CS 380: ARTIFICIAL INTELLIGENCE

PROBLEM SOLVING

9/29/2013
Santiago Ontañón
santi@cs.drexel.edu
https://www.cs.drexel.edu/~santi/teaching/2013/CS380/intro.html
• Last week we learned:
  • What is AI?
  • What is a rational agent

• In the remainder 9 weeks, we will cover:
  • How can rational agents solve problems?
  • How can rational agents represent knowledge?
  • How can rational agents learn?
Outline

• Problems and Problem Solving
• Example Problems
• Solving Problems
Outline

- Problems and Problem Solving
- Example Problems
- Solving Problems
Problem Solving

• What is a problem?
  • In the context of CS/AI: a question that requires an answer

• Many types of problems:
  • Decision problems: yes/no questions
  • Search problems: solutions are sequences
  • Counting problems: counting the number of solutions to a search problem
  • Optimization problems: finding the solution to a search problem that minimizes/maximizes some given criteria.
Problem Solving

• There are many problems for which we know specialized ways of solving them.

• For example:
  • Problem: “find the roots of the polynomial $ax + bx^2 + c = 0$”
  • We know mechanical procedures to solve this problem in an exact way
  • However, those procedures can only be applied to this problem, but not to any other.

• Problem Solving in AI:
  • Finding **general procedures** to solve general classes of problems
  • Is there any algorithm that can be used to solve any computationally solvable problem? (we humans seem to be able to!)
Example: Maze

- A robot is at the entrance of a maze. Problem: how to get to the exit?

- Step 1: formulate goal
  - Be in the exit

- Step 2: formulate problem
  - States, actions, etc.

- Step 3: Find solution
  - Specific sequence of actions
Problem Formulation

• Initial state: $S_0$
  • Initial configuration of the problem (e.g. starting position in a maze)

• Actions: $A$
  • The different ways in which the agent can change the state (e.g. moving to an adjacent position in the maze)

• Goal condition: $G$
  • A function that determines whether a state reached by a given sequence of actions constitutes a solution to the problem or not.

• Cost function: $c$
  • A function that assigns a numeric cost to a given sequence of actions.
Problem Formulation

- **Initial state:** $S_0$
  - Initial configuration of the problem

- **Actions:** $A$
  - The different ways in which the agent can change the state

- **Goal condition:** $G$
  - A function that determines whether a state reached by a given sequence of actions constitutes a solution to the problem or not.

- **Cost function:** $c$
  - A function that assigns a numeric cost to a given sequence of actions.

The initial state, together with the set of actions define the **State Space**, or Search Space (the set of all the possible states that are reachable from the initial state by some sequence of actions).
Problem Formulation

• Initial state: $S_0$
  • Initial configuration of the problem (e.g. starting position in a maze)

• Actions: $A$
  • The different ways in which the agent can change the state (e.g. moving to an adjacent position in the maze)

• Goal condition: $G$
  • A function that determines whether a state reached by a given sequence of actions constitutes a solution to the problem or not.

• Cost function: $c$
  • A function that assigns a numeric cost to a given sequence of actions.

This should remind you of the PEAS rational agent environment definition we discussed last week.
Problem Types

• Deterministic, fully observable:
  • The agent knows which state it is in.
  • Solution is a sequence of actions (a plan)

• Non-observable:
  • The agent does not know the current state
  • Solution, if it exists, is a sequence

• Non-deterministic or partially observable:
  • Percepts give an agent information about the current state, but not enough to determine it completely
  • Solution is a policy or contingent plan
Problem Types: Example

- Deterministic, fully observable:
  - Solution: [down, down, down, right, down, …]

- Non-observable:
  - Is there any sequence of movements that takes us to the exit no matter what is the starting point?

- Non-deterministic or partially observable:
  - Agent might now know its exact coordinates, percepts only tell them about the wall configuration around
  - Solution, policy (mapping from percepts, or states, to actions):
    - If no wall on left then turn left
    - If wall on left, no wall ahead, then advance
    - If wall on left, wall ahead, then turn right
Plan vs Policy

- **Plan:**
  - Specific sequence of actions:
    - left,
    - right,
    - up,
    - up

- **Contingent plan:**
  - Sequence of actions with conditionals:
    - left,
    - right,
    - If wall ahead then up else right

- **Policy:**
  - Mapping from percepts to actions:

