem Solving
- Complexity
- Uninformed search
  - Problem formulation
  - Search strategies: depth-first, breadth-first

Outline: Problem solving and search

- Informed search
  - Search strategies: best-first, A*
  - Heuristics
Example: Measuring problem!

- **Problem**: Using these three buckets, measure 7 liters of water.

Problem-Solving Agent

```plaintext
function SIMPLE-PROBLEM-SOLVING-AGENT( p ) returns an action
  inputs: p, a percept
  static: s, an action sequence, initially empty
          state, some description of the current world state
          g, a goal, initially null
          problem, a problem formulation
  state ← UPDATE-STATE(state, p)  //What is the current state?
  if s is empty then
    g ← FORMULATE-GOAL(state)     //From LA to San Diego (given current state)
    problem ← FORMULATE-PROBLEM(state, g)     //e.g. Gas usage
    s ← SEARCH( problem )
  action ← RECOMMENDATION( s, state )
  s ← REMAINDER( s, state )       //If fails to reach goal, update
  return action
```

Note: This is offline problem-solving. Online problem-solving involves acting w/o complete knowledge of the problem and environment.
Example: Buckets

- Measure 7 liters of water using a 3 liter, a 5 liter, and a 9 liter bucket.

**Formulate goal:** Have 7 liters of water in 9-liter bucket

**Formulate problem:**
- States: amount of water in the buckets
- Operators: Fill bucket from source, empty bucket

**Find solution:** sequence of operators that bring you from current state to the goal state

---

Remember (lecture 2): Environment types

<table>
<thead>
<tr>
<th>Environment</th>
<th>Accessible</th>
<th>Deterministic</th>
<th>Episodic</th>
<th>Static</th>
<th>Discrete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Virtual Reality</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes/No</td>
<td>No</td>
<td>Yes/No</td>
</tr>
<tr>
<td>Office Environment</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Mars</td>
<td>No</td>
<td>Semi</td>
<td>No</td>
<td>Semi</td>
<td>No</td>
</tr>
</tbody>
</table>

The environment types largely determine the agent design.

- **Accessible:** The complete states of the environment is accessible.
- **Deterministic:** The next state is determinable from the current state and the action.
Problem types

- **Single-state problem**: deterministic, accessible
  - Agent knows everything about world (the exact state),
  - Can calculate optimal action sequence to reach goal state.
  - *E.g.* playing chess. Any action will result in an exact state

Problem types

- **Multiple-state problem**: deterministic, inaccessible
  - Agent does not know the exact state (could be in any of the possible states)
    - May not have sensor at all
  - Assume states while working towards goal state.
  - *E.g.* walking in a dark room
    - If you are at the door, going straight will lead you to the kitchen
    - If you are at the kitchen, turning left leads you to the bedroom
    - ...
Problem types

- **Contingency problem:**
  - nondeterministic, inaccessible
  - Must use sensors during execution
  - Solution is a tree or policy
  - Often interleave search and execution
  - E.g. A new skater in an arena
    - Sliding problem.
    - Many skaters around

---

Problem types

- **Exploration problem:** unknown state space
  
  Discover and learn about environment while taking actions.
  
  E.g. Maze
Example: Vacuum world

- **Simplified world:** 2 locations, each may or not contain dirt, each may or not contain vacuuming agent.
- **Goal of agent:** clean up the dirt.

**Single-state, start in #5. Solution??**

**Multiple-state, start in \{1, 2, 3, 4, 5, 6, 7, 8\}**
e.g., *Right* goes to \{2, 4, 6, 8\}. **Solution??**

**Contingency, start in #5**
Murphy’s Law: *Stuck* can dirty a clean carpet
Local sensing: dirt, location only. **Solution??**

---

Example: Romania

- **Formulate problem:**
  - states: various cities
  - operators: drive between cities

- **Find solution:**
  - sequence of cities, such that total driving distance is minimized.
Problem formulation

A problem is defined by four items:

- initial state: e.g., “at Arad”

- operators (or successor function $S(x)$)
  e.g., Arad $\rightarrow$ Zerind, Arad $\rightarrow$ Sibiu, etc.

- goal test: can be
  explicit, e.g., $x = “at\text{ Bucharest}”$
  implicit, e.g., $NoDirt(x)$

- path cost (additive)
  e.g., sum of distances, number of operators executed, etc.

A solution is a sequence of operators
leading from the initial state to a goal state
Selecting a state space

- Real world is absurdly complex;

Example: 8-puzzle

- State:
- Operators:
- Goal test:
- Path cost:
Example: 8-puzzle

Why search algorithms?

Back to Vacuum World

states??
operators??
goal test??
path cost??
Example: Robotic Assembly

*states??:* real-valued coordinates of robot joint angles
*operators??:* continuous motions of robot joints
*goal test??:* complete assembly *with no robot included!!!
*path cost??:* time to execute

---

Enter schematics, do not worry about placement & wiring crossing.

Proteo's hierarchical schematic assign features let you take a "bottom up" or "top down" approach depending on your preferred methodology. Proteo can automatically generate sub-sheets based on high-level sheet symbols, or create sheet symbols based on existing sheets.
Real-life example: VLSI Layout

Q: How to put these chips and connection on a board?
Real-life example: VLSI Layout

“optimal way”??

