Problem Solving Agents & Problem Formulation

AIMA 2.3, 3.1-3

Outline for today’s lecture

- Defining Task Environments (AIMA 2.3)
- Environment types
- Formulating Search Problems
- Search Fundamentals

Task environments

- To design a rational agent we need to specify a task environment
  - a problem specification for which the agent is a solution

  **PEAS:** to specify a task environment
  - Performance measure
  - Environment
  - Actuators
  - Sensors

PEAS: Specifying an automated taxi driver

- **Performance measure:**
  - safe, fast, legal, comfortable, maximize profits
- **Environment:**
  - roads, other traffic, pedestrians, customers
- **Actuators:**
  - steering, accelerator, brake, signal, horn
- **Sensors:**
  - cameras, sonar, speedometer, GPS

PEAS: Medical diagnosis system

- **Performance measure:** Healthy patient, minimize costs, lawsuits
- **Environment:** Patient, hospital, staff
- **Actuators:** Screen display (form including: questions, tests, diagnoses, treatments, referrals)
- **Sensors:** Keyboard (entry of symptoms, findings, patient’s answers)
Outline for today's lecture

- Defining Task Environments
- Environment types (also AIMA 2.3)
- Formulating Search Problems
- Search Fundamentals

Environment types: Definitions I

- 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.
  - If the environment is deterministic except for the actions of other agents, then the environment is Strategic.
- Episodic (vs. sequential): The agent's experience is divided into atomic "episodes" during which the agent perceives and then performs a single action, and the choice of action in each episode depends only on the episode itself.

Environment types: Definitions II

- Static (vs. dynamic): The environment is unchanged while an agent is deliberating.
  - The environment is semidynamic if the environment itself does not change with the passage of time but the agent's performance score does.
- Discrete (vs. continuous): A limited number of distinct, clearly defined percepts and actions.
- Single agent (vs. multiagent): An agent operating by itself in an environment.

(See examples in AIMA, however I don't agree with some of the judgments)

Environment Restrictions for Now

- We will assume environment is
  - Static
  - Fully Observable
  - Deterministic
  - Discrete

The rational agent designer's goal

- Goal of AI practitioner who designs rational agents: given a PEAS task environment,
  1. Construct agent function \( f \) that maximizes (the expected value of) the performance measure,
  2. Design an agent program that implements \( f \) on a particular architecture

Outline for today's lecture

- Defining Task Environments
- Environment types
- Formulating Search Problems (AIMA, 3.1-3.2)
- Search Fundamentals
Example search problem: 8-puzzle

- **Formulate goal**
  - Pieces to end up in order as shown...

- **Formulate search problem**
  - **States**: configurations of the puzzle (9! configurations)
  - **Actions**: Move one of the movable pieces (≤4 possible)
  - **Performance measure**: minimize total moves

- **Find solution**
  - Sequence of pieces moved: 3,1,6,3,1,...

Example search problem: holiday in Romania

Holiday in Romania II

- On holiday in Romania; currently in Arad
  - Flight leaves tomorrow from Bucharest

- **Formulate goal**
  - Be in Bucharest

- **Formulate search problem**
  - **States**: various cities
  - **Actions**: drive between cities
  - **Performance measure**: minimize distance

- **Find solution**
  - Sequence of cities; e.g. Arad, Sibiu, Fagaras, Bucharest, ...

More formally, a problem is defined by:

1. A set of states $S$
2. An initial state $s_i \in S$
3. A set of actions $A$
   - $\forall s \in S, \text{Actions}(s) = \text{the set of actions that can be executed in } s$
   - that are applicable in $s$
4. Transition Model: $\forall s, a \in \text{Actions}(s), \text{Result}(s, a) \rightarrow s_r$
   - $s_r$ is called a successor of $s$
   - $\text{Successors}(s) = s_r$
   - $\text{Successors}(s) \in \text{state space}$
5. **Goal test** $\text{Goal}(s)$
   - Can be implicit, e.g. checkmate($x$)
   - $s$ is a goal state if $\text{Goal}(x)$ is true
6. **Path cost** (additive)
   - e.g. sum of distances, number of actions executed, ...
   - $c(x, a, y)$ is the step cost, assumed $\geq 0$
   - (where action $a$ goes from state $x$ to state $y$)

Solution

A **solution** is a sequence of actions from the **initial state** to a **goal state**.

**Optimal Solution:**
A solution is **optimal** if no solution has a lower path cost.

Hard subtask: Selecting a state space

- Real world is absurdly complex
  - State space must be abstracted for problem solving
- (abstract) **State** = set (equivalence class) of real world states
- (abstract) **Action** = equivalence class of combinations of real world actions
  - e.g. Arad $\rightarrow$ Zerind represents a complex set of possible routes, detours, rest stops, etc
  - The abstraction is valid if the path between two states is reflected in the real world
- Each abstract action should be “easier” than the real problem
Art: Formulating a Search Problem

Decide:

- Which properties matter & how to represent
  - Initial State, Goal State, Possible Intermediate States
- Which actions are possible & how to represent
  - Operator Set: Actions and Transition Model
- Which action is next
  - Path Cost Function

Formulation greatly affects combinatorics of search space and therefore speed of search.

