Fantana muzicala Arad - Musical fountain Arad, Romania

Problem Solving Agents & Problem Formulation

AIMA 3.1, 3.2

Outline for today's lecture
- Defining Task Environments
- Environment types
- Formulating Search Problems

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:
- ?

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
- Formulating Search Problems

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)

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

Environment Restrictions for Now

- We will assume environment is
  - **Static**
  - **Fully Observable**
  - **Deterministic**
  - **Discrete**
Outline for today's lecture

- Defining Task Environments
- Environment types
- Formulating Search Problems (AIMA, 3.1-3.2)

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

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

- Formulate goal
  - Be in Bucharest

- Formulate a 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 \( A \)
   - \( \forall s \), \( \text{Actions}(s) = \) the set of actions that can be executed in \( s \), that are applicable in \( s \).
4. Transition Model: \( \forall s \forall a \in \text{Actions}(s), \text{Result}(s, a) \rightarrow s_f \)
   - \( s_f \) is called a successor of \( s \)
   - \( \forall s \), \( \text{Successors}(s) * = \) state space
5. Goal test \( \text{Goal}(s) \)
   - Can be implicit, e.g. \( \text{checkmate}(s) \)
   - \( s \) is a goal state if \( \text{Goal}(s) \) is true
6. Path cost (additive)
   - \( e(x,y) \) 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** = complex combination of real world actions
  - e.g. Arad → 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
- (abstract) **Solution** = set of abstract paths that are solutions in the abstract space
- Each abstract action should be “easier” than the real problem

**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

**Example: 8-puzzle**

- States??
- Initial state??
- Actions??
- Transition Model??
- Goal test??
- Path cost??

**Example: 8-puzzle**

- States?? List of 9 locations - e.g., [7,2,4,5,-,6,8,3,1]
- 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. (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)
Initial 331  Goal 000

Actions:
Travel Across  Travel Back
-101  101
-201  201
-011  011
-021  021
-111  111

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.
- 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

Search Fundamentals

AIMA 3.3

Basic search algorithms: Tree Search

- How do we find the solutions for the previous problem formulations? (Review from CIS 121)
  - Enumerate in some order all possible paths from the initial state
  - Here: search through explicit tree generation
    - ROOT= initial state.
    - Nodes and leafs generated through transition model
  - In general search generates a graph (same state through multiple paths), but we’ll just look at trees in lecture
    - Treats different paths to the same node as distinct
  - (Extension to graphs: just check to see if any node has been visited before adding to search queue)

Review (CIS 121): Generalized tree search

function TREE-SEARCH(problem, strategy) return a solution or failure
Initialize frontier to the initial state of the problem
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

Determines search process?
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(x)
  - Here node = <state, parent node, action, path-cost, depth>
- States do not have parents, children, depth or path cost!

The EXPAND function
- Uses the Actions and Transition Model to create the corresponding states
  - Creates new nodes, fills in the various fields

8-Puzzle Search Tree

- Leaving Action, Cost, Depth implicit
- Ignores “backwards” moves

Search Strategies

- **Strategy** = order of expansion
- Dimensions for evaluation
  - Completeness: Always find the solution?
  - Time complexity: # of nodes generated
  - Space complexity: # of nodes in memory
  - Optimality: Finds a least cost solution (lowest path cost)?

Time/space complexity measurements
- b, maximum branching factor of search tree
- d, depth of the shallowest goal node
- m, maximum length of any path in the state space (potentially ∞)

Uninformed search strategies

- (a.k.a. blind search) = use only information available in problem definition.
  - When strategies can determine whether one non-goal state is better than another → informed search.