- minimize surface area
- minimize number of signal layers
- minimize number of vias (connections from one layer to another)
- minimize length of some signal lines (e.g., clock line)
- distribute heat throughout board
- etc.

Use automated tools to place components and route wiring.
Search algorithms

Basic idea:
offline, systematic exploration of simulated state-space by generating successors of explored states (expanding)
Search algorithms

**Function** General-Search(problem, strategy) returns a solution, or failure

initialize the search tree using the initial state 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 resulting nodes to the search tree

  end

---

Example: Traveling from Arad To Bucharest
From problem space to search tree

- Some material in this and following slides is from http://www.cs.kuleuven.ac.be/~dannyd/FAI/ check it out!

Problem space

Associated loop-free search tree
Paths in search trees

General search example

Arad
General search example

Implementation of search algorithms

Function General-Search(problem, Queuing-Fn) returns a solution, or failure

nodes $\leftarrow$ make-queue(make-node(initial-state[problem]))

loop do
  if nodes is empty then return failure
  node $\leftarrow$ Remove-Front(nodes)
  if Goal-Test[problem] applied to State(node) succeeds then return node
  nodes $\leftarrow$ Queuing-Fn(nodes, Expand(node, Operators[problem]))
end
**Encapsulating state information in nodes**

A state is a (representation of) a physical configuration. A node is a data structure constituting part of a search tree that includes parent, children, depth, path cost $g(x)$.

States do not have parents, children, depth, or path cost!

The expand function creates new nodes, filling in the various fields and using the Operators (or SuccessorFn) of the problem to create the corresponding states.

---

**Evaluation of search strategies**

- A search strategy is defined by picking the order of node expansion.

- Four evaluation criteria:
  - Completeness:
  - Time complexity:
  - Space complexity:
  - Optimality:
Evaluation of search strategies

- Time and space complexity are measured in terms of:
  - $b$ – max branching factor of the search tree
  - $d$ – depth of the least-cost solution
  - $m$ – max depth of the search tree (may be infinity)

Binary Tree Example

Depth = 0

Depth = 1

Depth = 2

Number of nodes: $n = ?$
Number of levels (max depth) = ?
Complexity

- Why worry about complexity of algorithms?

Complexity: Tower of Hanoi

Figure 11-6  Tower of Hanoi problem with three disks
Complexity: Tower of Honoi

- 3 Disk problem: $2^3 - 1 = 7$ moves
- 64 disk problem: $2^{64} - 1$.
  - $2^{10} = 1024 \approx 1000 = 10^3$,
  - $2^{64} = 2^4 \times 2^{60} \approx 2^4 \times 10^{18} = 1.6 \times 10^{19}$
- One year $\approx 3.2 \times 10^7$. “

Complexity

- How can we evaluate the complexity of algorithms?
  - through asymptotic analysis, i.e., estimate time (or number of operations) necessary to solve an instance of size $n$ of a problem when $n$ tends towards infinity
- See AIMA, Appendix A.
Complexity example: Traveling Salesman Problem

- There are $n$ cities, with a road of length $L_{ij}$ joining city $i$ to city $j$.
- Visit all cities considering:
  - each city is visited only once, and
  - the total route is as short as possible.

Why is exponential complexity “hard”?

It means that the number of operations necessary to compute the exact solution of the problem grows exponentially with the size of the problem (here, the number of cities).

- $\exp(1) = 2.72$
- $\exp(10) = 2.20 \times 10^4$ (daily salesman trip)
Why is exponential complexity “hard”?

- \( \text{exp}(1) \) = 2.72
- \( \text{exp}(10) \) = 2.20 \( \times \) 10^4 (daily salesman trip)
- \( \text{exp}(100) \) = 2.69 \( \times \) 10^43 (monthly salesman planning)
- \( \text{exp}(500) \) = 1.40 \( \times \) 10^217 (music band worldwide tour)
- \( \text{exp}(250,000) \) = 10^{108.573} (fedex, postal services)
- Fastest computer = \( 10^{12} \) operations/second

So...

In general, exponential-complexity problems cannot be solved for any but the smallest instances!
Complexity

- Polynomial-time (P) problems:

  - Example: sort n numbers into increasing order:
    - poor algorithms have $n^2$ complexity,
    - better ones have $n \log(n)$ complexity.

Question:

- Are there algorithms that require more than polynomial time?
  - Yes

  - In particular, exponential-time algorithms are believed to be NP.
Note on NP-hard problems

The formal definition of NP problems:

A problem is **nondeterministic polynomial** if there exists some algorithm that can guess a solution and then verify whether or not the guess is correct in polynomial time.

Complexity and the human brain

- Current computer chip (CPU):
  - $10^3$ inputs (pins)
  - $10^7$ processing elements (gates)
  - 2 inputs per processing element (fan-in = 2)
  - processing elements compute boolean logic (OR, AND, NOT, etc)
Complexity and the human brain

- Typical human brain:
  - $10^7$ inputs (sensors)
  - $10^{10}$ processing elements (neurons)
  - fan-in = $10^3$
  - processing elements compute complicated functions

Still a lot of improvement needed for computers; but computer clusters come close!