<table>
<thead>
<tr>
<th>Percept</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>No walls</td>
<td>advance</td>
</tr>
<tr>
<td>Wall left</td>
<td>advance</td>
</tr>
<tr>
<td>Wall right</td>
<td>left</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Walls in all directions</td>
<td>No-op</td>
</tr>
</tbody>
</table>
Outline

• Problems and Problem Solving
• Example Problems
• Solving Problems
Example: Maze

- **States:**
- **Actions:**
- **Goal:**
- **Cost:**
Example: Maze

- **States:**
  - (x,y) coordinate of robot + facing direction
  - Initial state: (0,0), down

- **Actions:** advance, left, right

- **Goal:** reach state ((9,4), *)

- **Cost:** 1 per action executed
Example: Maze

- **States:**
  - \((x,y)\) coordinate of robot + facing direction
  - Initial state: \(((0,0), \text{down})\)
- **Actions:** advance, left, right
- **Goal:** reach state \(((9,4),*)\)
- **Cost:** 1 per action executed

How difficult is a problem?

A typical measure of “difficulty” is the size of the state space (if it’s small, it’s very easy to find the solution).

For this maze here, the state space has 200 states (50 positions * 4 directions)

Chess has about \(10^{50}\) states
Example: 8-puzzle

- States:
- Actions:
- Goal:
- Cost:
Example: 8-puzzle

- **States:**
  - Board configuration, represented as a vector of 9 positions (each position represents one of the cells in the board)
  - Initial state: (8, blank, 6, 5, 4, 7, 2, 3, 1)
- **Actions:**
- **Goal:**
- **Cost:**
Example: 8-puzzle

- **States:**
  - Board configuration, represented as a vector of 9 positions (each position represents one of the cells in the board)
  - Initial state: (8, blank, 6, 5, 4, 7, 2, 3, 1)
- **Actions:**
  - move blank up, down, left, right
- **Goal:**
- **Cost:**
Example: 8-puzzle

- **States:**
  - Board configuration, represented as a vector of 9 positions (each position represents one of the cells in the board)
  - Initial state: (8,blank,6,5,4,7,2,3,1)
- **Actions:**
  - move blank up, down, left, right
- **Goal:**
  - Reach state (blank,1,2,3,4,5,6,7,8)
- **Cost:**
  - 1 per move
Example: 8-puzzle

- **States:**
  - Board configuration, represented as a vector of 9 positions (each position represents one of the cells in the board)
  - Initial state: (8, blank, 6, 5, 4, 7, 2, 3, 1)
- **Actions:**
  - move blank up, down, left, right
- **Goal:**
  - Reach state (blank, 1, 2, 3, 4, 5, 6, 7, 8)
- **Cost:**
  - 1 per move

What’s the size of the state space for the 8-puzzle?
Example: 8-puzzle

- **States:**
  - Board configuration, represented as a vector of 9 positions (each position represents one of the cells in the board)
  - Initial state: (8, blank, 6, 5, 4, 7, 2, 3, 1)

- **Actions:**
  - move blank up, down, left, right

- **Goal:**
  - Reach state (blank, 1, 2, 3, 4, 5, 6, 7, 8)

- **Cost:**
  - 1 per move

What’s the size of the state space for the 8-puzzle?
9! = 362880
Example: Vacuum World

- **States:**
- **Actions:**
- **Goal:**
- **Cost:**
Example: Vacuum World

- **States:**
  - Position of robot + dirt in A? + dirt in B?
  - Initial state: (A, no, yes)
- **Actions:**
- **Goal:**
- **Cost:**
Example: Vacuum World

- **States:**
  - Position of robot + dirt in A? + dirt in B?
  - Initial state: (A, no, yes)
- **Actions:**
  - Left, right, suck, no-op
- **Goal:**
- **Cost:**
Example: Vacuum World

- **States:**
  - Position of robot + dirt in A? + dirt in B?
  - Initial state: (A, no, yes)

- **Actions:**
  - Left, right, suck, no-op

- **Goal:**
  - (-, no, no)

- **Cost:**
Example: Vacuum World

- **States:**
  - Position of robot + dirt in A? + dirt in B?
  - Initial state: (A, no, yes)

- **Actions:**
  - Left, right, suck, no-op

- **Goal:**
  - (-,no,no)

- **Cost:**
  - 1 per move, 1 per suck, 0 per no-op
Example: Vacuum World

- States: integer dirt and robot locations (ignore dirt amounts etc.)
- Actions: Left, Right, Suck, NoOp
- Goal test: no dirt
- Path cost
Example: Fox, Goat, Cabbage