Categories defined by expansion algorithm:
- Review: Breadth-first search
- Review: Depth-first search
- (Depth-limited search)
- Iterative deepening search
- Uniform-cost search
- Bidirectional search

Review: Breadth-first search

- **Idea:**
  - Expand shallowest unexpanded node

- **Implementation:**
  - frontier is FIFO (First-In-First-Out) Queue:
    - Put successors at the end of frontier successor list.
Breadth-first search (simplified)

function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
node ← a node with STATE = problem.INITIAL-STATE, PATH-COST=0
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
frontier ← a FIFO queue with node as the only element
loop do
if EMPTY?(frontier) then return failure
node ← POP(frontier) // chooses the shallowest node in frontier
add node.STATE to explored
for each action in problem.ACTIONS(node.STATE) do
child ← CHILD-NODE(problem, node, action)
if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
frontier ← INSERT(child, frontier)

From Figure 3.11 Breadth-first search (ignores loops, repeated nodes)

Introduction to space complexity

- You know about:
  - "Big O" notation
  - Time complexity
- Space complexity is analogous to time complexity
  - Units of space are arbitrary
  - Doesn’t matter because Big O notation ignores constant multiplicative factors
  - Space units:
    - One Memory word
    - Size of fixed Data structure

Properties of breadth-first search

- Complete?
- Time Complexity?
- Space Complexity?
- Optimal?

Properties of breadth-first search

- Complete? Yes (if \( b \) is finite)
- Time? \( 1 + b + b^2 + b^3 + \ldots + b^d = O(b^d) \)
- Space? \( O(b^d) \) (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step) (not optimal in general)

Exponential Space/Time Not Good...

- Exponential complexity search problems cannot be solved by uninformed search methods for any but the smallest instances.
- (Memory requirements are a bigger problem than execution time.)

<table>
<thead>
<tr>
<th>DEPTH</th>
<th>NODES</th>
<th>TIME</th>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>110</td>
<td>0.11 milliseconds</td>
<td>107 kilobytes</td>
</tr>
<tr>
<td>4</td>
<td>11110</td>
<td>11 milliseconds</td>
<td>10.6 megabytes</td>
</tr>
<tr>
<td>6</td>
<td>( 10^6 )</td>
<td>1.1 seconds</td>
<td>1 gigabyte</td>
</tr>
<tr>
<td>8</td>
<td>( 10^8 )</td>
<td>2 minutes</td>
<td>183 gigabytes</td>
</tr>
<tr>
<td>10</td>
<td>( 10^{10} )</td>
<td>3 hours</td>
<td>10 terabytes</td>
</tr>
<tr>
<td>12</td>
<td>( 10^{12} )</td>
<td>13 days</td>
<td>1 petabyte</td>
</tr>
<tr>
<td>14</td>
<td>( 10^{14} )</td>
<td>3.5 years</td>
<td>99.9 petabytes</td>
</tr>
</tbody>
</table>

Fig 3.13 Assumes \( b=10 \), 1M nodes/sec, 1000 bytes/node

Review: Depth First Search
Depth-first search

- **Idea:**
  - Expand *deepest* unexpanded node

- **Implementation:**
  - *frontier* is LIFO (Last-In-First-Out) Queue:
    - Put successors at the front of *frontier* successor list.

Uninformed search strategies

- **Categories defined by** expansion algorithm:
  - Breadth-first search: FIFO queue
  - Depth-first search: LIFO queue
  - (Depth-limited search)
  - Iterative deepening search
  - Uniform-cost search
  - Bidirectional search

Properties of depth-first search

- **Complete?** No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path
    - complete in finite spaces
- **Time?** \( O(b^d) \): terrible if \( m \) is much larger than \( d \)
  - But if solutions are dense, may be much faster than breadth-first
- **Space?** \( O(b^m) \), i.e., linear space!
- **Optimal?** No

Depth-first vs Breadth-first

- **Use depth-first if**
  - Space is restricted
  - There are many possible solutions with long paths and wrong paths can be detected quickly
  - Search can be fine-tuned quickly
- **Use breadth-first if**
  - Possible infinite paths
  - Some solutions have short paths
  - Can quickly discard unlikely paths