Example: 8-puzzle

- States??
- Initial state?? [7,2,4,5,-,6,8,3,1]
- Actions?? {Left, Right, Up, Down}
- Transition Model?? ...
- Goal test?? Check if goal configuration is reached
- Path cost?? Number of actions to reach goal

Example: Missionaries & Cannibals

Three missionaries and three cannibals come to a river. A rowboat that seats two is available. If the cannibals ever outnumber the missionaries on either bank of the river, the missionaries will be eaten. (AIMA problem 3.9)

How shall they cross the river?

Formulation: Missionaries & Cannibals

- How to formalize:
  - Initial state: all M, all C, and boat on one bank
  - Actions: ??
  - Transition Model??
  - Goal test: True if all M, all C, and boat on other bank
  - Cost: ??

Remember:

- Representation:
  - States: Which properties matter & how to represent
  - Actions & Transition Model: Which actions are possible & how to represent
  - Path Cost: Deciding which action is next

Missionaries and Cannibals

States: (CL, ML, BL)

<table>
<thead>
<tr>
<th>Initial</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>331</td>
<td>000</td>
</tr>
</tbody>
</table>

Actions:

<table>
<thead>
<tr>
<th>Travel Across</th>
<th>Travel Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>-101</td>
<td>101</td>
</tr>
<tr>
<td>-201</td>
<td>201</td>
</tr>
<tr>
<td>-011</td>
<td>011</td>
</tr>
<tr>
<td>-021</td>
<td>021</td>
</tr>
<tr>
<td>-111</td>
<td>111</td>
</tr>
</tbody>
</table>
Outline for today's lecture

- Defining Task Environments
- Environment types
- Formulating Search Problems
- Search Fundamentals (AIMA 3.3)

Useful Concepts

- **State space**: the set of all states reachable from the initial state by *any* sequence of actions
  - When several operators can apply to each state, this gets large very quickly
  - Might be a proper subset of the set of configurations
- **Path**: a sequence of actions leading from one state $s_i$ to another state $s_k$
- **Frontier**: those states that are available for expanding (for applying legal actions to)
- **Solution**: a path from the initial state $s_i$ to a state $s_f$ that satisfies the goal test

Basic search algorithms: *Tree Search*

- Generalized algorithm to solve search problems (Review from CIS 121)
  - Enumerate in some order all possible paths from the initial state
  - Here: search through explicit tree generation
  - Nodes in search tree generated through transition model
  - In general search generates a graph (same state through multiple paths), but we'll just look at trees in lecture
  - Tree search treats different paths to the same node as distinct

Review (CIS 121): Generalized tree search

function TREE-SEARCH(problem, strategy) return a solution or failure
   Initialize frontier to the initial state
   do
      if the frontier is empty then return failure
      choose leaf node for expansion according to strategy & remove from frontier
      if node contains goal state then return solution
      else expand the node and add resulting nodes to the frontier
   end

8-Puzzle: States and Nodes

- A *state* is a (representation of a) physical configuration
- A *node* is a data structure constituting part of a search tree
  - Also includes parent, children, depth, path cost $g(s)$
  - Here node = $\langle$state, parent-node, children, action, path-cost, depth$\rangle$
- States do not have parents, children, depth or path cost!

<table>
<thead>
<tr>
<th>State</th>
<th>Node</th>
<th>Action= Up</th>
<th>Cost = 6</th>
<th>Depth = 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 2 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 6 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 3 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The EXPAND function
  - uses the Actions and Transition Model to create the corresponding states
  - creates new nodes

8-Puzzle Search Tree

- (Nodes show state, parent, children - leaving Action, Cost, Depth implicit)
- Suppressing useless "backwards" moves
Problem: Repeated states

- Failure to detect repeated states can turn a linear problem into an exponential one!

Solution: Graph Search!

- Graph search — Optimal but memory inefficient

Graph Search vs Tree Search

| Function TREE-SEARCH (problem) returns a solution, or failure |
|-------------|--------------------------------------------------|
| Initialization the frontier using the initial state of problem |
| loop do |
|     if the frontier is empty then return failure |
|     choose a leaf node and remove it from the frontier |
|     expand the chosen node, adding the resulting nodes to the frontier |
| end loop |

<table>
<thead>
<tr>
<th>Function GRAPH-SEARCH (problem) returns a solution, or failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize the explored set by empty</td>
</tr>
<tr>
<td>loop do</td>
</tr>
<tr>
<td>if the frontier is empty then return failure</td>
</tr>
<tr>
<td>choose a leaf node and remove it from the frontier</td>
</tr>
<tr>
<td>if the node contains a goal state then return the corresponding solution</td>
</tr>
<tr>
<td>add the node to the explored set</td>
</tr>
<tr>
<td>expand the chosen node, adding the resulting nodes to the frontier</td>
</tr>
<tr>
<td>end loop</td>
</tr>
</tbody>
</table>

Figure 3.7: An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH method in bold fade are the additions needed to handle repeated states.