- States
- Actions
- Goal
- Cost
Example: Fox, Goat, Cabbage

- States:
  - Location of fox, goat, cabbage (left bank, boat, right bank) and boat (left, right)
  - Initial state: (left, left, left, left)

- Actions
- Goal
- Cost
Example: Fox, Goat, Cabbage

- **States:**
  - Location of fox, goat, cabbage (left bank, boat, right bank) and boat (left, right)
  - Initial state: (left, left, left, left)

- **Actions:**
  - Boat left, boat right
  - Load fox, unload fox
  - Load goat, unload goat
  - Load cabbage, unload cabbage
  - (never leaving fox with goat, goat with cabbage alone)

- **Goal:**

- **Cost:**
Example: Fox, Goat, Cabbage

- **States:**
  - Location of fox, goat, cabbage (left bank, boat, right bank) and boat (left, right)
  - Initial state: (left, left, left, left)

- **Actions:**
  - Boat left, boat right
  - Load fox, unload fox
  - Load goat, unload goat
  - Load cabbage, unload cabbage
  - (never leaving fox with goat, goat with cabbage alone)

- **Goal:**
  - (right, right, right, right)

- **Cost:**
Example: Fox, Goat, Cabbage

- **States:**
  - Location of fox, goat, cabbage (left bank, boat, right bank) and boat (left, right)
  - Initial state: (left, left, left, left)

- **Actions:**
  - Boat left, boat right
  - Load fox, unload fox
  - Load goat, unload goat
  - Load cabbage, unload cabbage
  - (never leaving fox with goat, goat with cabbage alone)

- **Goal:**
  - (right, right, right, right)

- **Cost:**
  - 1 per action
Example: Fox, Goat, Cabbage

- **States:**
  - Location of fox, goat, cabbage (left bank, boat, right bank) and boat (left, right)
  - Initial state: (left, left, left, left)

- **Actions:**
  - Boat left, boat right
  - Load fox, unload fox
  - Load goat, unload goat
  - Load cabbage, unload cabbage
  - (never leaving fox with goat, goat with cabbage alone)

- **Goal:**
  - (right, right, right, right)

- **Cost:**
  - 1 per action

What’s the size of the state space?
Example: Fox, Goat, Cabbage

• States:
  • Location of fox, goat, cabbage (left bank, boat, right bank) and boat (left, right)
  • Initial state: (left, left, left, left)

• Actions:
  • Boat left, boat right
  • Load fox, unload fox
  • Load goat, unload goat
  • Load cabbage, unload cabbage
  • (never leaving fox with goat, goat with cabbage alone)

• Goal:
  • (right, right, right, right)

• Cost:
  • 1 per action

What’s the size of the state space? At most $3 \times 3 \times 3 \times 2 = 54$ (less when we apply the action constraints)
More Examples

- Search Problems are far more general than it might seem.
- For example, we can formulate any of the following as a search problem:
  - Proving a theorem
  - Solving a physics problem
  - Finding your way around a city (path finding)
  - Playing individual sports (multiplayer sports and games belong to another category of problems, we will discuss in a few weeks).
  - Playing a computer game
  - Etc.
Outline

• Problems and Problem Solving
• Example Problems
• Solving Problems
Solving Problems

• Let us only consider deterministic, fully-observable problems

• Simple algorithm:
  • Enumerate every possible sequence of actions:
    • [left]
    • [right]
    • [advance]
    • [left,left]
    • [left,right]
    • [left,advance]
    • ...
  • Try one by one, until one reaches the solution
Solving Problems

- General idea:
  - Given the state space
  - Given the initial state

- Explore the state space until a state that matches the goal condition is found.

- Different problem solving methods are actually just different search strategies.
Random Walk

• Algorithm:
  
  • Initial state: \( S \)
  • 1) Determine set of applicable actions \( A = \text{validActions}(S) \)
  • 2) Choose an action \( a \) in \( A \) at random
  • 3) Apply \( a \) to the current state: \( S = \text{apply}(S, a) \)
  • 4) If we have not reached the goal (\( G(S) == \text{false} \)), go to 1
  • 5) done

• This algorithm:
  • Does not ensure finding a solution
  • Does not ensure finding the optimal solution (minimal cost)
Next Class

- Classic problem solving strategies